POLITECNICO DI BARI
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE – BARI
CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

TESI DI LAUREA
IN
SISTEMI E APPLICAZIONI INFORMATICHE

SVILUPPO DI UN SISTEMA PER IL MONITORAGGIO DELLE CONDIZIONI DEL MANTO STRADALE MEDIANTE L'UTILIZZO DI SENSORI NEGLI SMARTPHONE

Relatore:
Chiar.mo Prof. Ing. Filippo Attivissimo

Correlatori:
Chiar.mo Prof. Ing. Attili Di Nisio
Chiar.mo Prof. Ing. Maurizio Spadavecchia

Laureando: Giuseppe Murgolo

ANNO ACCADEMICO 2020/2021

Sommario

Ringraziamenti

Desidero ringraziare innanzitutto il mio relatore, il professor Filippo Attivissimo, e correlatori, il professor Attilio Di Nisio ed ingegner Maurizio Spadavecchia, per la grandissima disponibilità e fiducia che mi è stata data fin dal primo momento nella creazione di questa tesi. Grazie ai loro consigli ed aiuti sono stato in grado di raggiungere il traguardo prefissato potendo essere più che fiero del lavoro che ho svolto.

Dopo tre estenuanti anni posso finalmente guardare all'intero tracciato che mi ha portato a questo punto. Non poche sono state le fatiche e difficoltà che ho incontrato, accompagnate da personali trionfi e vittorie che mi hanno reso appagato durante la sfida del raggiungimento di questo traguardo. Non sempre ho visto la luce ma molte persone mi hanno sopportato, aiutato e sostenuto affinché non cadessi, e se fossi caduto a rialzarmi per non fermarmi.

Desidero ringraziare di cuore i miei genitori che, fin da quando ero bambino, mi hanno sostenuto nelle mie scelte lasciandomi la completa libertà di poter scegliere la mia strada di vita. Avere dei genitori come loro è probabilmente il primo miglior regalo che il destino mi abbia mai fatto.

Desidero ringraziare per il grandissimo aiuto fornito nella stesura di questa tesi i miei amici Antonio Tornesello e Romy Ricci. La qualità del mio lavoro non sarebbe stata la stessa senza i loro consigli ed aiuti che mi hanno portato a raffinarlo di non poco.

E per ultimi ma per nessun modo meno importanti, desidero ringraziare le persone che non considero più solo come dei semplici amici, perché sarebbe troppo diminutivo, ma come una vera e propria seconda famiglia che mi è stata regalata dal destino. Nel corso degli anni ho avuto la gioia di poterli conoscere e si sono rivelate delle splendide persone che mi hanno accettato per quel che sono, con cui scherzo e con cui ho condiviso e condividerò altri splendidi momenti insieme. Non di poco conto è stato l'aiuto di probabilmente la mia più cara amica Mela, è stata una di quelle pochissime persone ad avermi dato i miei tempi, ad avermi accettato per come sono e a permettermi di aprirmi sempre di più con gli altri combattendo insieme molte delle mie insicurezze.

Introduzione

Grazie ai sensori implementati nella stragrande maggioranza degli ***smartphone*** odierni, è possibile elaborare dati grezzi ed ottenere molte informazioni. Essi possono essere utilizzati per studi statistici, implementare funzionalità quali il rilevamento di onde sismiche per terremoti imminenti, rilevamento degli incidenti automobilistici con uscita istantanea degli airbag.

Verrà realizzata un'applicazione, con relativo sviluppo documentato, che cercherà di determinare lo stato di una strada cercando di stabilire se essa è particolarmente dissestata o meno attraverso i dati generati dall'app. I dati verranno raccolti dall'applicazione, la quale verrà installata su degli smartphone test, immagazzinati su un server ed elaborati cercando possibili informazioni e correlazioni utili.

Lo studio sarà possibile grazie ad un particolare sensore chiamato ***accelerometro,*** il quale permette di misurare le vibrazioni e accelerazioni improvvise in correlazione con le caratteristiche delle strade prese in esame.

Lo scopo principale di questa tesi è di determinare il livello di utilità delle informazioni fornite dal sensore in questione nell'ambito di monitoraggio delle strade; dopo una breve introduzione su argomenti chiave, quali nozioni sul sistema operativo, un'analisi di mercato per la scelta del sistema più adatto e la sua descrizione, verrà documentato lo sviluppo dell'applicazione, con relative scelte in ambito di tecniche e tecnologie utilizzate. Infine, con i dati raccolti dall'applicazione verrà affrontato uno studio che determinerà l'effettiva utilità dei dati in un contesto reale.

I motivi che spingono alla stesura della tesi si basano su un effettivo riscontro reale delle informazioni che possono essere ricavate da un comune smartphone, strumento essenziale nella vita di moltissime persone. Essere in grado di sfruttare tale dispositivo al massimo delle proprie funzionalità e capacità consentirebbe uno sviluppo in svariati campi e nel caso di questa tesi a mappare ed evidenziare punti critici delle strade.

Capitolo 1

Studio sul sistema operativo mobile

1.1. *Sistema operativo mobile*

Data la natura dell'applicazione sviluppata, è necessario fornire nozioni generali sui sistemi operativi mobili al fine esplicitare alcune delle principali caratteristiche in comune ai diversi sistemi, verranno inoltre aggiunti requisiti che dovranno essere rispettati anche in seguito. Infine, con opportuni studi del mercato e di altri fattori, verrà scelto il sistema operativo presumibilmente migliore su cui si è basato lo sviluppo e l'esecuzione dell'applicazione.

Un sistema operativo mobile è un particolare tipo di *OS*, *Operating System*, progettato per un dispositivo mobile che secondo la pubblicazione della *NIST*, istituto nazionale (statunitense) per gli standard e la tecnologia (*National Institute of Standards and Technology*), viene definito come:

> *"Un dispositivo di computazione portatile che:*
>
> *(i) ha un piccolo formato tale che sia facile per essere portato da singoli individui;*
>
> *(ii) è progettato per operare senza una connessione fisica (exempli gratia, trasmettere o riceve informazione in modalità wireless);*
>
> *(iii) possiede memorie locali per dati, removibili o non removibili;*
>
> *(iv) include una sorgente di energia autonoma. […]"* [1]

[1] Security and Privacy Controls for Federal Information Systems and Organizations (2020)

In generale un OS mobile si comporta come un qualsiasi sistema operativo poiché deve essere in grado di poter garantire:

- Controllo e gestione delle risorse, quali CPU e memoria primaria;
- Accesso ai diversi componenti hardware che compongono il dispositivo;
- Esecuzione ed assegnazione delle risorse ai diversi programmi;
- Gestione dell'archiviazione dei dati su memoria di massa da parte dei programmi.

Inoltre, per gli OS mobili è di particolare importanza che venga gestita un'interfaccia grafica per favorire un'interazione semplice tra dispositivo e utente, che segue un approccio differente rispetto a quello utilizzato in un OS desktop o per macchine che fungono da server. I sistemi operativi mobili più conosciuti ed utilizzati sono Android e iOS, ma ne sono anche presenti altri meno diffusi che ricoprono circa 1-2% del mercato.

1.1.1 Differenze tra OS e OS mobili

In generale, quando si parla delle varie differenze tra sistemi operativi a confronto con quelli mobili s'intendono quasi sempre caratteristiche che interessano le architetture utilizzate, le varie funzionalità ed i metodi di ottimizzazione per l'utilizzo delle risorse e del consumo energetico del dispositivo.

L'architettura utilizzata negli smartphone è chiamata *Advanced RISC Machines*, abbreviato in ARM, utilizzata per la realizzazione di *sistemi embedded*[2], ovvero sistemi per dispositivi che svolgono funzioni per uno scopo ben preciso, quali appunto gli smartphone, che limitano il numero di risorse che posso essere integrate nel sistema ma in grado di poter raggiungere delle buone prestazioni in termini di consumi energetici generalmente bassi.

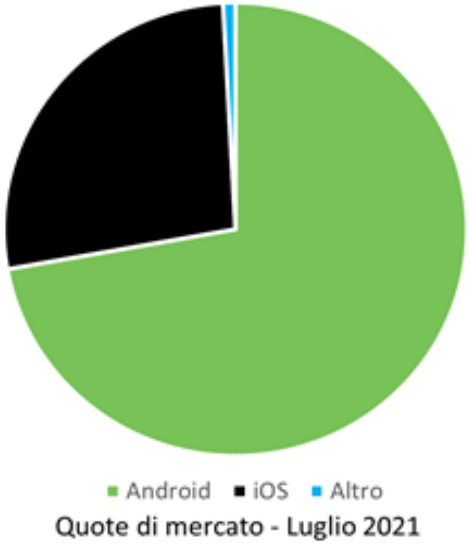
Data la natura architetturale degli smartphone, è importante sottolineare il ruolo che ricopre l'accesso e l'uso delle risorse disponibili nel sistema, questo perché generalmente sono molto limitate rispetto alle risorse di una macchina server o computer che permettono al rispettivo OS di eseguire molte più funzionalità in background, ovvero l'esecuzione di programmi non visibili dall'utente generalmente di gestione del sistema. Diventa fondamentale limitare il numero di operazioni che un OS mobile deve svolgere, quindi utilizzare la minor quantità di risorse possibili per massimizzare il risultato finale.

[2] Per approfondimenti: Collins COBUILD Advanced Learner's Dictionary (2021)

1.1.2 Analisi dei diversi sistemi operativi mobili

Poiché l'applicazione di raccolta dati dovrà essere necessariamente scritta in un linguaggio di programmazione tale da poter essere eseguita sul sistema operativo mobile target, diventa necessario uno studio sull'OS mobile più adatto per lo sviluppo. Lo studio verrà concentrato su una analisi del mercato e della portabilità degli OS.

1.1.3 Analisi del mercato



Dai dati forniti dal sito *statcounter.com*[3] riguardanti le quote del mercato dei diversi OS mobili, risulta abbastanza evidente che i principali sistemi operativi mobili presenti in commercio sono Android e iOS, essi compongo circa il 99% del mercato attuale con uno scarto di circa 1% per sistemi operativi poco conosciuti o che negli anni non sono stati in grado di soddisfare la richiesta del mercato.

Risulta palese a questo punto che per quanto riguarda la scelta del sistema operativo mobile ci potremo basare solo sulla scelta tra iOS o Android.

1.1.4 Analisi di portabilità

Prima di approfondire lo studio della *portabilità* dei vari sistemi operativi è necessario fornire una definizione ben precisa:

> *"Proprietà di un sistema, tipicamente software, di poter essere utilizzato su diverse piattaforme. […]"* [4]

Contestualizzato allo scopo della tesi, la portabilità rappresenta la proprietà di una applicazione di poter essere sviluppata ed eseguita su più piattaforme (Android e iOS).

---

[3] *Mobile Operating System Market Share Worldwide* – statcounter.com (07/2020 – 07/2021)

[4] Definizione da: Treccani, Enciclopedia della Scienza e della Tecnica "*portabilita*" (2008)

Secondo degli studi effettuati e pubblicati sui siti *hindawi.com*[5] e *citeseerx.ist.psu.edu*[6], le tecnologie e tool di sviluppo presenti attualmente, mediante opportuni *framework*, ci permettono di poter ottenere applicazioni funzionanti sia per iOS che per Android, quindi si è in grado di sviluppare l'applicazione con un solo tool raggiungendo un target di utenti prossimo a 100%.

1.2. *Conclusioni e scelta del sistema operativo mobile*

Sebbene sia possibile, come già rilevato, creare una applicazione *cross-platform* per iOS e Android con gli opportuni tool di sviluppo, si è deciso di optare per lo sviluppo di una applicazione ***Android***. Questa scelta è dovuta, in parte, al fatto che l'utilizzo di tool complessi per la generazione di applicazioni in ambe le piattaforme comporta un effort giustificabile solo per progetti più complessi. Tuttavia, la motivazione principale risiede nel fatto che tali piattaforme tendono a semplificare e restringere eccessivamente l'uso dei sensori rispetto ad applicazione native, con possibile riduzione delle prestazioni di misura.

Pur considerando la sola fetta di Android, l'applicazione sarà disponibile per il 72% circa degli utenti al livello mondiale, o 78% in Italia (Fonti sempre dal sito *statcounter.com*). Inoltre, non è da sottovalutare la grande possibilità da parte di Android di poter sfruttare molti più componenti hardware rispetto ad iOS, questo perché Android è definito come software *open-source*[7], ovvero il codice dell'OS mobile stesso è modificabile da chiunque per permettere l'esecuzione del sistema sulla stragrande maggioranza degli hardware. D'altro canto, iOS viene definito *Closed*, ovvero il codice del sistema operativo non è accessibile a tutti e quindi non è modificabile.

Infine, l'utilizzo di Android apre la possibilità di eventuali sviluppi futuri per la creazione di dispositivi ad hoc i quali supportano questo OS e sono dotati di sensori particolarmente idonei allo scopo della tesi.

[5] Per approfondimenti: *Porting Mobile Apps from iOS to Android: A Practical Experience* - K. Lamhaddab , M. Lachgar , K. Elbaamrani (2019)

[6] Per approfondimenti: *Portability Study of Android and iOS* – B.Stewart (2012)

[7] Per approfondimenti: *Treccani. Dizionario della lingua italiana* (2017)

Capitolo 2

Android, API e sensoristica

2.1. *Android*

Come già anticipato nel capitolo 1, Android è un sistema operativo mobile ed il suo codice open-source è stato sviluppato dell'azienda *Google*.

Prima di parlare del sistema operativo e di alcuni punti chiave al fine della tesi, è necessario definire brevemente il concetto di *kernel* di un sistema operativo.

Il kernel costituisce il cuore di un qualsiasi sistema operativo, questo perché è un programma che permette la comunicazione tra applicazioni e componenti hardware. È uno dei primi programmi ad essere eseguito e si interfaccia direttamente con i componenti hardware (CPU, RAM …), inoltre lavora al livello più basso nell'architettura del sistema.

Sempre durante il capitolo 1, è stato accennato che un OS mobile fosse un particolare tipo di sistema operativo, infatti il kernel stesso di Android si basa su quello del sistema operativo *Linux*, il quale è utilizzato per lo sviluppo professionale di applicazioni, come sistema operativo per server e per molti altri scopi.

Data la continua revisione e miglioramento del codice di Android, negli anni sono state rilasciate diverse versioni[8] del sistema, con nuove funzionalità, miglioramento in termini di sicurezza, di ottimizzazione e anche dall'API level[9]; le versioni vengono inoltre identificate da un numero univoco chiamato semplicemente "*Version*".

[8] Per approfondimenti: *Codenames, Tags, and Build Numbers* - Android.com

[9] In seguito verrà approfondito lo studio degli API in Android

2.1.1 Utilizzi e accenni di sviluppo Android

Data la natura open-source del codice, tutte le eventuali customizzazioni sono implementabili da utenti esperti e ciò ha permesso lo sviluppo di molti software di simulazione delle applicazioni i quali sono presenti in rete. Alcuni esempi possono essere *bluestacks,* per la sola esecuzione di applicazioni, oppure *Android Studio*, che viene utilizzato per la simulazione in real-time di applicazioni in fase di sviluppo. Avere una vasta gamma di scelta per quanto riguarda la simulazione dell'applicazione permette di avere a disposizione un ambiente di sviluppo agevolato, l'app potrà essere controllata su diverse versioni e in diversi ambienti; sarebbe inoltre possibile un controllo diretto sull'hardware mediante installazione di prototipi dell'applicazione sugli smartphone o qualsivoglia dispositivo equipaggiato con lo stesso sistema operativo.

È importante scegliere i target di versione del sistema su cui l'app si dovrà basare, questo perché ogni versione è in possesso di specifiche funzionalità che potrebbero non essere presenti nelle altre, in particolare in quelle precedenti.
Scegliere una versione troppo datata potrebbe limitare l'utilizzo di determinate funzionalità mentre utilizzarne una troppo recente porterebbe a limitare il numero di utenti che potranno avere accesso all'applicazione.

2.2. ***API***

Con il termine API, acronimo inglese di *application programming interface*, s'intende l'interfaccia messa a disposizione da un software al fine di poter comunicare con altri software o componenti hardware.

È molto importante non confondersi con concetto di kernel:

gli API sono un'interfaccia per gli sviluppatori che utilizzeranno apposite librerie durante la scrittura del codice per usarli, il kernel è una interfaccia per le applicazioni permettendo la comunicazione con i componenti interessati e fornendo le risorse richieste, inoltre non possiede API e non necessita di librerie

2.2.1 Vantaggi API

Il grande vantaggio degli API sta nel fatto che possono permettere la comunicazione senza che i due enti conoscano l'architettura dell'altro.

Dato che sono messi a disposizione dai programmatori di un software verso enti sconosciuti, o esterni al sistema, sarebbe possibile realizzarne alcuni permettendo a terzi di ottenere dati o servizi senza che essi implementino architetture complesse.

Non possiedono limiti per quanto riguarda la personalizzazione poiché tali limiti vengono definiti dai programmatori stessi; è più efficiente utilizzarli per permettere la comunicazione tra diverse applicazioni poiché esse dovranno essere a conoscenza solo degli API e non della struttura delle altre applicazioni, se ben definiti permettono l'integrazione di funzioni o servizi su molteplici piattaforme, applicazioni e molto altro.

2.2.2 API Android – API Level

L'API Level è un numero intero che identifica il framework API presente in una specifica versione della piattaforma Android, per questo è una delle informazioni chiave che viene fornita al rilascio di nuove versioni.

Durante la fase di sviluppo, conoscere il valore dell'API Level corrisponde a sapere in quali contesti l'applicazione è in grado di funzionare, inoltre per lo sviluppo mobile e l'utilizzo delle interfacce API è necessario conoscere il concetto di *SDK*:

SDK è l'acronimo di *Software Development Kit*, ovvero un kit di sviluppo del software, nella programmazione mobile è molto importante utilizzare un kit, o per meglio dire la SDK migliore per lo sviluppo di un'app su determinate piattaforme.

Nello sviluppo mobile Android, utilizzare la giusta SDK permette un utilizzo facilitato delle interfacce API che vengono utilizzate dall'applicazione ai fini della tesi.

2.3. *Sensoristica smartphone*

Grazie al continuo sviluppo delle tecniche e tecnologie utilizzate per la creazione di componenti fisici, gli hardware utilizzati in diversi dispositivi, come ad esempio alcuni sensori[10] negli smartphone, sono stati miniaturizzati ed integrati nei sistemi embedded fornendo ai programmatori una vasta quantità di dati utilizzabili per la realizzazione di molteplici attività che prima non sarebbero mai potute essere realizzate.

Tra gli scopi più comuni dei sensori presenti in uno smartphone ci sono:

- **GPS**, localizzazione della posizione, utilizzato dalle applicazioni per l'utilizzo di mappe;
- **Sensore di prossimità**, determina la posizione dello smartphone rispetto ad una superficie, utilizzato per lo spegnimento dello schermo durante una chiamata;
- **Giroscopio**, rileva i movimenti compiuti dal dispositivo, utilizzato per il cambio di orientazione dello schermo;

[10] Definizione da: Treccani, Enciclopedia della Scienza e della Tecnica "*Sensori*" (2008)

- **Accelerometro**, misura le accelerazioni compiute dal dispositivo, utilizzato nelle applicazioni di tipo contapassi;
- **Bussola**, determina le posizioni nel rispetto dei punti cardinali, utilizzato per la misura della forza e direzione dei campi magnetici;
- **Barometro**, misura dei cambiamenti di pressione, migliora la posizione fornita dalle applicazioni che utilizzano le mappe;
- **Sensori di luminosità**, determinano l'intensità della luce, utilizzati per il cambiamento automatico della luminosità dello schermo del dispositivo.

All'interno di uno smartphone potrebbero essere presenti i sensori sopra elencati. Tali componenti forniscono molte delle funzionalità che normalmente non sarebbero implementabili in una applicazione, o come funzionalità base del sistema stesso, come ad esempio la regolazione automatica della luminosità dello schermo e rotazione automatica.

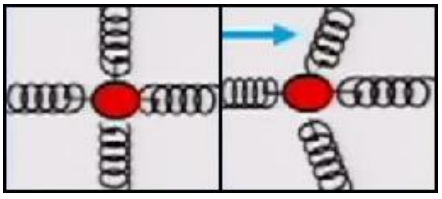
2.4. *Accelerometro*

L'accelerometro è un particolare tipo di sensore incaricato di misurare e rilevare cambiamenti di velocità dovuti a forze che alterano l'accelerazione in un corpo, nel rispetto degli assi cartesiani.

Le applicazioni di questo sensore sono notevolmente aumentate negli anni permettendo il suo sviluppo e quindi utilizzo in diversi campi come ad esempio:

- Rilevamento delle onde sismiche dovute a terremoti;
- Misura delle vibrazioni presenti in una automobile o in una costruzione edile;
- Monitoraggio delle vibrazioni di componenti industriali a rotazione (pompe, ventole, ecc…) al fine di determinare cambiamenti nel tempo valutandone lo stato di vita;
- Supporto medico e contapassi al fine di stimare il numero di passi effettuati in giornata.

2.4.1 Funzionamento dell'accelerometro



Il funzionamento dell'accelerometro è semplice, ipotizzando di avere una massa collegata ad una serie di molle, cambiamenti di velocità portano ad un movimento di questa massa generando delle forze misurabili ai capi delle molle.

Gli accelerometri digitali sono progettati in modo tale da fornire un apposito segnale digitale per il dispositivo a cui esso è collegato.

2.5. *Tecnologia MEMS*

Il tipo di tecnologia che ha permesso uno sviluppo significativo in termini di riduzione di consumi energetici, aumento della mobilità e basso costo di produzione viene definita tecnologia *MEMS* (*Micro Electro Mechanical System*).

La tecnologia *MEMS* presenta componenti che si aggirano sull'ordine di grandezza del micrometro; grazie alle dimensioni ridotte, è possibile installare numerosi componenti su un unico dispositivo mobile quale lo smartphone. Altri vantaggi noti della tecnologia *MEMS* sono[11]:

- Minimizzazione dei requisiti energetici e dei materiali;
- Aumento della sensibilità, accuratezza e affidabilità;
- Produzioni di massa a basso costo;
- Diminuzione della potenza richiesta per il funzionamento, in riferimento ad altri sistemi;
- Facilmente integrabili nei sistemi;
- Componenti facilmente intercambiali rispetto alle controparti macroscopiche.

L'applicazione sfrutterà le potenzialità dell'accelerometro *MEMS* presente sul dispositivo.

2.6. *Implicazioni con Android e audience*

L'accelerometro è accessibile attraverso degli opportuni API che sono messi a disposizione per i programmatori e documentati sul sito ufficiale *android.com*[12], si precisa che *accelerometro* e *giroscopio* vengono classificati come sensori di movimento (*Motion sensors*).

Nella documentazione viene accennato che le prime interfacce sui motion sensor sono state implementate dall'API Level 3; negli anni molte più funzioni sono state eliminate, altre aggiornate e ne sono state introdotte di nuove.

Leggendo accuratamente la documentazione, un livello minimo adeguato per gli API potrebbe corrispondere a quello di livello 15, associato alla versione Android 4.0.3 – 4.0.4. Qualora fosse necessario, analizzando i dati presenti sul sito *statcounter.com*[13] relativo alle percentuali di utilizzo delle diverse versioni Android, si può evincere che

[11] "*MEMS Technology: A Review*" Manish Kumar Mishra, Vikas Dubey, P. M. Mishra and Isharat Khan

[12] Per approfondimenti: *Sensor* – android.com

[13] *Mobile & Tablet Android Version Market Share Worldwide* - statcounter.com (07/2021 - 07/2021)

circa l'88% di tutti gli utilizzatori del sistema hanno una versione almeno uguale, se non superiore, alla 7.0 a cui corrisponde un API Level di 24, ovvero l'applicazione sarebbe accessibile a circa il 63.4% della popolazione mondiale nel caso venisse utilizzato tale livello, che corrisponde al 68.6% secondo i dati riguardanti la popolazione italiana.

Nella versione Android 8.0 (API Level 26) è stato introdotto un nuovo set di variabili ed eventi, riguardanti anche l'accelerometro, in particolare la classe Sensor aggiunge una costante chiamata *TYPE_ACCELEROMETER_UNCALIBRATED* con 6 nuovi parametri che, nel rispetto agli assi cartesiani, forniscono la visione dei dati sia calibrati che non calibrati (la calibrazione dei dati verrà meglio analizzata in seguito nel corso della tesi). Riducendo il numero di utilizzatori finali a circa l'80% saremmo in grado di accedere a tali dati, caso contrario si avrebbe un numero maggiore ma meno dati da analizzare.

Durante la fase di sviluppo, date le necessità dell'applicazione, sono state fatte scelte progettuali che hanno determinato l'API Level più adeguato.

2.7. *Accuratezza*

Ai fini della tesi, è molto importante approfondire la precisione dei dati forniti dal sensore, evidenziando particolari interferenze che potrebbero compromettere il risultato finale. Parte dello studio relativa alla precisione dei sensori è stato pubblicato e approfondito in letteratura[14], dove viene affrontato come i diversi fattori ambientali, tecnologie presenti su diversi smartphone, piattaforme, metodi di accesso e diversa conversione dei dati possano alterare il risultato finale. Da questo studio viene evidenziato che:

> *"L'accuratezza dei dati relativi all'orientazione spaziale è prevalentemente basata sull'investigazione di influenze esterne ed il corso naturale dei valori, per lo più implementati durante la produzione (Grewal & Andrews, 2010). Uno di queste influenze esterne è la temperatura a cui il sensore di orientazione opera. […]".*

Secondo la documentazione presente su *android.com* [15] , le letture da parte dell'accelerometro vengono calibrate per compensare i possibili errori del sensore, tra cui anche gli errori derivanti dalla temperatura. In generale, il costruttore del componente fisico si adopera per la compensazione delle letture fornite. Nel caso fosse necessario, gli API di android permettono la visione dei dati non calibrati lasciando spazio per elaborazioni e compensazioni custom made non limitandosi a quelle svolte di default.

---

[14] *Smartphone sensor accuracy varies from device to device in mobile research: The case of spatial orientation* - T. Kuhlmann, P. Garaizar & U. Reips (2020)

[15] Per approfondimenti: *Sensor types* – android.com

Vibrazioni e quindi accelerazioni improvvise da parti di fonti esterne contribuiscono all'inaccuratezza dei dati. Data la natura dell'app, le vibrazioni che verranno generate dal mezzo di trasporto al fine di determinare l'andamento della strada contribuiscono anch'essi ad un aumento del rumore complessivo registrato.

Anche se di minor impatto, il programma utilizzato, che sia una applicazione nativa, mediante utilizzo di framework o browser, contribuirà ad un intorpidimento dei dati raccolti, lo studio consiglia l'utilizzo di API in grado di poter fornire risultati simili su dispositivi aventi hardware e sistemo operativi differenti.

Infine e non da meno, il ruolo che ricoprono gli ammortizzatori nei veicoli costituiscono un vero e proprio problema poiché attenuano le vibrazioni esterne provenienti dalla strada. Dato il costante sviluppo automobilistico ed in particolare degli ammortizzatori installati sulle vetture, questo punto potrebbe risultare decisivo nel valutare il livello di utilità delle informazioni ricavate dai dati dei sensori, i quali potrebbero risultare eccessivamente compromessi per uno studio approfondito delle caratteristiche della strada.

2.8. *Incertezza accelerometri MEMS*

Le differenze principali tra gli accelerometri *MEMS* e le loro controparti macroscopiche consistono nel differente grado di incertezza a cui da cui sono affetti e dai diversi range di lavoro. Durante la realizzazione di particolari progetti, non è sempre possibile utilizzare componenti *MEMS* poiché non sono in grado di garantire un alto grado di precisione.

Di seguito vengono riportati i *datasheet* delle caratteristiche di alcuni modelli di accelerometri *MEMS* triassiali presenti in commercio:

Modello \ Caratteristica	Sensibilità $[mV/g]$		Range di misurazione $[g\ pk]$	Errore di Linearità	Errore di fase	
PCB-3713F112G	675	$(\pm 3\%)$	± 2	$\leq 1\%$	$< 2.5°$	$(10\ Hz)$
PCB-3713F1110G	135	$(\pm 3\%)$	± 10	$\leq 1\%$	$< 10°$	$(100\ Hz)$
PCB-3713F1130G	45	$(\pm 5\%)$	± 30	$\leq 1\%$	$< 10°$	$(100\ Hz)$

Capitolo 3

Requisiti e funzionalità del sistema

3.1. *Cenni del linguaggio di programmazione – Java*

Il linguaggio di programmazione scelto per lo sviluppo dell'applicazione è Java, linguaggio *class-based*, *object-oriented* molto generale e versatile grazie alla particolarità di poter essere eseguito su diverse piattaforme. Il programma scritto degli sviluppatori, una volta compilato, verrà convertito in un particolare codice *bytecode*[16] comprensibile dal linguaggio Java. Tutte le piattaforme in possesso della macchina virtuale Java (*Java Virtual Machine* o *JVM*) saranno in grado di eseguire tale codice.

La JVM è una macchina virtuale, ovvero è un particolare componente software e non hardware, ciò comporta che potrà essere semplicemente installata su una macchina senza dover incorrere in particolari costi aggiuntivi per l'aggiunta di nuovi componenti fisici.

Grazie alla struttura *object-oriented* del linguaggio, è possibile raggruppare tutti i dati e procedure in un'unica entità chiamata *classe*. Gli *object* sono istanze della classe, ovvero pezzi di codice sicuro al cui interno sono presenti dati per caratterizzarli e metodi per interagirci derivanti dalla classe di riferimento. Questa feature del codice di programmazione permetterà miglioramenti a tutti gli oggetti modificando una sola volta la loro classe di riferimento, questo per eventuali manutenzioni o sviluppi futuri.

[16] Il *bytecode* è un codice sorgente compilato per poter essere interpretato da un opportuno software, come ad esempio la JVM per il codice Java. Questo codice si pone tra il linguaggio macchina e quello di programmazione.

3.2. *Cenni IDE – Android Studio*

Per lo sviluppo vero e proprio dell'applicazione è stato utilizzato un IDE (*Integrated Development Enviroment*), ovvero un ambiente di sviluppo integrato, chiamato *Android Studio*, IDE ufficiale presente nella sezione developer di *android.com*. Utilizzando Android Studio sarà possibile utilizzare il linguaggio di programmazione Java.

L'IDE in questione permette di creare progetti completamente vuoti oppure con dei *template*[17] e funzionalità già presenti sin dalla creazione del progetto, facilitando e riducendo i tempi di sviluppo delle applicazioni.

I linguaggi di programmazione che vengono consigliati dall'IDE sono Java e Kotlin; per quanto riguarda quest'ultimo, anche se risulta essere un linguaggio più utilizzato rispetto a Java nell'ambito di programmazione di applicazioni al livello professionale in Android, avendo avuto più anni di esperienze col codice Java si è deciso di optare per l'utilizzo di tale linguaggio. Nel caso in cui fosse stato scelto Kotlin come linguaggio di programmazione dell'applicazione, si sarebbero riscontrate migliori prestazioni, in termini di tempi di compilazione del codice e dimensione di tali file da interpretare, dato che tale linguaggio risulta essere più performante rispetto a Java.
In entrambi i casi i linguaggi sono compilabili ed eseguibili su una JVM e quindi i codici sarebbero eseguibili su tutte le piattaforme in cui tale macchina virtuale è installata.

3.2.1 Ambienti di sviluppo, scelte progettuali e AVD

Grazie ad Android studio, come già introdotto nel capitolo 2, è possibile verificare la normale funzionalità dell'app in diverse versioni del sistema operativo Android. L'IDE mette a disposizione la possibilità di creare degli *AVD* (*Android Virtual Device*), ovvero dei dispositivi virtuali con OS Android i quali potranno essere caratterizzati da una particolare versione di Android, con differenti dimensioni dello schermo virtuale in pollici e con una risoluzione in pixel; non bisogna sottovalutare che l'interfaccia grafica rappresenta un elemento quasi essenziale di ogni applicazione al livello professionale.

Per lo sviluppo dell'applicazione si è scelto di partire da una versione di Android minima pari alla 8.0 (*Oreo*) che corrisponde ad un API Level di 26, questa scelta limita l'applicabilità massima a circa l'80% e rappresenta un giusto compromesso tra il numero massimo di utilizzatori finali e di nuove funzionalità disponibili. Nel capitolo 2 è stato accennato che dei nuovi metodi sono stati aggiunti da questa versione in poi per la lettura

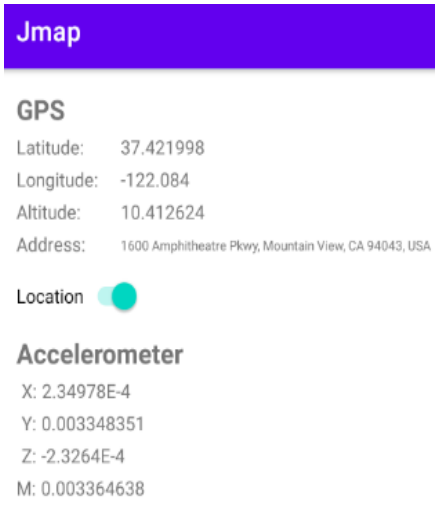
[17] Con *template* intendiamo il modello dell'interfaccia grafica e funzionale che verrà presentata all'utente finale.

di nuovi dati provenienti dall'accelerometro. Dato che non è certo che i dati forniti dai sensori non siano stati alterati al punto da essere inservibili per determinare l'andamento di una generica strada, un'altra delle motivazioni per quanto riguarda la scelta di questa versione minima è quella di privilegiare la precisione, in termini di funzionalità aggiuntive e nuovi dati presenti nelle versioni più recenti dell'OS. L'applicazione è stata creata in modo tale da privilegiare la precisione a discapito del numero di utenti finali che saranno in grado di poter installare l'app; se nonostante questo il sistema risulterà essere inefficiente allo scopo prefissato, non avrebbe senso optare per una riduzione della versione minima richiesta.

Sarà possibile installare l'applicazione su smartphone fisici, i quali dovranno essere in possesso di una versione dell'OS pari o superiore alla 8.0 e dal quale sarà possibile verificare i file *log* del sistema per monitorare il funzionamento del dispositivo.

Infine, dato che si è deciso di voler privilegiare la precisione dei dati forniti dai sensori, non verranno utilizzati metodi di risparmio energetico messi a disposizione dalla SDK di Android studio. A discapito delle risorse utilizzate e del consumo energetico rimane imperativo determinare l'utilità delle informazioni ricavabili dai dati. Nel caso in cui le informazioni dovessero risultare sufficientemente utili, un ipotetico sviluppo futuro sarebbe quello di limitare la precisione, diminuendo la versione minima richiesta di Android, aumentando l'applicabilità e limitando il consumo dell'app con i metodi di ottimizzazione delle risorse e consumo.

3.3. *Funzionalità dell'applicazione - Jmap*



L'applicazione presenta una *UI*[18] molto semplice per l'utente; le uniche operazioni consentite saranno la conferma, sottoforma di permessi utente, all'utilizzo dei dati forniti dal GPS e l'attivazione di tale servizio. Nel caso in cui non fossero stati consentiti i permessi di geolocalizzazione, l'applicazione non garantirà il suo completo e corretto funzionamento.

All'attivazione del servizio, l'utente potrà osservare la sua posizione, nel rispetto delle coordinate geografiche con l'aggiunta della variabile che indicherà l'indirizzo, in cui il dispositivo è presumibilmente situato. In background l'applicazione procederà con il campionamento dei dati dell'accelerometro ad istanti T_r,

[18] UI: *User Interface* – Interfaccia utente, sia grafica che funzionale.

inizialmente posto pari a $100\ ms$, tempo adatto a scongiurare un sovraccarico di dati da inviare successivamente al server remoto. Per evitare l'eccessiva perdita di informazioni nell'intervallo T_r tra i campioni, è stato realizzato un'elaborazione dei dati utilizzando il metodo della *non-overlapping window*:
ogni 10 millisecondi (T_c) viene catturato un campione fino ad arrivare ad una lista composta da 10 elementi da cui verranno ricavate alcune informazioni, come il valor medio e la deviazione standard. Questi nuovi valori verranno incorporati in nuovi campioni, registrati ad intervalli T_r, i quali conterranno informazioni della loro finestra di riferimento.

Dato che non è possibile inondare il server di richieste generate a ogni istante di tempo T_r e che il dispositivo potrebbe non avere un costante accesso ad internet, è stato necessario definire un meccanismo che consentisse di non perdere alcun tipo di dato durante l'esecuzione dell'applicazione e permettesse l'invio dei dati sul server in un formato compatto, comprensibile e che racchiuda più campioni possibili al suo interno.

3.3.1 Sincronizzazione e cleanup dei dati

Durante l'esecuzione dell'applicazione, tutti i dati vengono salvati su un database SQL locale chiamato *Room;* una possibile alternativa sarebbe potuta essere *SQlite*[19] ma per questioni di semplicità questa strada non è stata praticata. Quando il servizio è attivo, tutti i dati e campioni vengono registrati nel database locale del dispositivo, includendo anche i dati elaborati col metodo della *non-overlapping window*, dato che vengono incorporati come parametri di tali campioni.

Tutte le operazioni che vengono svolte da una qualsiasi applicazione vengono gestite in un *thread*[20] principale, detto anche main o UI thread nel caso di Android. Durante la connessione ad un server remoto sono presenti incognite che non possono essere gestite dal dispositivo, tra cui quelle di percorso e dello stato server. Utilizzare il main thread per svolgere le operazioni d'instaurazione della connessione potrebbe portare ad una serie di errori tra cui la saturazione delle risorse con conseguente blocco o crash dell'applicazione.

È stato necessario implementare un apposito thread che si occupasse dell'invio dei dati al server in maniera asincrona. Ogni intervallo di tempo T_m, posto pari a $180\ s$, un set di finestre, non superiori ad un limite di 5000, vengono convertite in una stringa in formato

[19] *Room* risulta essere più semplice in termini implementazione con l'aggiunta di varie fasi di verifica, esso è una ulteriore astrazione di *SQlite*, il quale non possiede fasi di validazione durante la compilazione. In conclusione, *SQlite* risulta essere più performante e generico, *Room* invece è più sicuro ma più lento.

[20] Il *thread* rappresenta una suddivisione di un processo adibito per l'esecuzione di un set di operazioni.

JSON[21] ed inoltrati al server remoto dotato di appositi API scritti ad hoc e messi a disposizione dell'applicazione.

Il server svolgerà tutte le operazioni necessarie al fine di garantire l'integrità dei dati e continuità del servizio, tra cui conversione del pacchetto in ingresso formattato in notazione *JSON* in formati di dati comprensibili dal server, controllo dei dati ricevuti e aggiunta di tali dati in un database. Il server infine fornirà un pacchetto di risposta, sempre in formato *JSON*, il quale fornirà le informazioni necessarie sui campioni che dovranno essere eliminati dal database locale del dispositivo (fase di cleanup).

3.4. *Funzionalità essenziali del server*

Come già accennato nei paragrafi precedenti, il server dovrà essere in grado di acquisire i dati provenienti dall'applicazione mediante l'invio di formati comprensibili da entrambe le strutture. Le sue funzionalità possono essere divise in tre principali categorie:

- **Ricezione e conversione dei dati**, la porta del server rimarrà sempre in ascolto per richieste di tipo HTTPS (porta 443).
  Sono stati scritti degli API ad hoc nel linguaggio *PHP*[22] i quali, dato un input in notazione *JSON*, si accerteranno che siano stati inseriti tutti i dati necessari al fine di ottenere una loro corretta conversione[23] da inserire nel database;
- **Connessione al database**, in via di sviluppi futuri non si è esclusa la possibilità di separare le macchine fisiche su cui avverranno le operazioni di ricezione dei dati e quelle di storage in un database. Cambiando i file di configurazione della connessione[24], sarebbe possibile utilizzare un database remoto ma che per questioni di semplicità si è deciso di utilizzare una sola macchina per entrambe le operazioni;
- **Generazione dei file contenenti campioni**, è stato scritto un API ad hoc che, fornito in input uno specifico punto geografico espresso in funzione di latitudine e longitudine, genererà un file di tutti i campioni raccolti dai diversi dispositivi mobili in vicinanza del punto d'interesse nel raggio di r metri.

[21] *JSON - JavaScript Object Notation*: formato semplice per lo scambio di dati alfanumerici. I dati vengono convertiti in tale notazione, inoltrati e riconvertiti in formati adatti per la struttura ricevente.

[22] *PHP – Hypertext Preprocessor*: linguaggio utilizzato lato server per la generazione di pagine *HTML* dinamiche, ovvero che cambiano a seconda del contesto in cui si trovano.

[23] Per conversione dei dati intendiamo la trasformazione dell'input da notazione JSON a dei tipi di dati comprensibili per il *PHP*.

[24] Di default il file provvederà ad una connessione al *localhost*, ovvero alla stessa macchina fisica

È ben chiaro che lo scopo principale del server è quello di essere un intermediario per tutti i dispositivi in grado di inviare un *JSON* contenente tutte le informazioni necessarie allo studio, questo permetterà lo sviluppo del sistema per diverse piattaforme mobile.

3.4.1 Specifiche sul database remoto

Al fine di permettere una riduzione significativa delle informazioni da salvare nel database, si è deciso di inglobare i valori di *uuid*[25] e *model*[26] in un'unica tabella chiamata *flow*, nel quale ogni riga verrà identificata da un valore intero (*id*). Con questo cambio di chiavi primarie evitiamo che le stringhe dei parametri interessati vengano ripetute per ogni campione inserito riducendo lo spazio necessario per registrare le stesse informazioni. In particolare, sappiamo che *uuid* è una stringa di 36 caratteri, dove ogni carattere utilizza 1 byte di dimensione, mentre *model* è una stringa di dimensione che si aggira intorno a 20 caratteri. Associando la coppia di *uuid* e *model* ad un valore intero (*id*) di 4 byte di dimensione siamo in grado di ridurre di circa 52 byte la dimensione per ogni finestra avente gli stessi valori delle due variabili in questione.

Evidentemente, come già annotato in figura 1, la combinazione degli attributi *uuid* e *model* deve risultare unica tra tutte le righe della tabella altrimenti, con l'attuale configurazione, si andrebbe incontro all'inserimento di dati ridondanti, ovvero si verrebbero a creare più righe aventi *id* differenti con gli stessi valori di *uuid* e *model*.

Si è deciso di aggiungere il campo flow.time fornendo informazioni relative a quando un determinato flusso di campioni è stato aggiunto per la prima volta nel database. Nel caso di sample.time indicherà l'istante di tempo in cui il campione è stato letto.

Negli gli attributi *x*, *y* e *z* sono indicate le medie delle letture da parte dell'accelerometro sui vari dispositivi, con le loro relative deviazioni standard *devx*, *devy* e *devz*. Questi dati sono stati calcolati utilizzando il metodo della *non-overlapping window* insieme al parametro *strength* che indicherà la forza di vibrazione complessiva di una data finestra.

Infine, con gli attributi *alt*, *lon* e *alt* indicheremo le coordinate geografiche, rispettivamente altitudine, longitudine e altitudine[27] relativi alla posizione dell'ultimo campione letto da una data finestra.

---

[25] Con il valore di *uuid*, variabile di 32 caratteri scelti casualmente, vengono classificati campioni letti ad intervalli di tempo consegutivi.

[26] Nella variabile *model* viene registrato il modello del dispositivo.

[27] L'aggiunta della variabile relativa all'altitudine è prettamente legato ad eventuali sviluppi futuri del sistema. Al fine degli studi della tesi si ipotizzerà una altitudine costante.

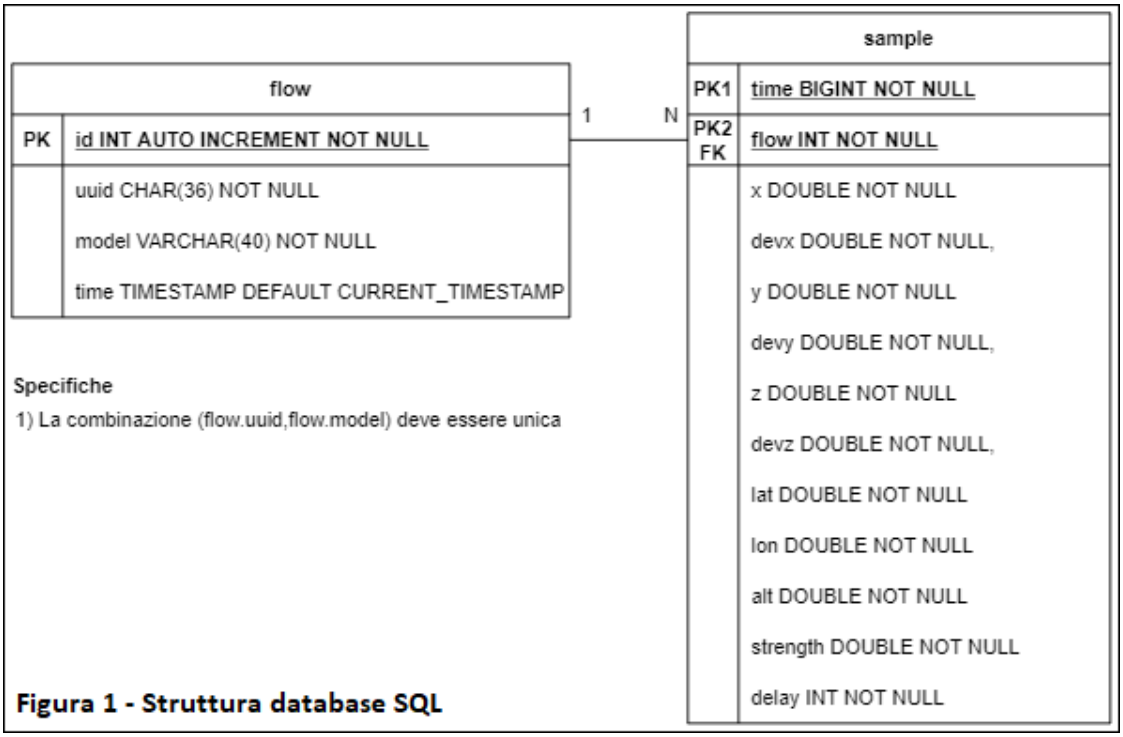


Figura 1 - Struttura database SQL

3.5. *Programma per la rappresentazione grafica dei dati*

Il linguaggio di programmazione incaricato per la scrittura del programma che si adopererà per la rappresentazione grafica dei dati immagazzinati sul database è *Matlab*, linguaggio progettato per semplificare complessi calcoli, tra cui ad esempio calcoli matriciali e risoluzione di sistemi lineari, e molto spesso utilizzato anche per lo studio e analisi di dati.

Per evitare sovraccarichi di elaborazione da parte di specifici componenti del sistema, cercando anche di renderlo più semplice e flessibile, molte operazioni vengono direttamente eseguite sui dispositivi mobili stessi. Definendo n come il rapporto dei tempi tra T_r e T_c, possiamo affermare che il metodo della *non-overlapping window* riduce di circa n volte il numero di campioni che saranno immagazzinati nel database e successivamente rappresentati dal programma. Come effetto collaterale della compressione di circa n campioni si va incontro ad una perdita di informazioni di cui il programma dovrà tener conto, infatti sono state introdotte variabili per cercare di attenuare il problema:

- ***strength***, una variabile in grado di quantificare un valore di forza relativa alla vibrazione massima avvenuta in una particolare finestra.
  Il calcolo di tale variabile avviene secondo la seguente formula[28]:

[28] Formula presente nell'articolo: *Accident Analysis and Prevention* – (2021) Q. Ma, H. Yang, A. Mayhue, Y. Sun, Z. Huang, Y. Ma

$$strength = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2}$$

I valori di *max* e *min* identificano rispettivamente i picchi di massimo e minimo in una finestra nelle diverse coordinate;

- ***delay***, sottrazione dei tempi di lettura dell'ultimo campione di una finestra per il tempo del primo campione. Tale variabile verrà utilizzata per uno studio diretto sul tempo di generazione di ogni finestra al fine di determinare se vengono create ad intervalli di tempo costanti ed evidenziando particolari anomalie.

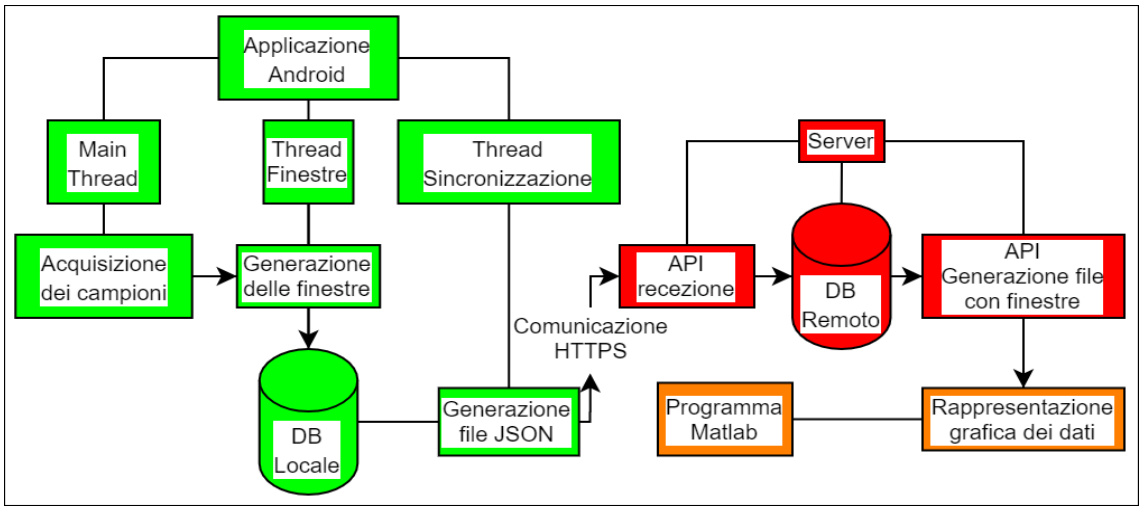
Il programma, dopo opportuni calcoli della media e della deviazione standard delle coordinate *x*, *y* e *z*, dovrà effettuare dei controlli sulla variabile *strength*, valutando la presenza di particolari eventi di rilievo.

È stato effettuato uno studio utilizzando la variabile $delay$ per valutare la precisione e costanza del tempo di campionamento dell'accelerometro. Con riferimento alla stessa variabile, verranno eliminate finestre relative a particolari valori ritenuti inaffidabili.

Per mantenere una costanza col metodo di comunicazione avvenuta tra i diversi componenti del sistema, il programma è stato dotato di una funzione in grado di leggere dei file *JSON* permettendo la lettura di formati generati da sistemi esterni, tutto purché tali file siano generati rispettando una precisa struttura.

3.6. *Riassunto del sistema*

Di seguito è riportato uno schema che riassume il comportamento generale del sistema:



Capitolo 4

Analisi del codice dell'applicazione

4.1. *Configurazione del servizio di geolocalizzazione*

Per la lettura dei dati provenienti dal GPS sono stati utilizzati i servizi destinati per la geolocalizzazione messi a disposizione dagli API di *Google Play*, in particolare il servizio utilizzato è chiamato *Fused Location Provider*[29] per Android.

Per poter utilizzare il servizio è necessario aggiungere una apposita dipendenza nella sezione del *build.gradle*[30] adibito alle dipendenze esterne.

```
1 // Dipendenze del GPS - Fused Location Provider for Android
2 implementation 'com.google.android.gms:play-services-location:18.0.0'
```

Al fine di permettere all'applicazione l'utilizzo dei dati del GPS è necessario che nel file *AndroidManifest.xml*[31] vengano inseriti i metadati necessari per poter usufruire del servizio; questi due passaggi sono fondamentali per il corretto funzionamento dell'app.

```
1 <!-- Permessi per la geolocalizzazione-->
2 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
3 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

[29] Per approfondimenti consultare la documentazione "*Simple, battery-efficient location API for Android*"

[30] Il *build.gradle* costituisce la parte dell'applicativo in cui vengono specificati tutti i moduli e le configurazioni utilizzate all'interno del progetto.

[31] *AndroidManifest.xml* è un file XML contenente tutti i metadati necessari per il corretto funzionamento dell'applicazione.

I dati del GPS, per il sistema Android, vengono considerati come dati sensibili. Per l'effettivo utilizzo di queste informazioni è necessaria una conferma, da parte dell'utente, all'accesso alla posizione del dispositivo mediante un popup di conferma.

4.1.1 Utilizzo del servizio di geolocalizzazione

Per ottimizzare il codice, è stata creata una funzione che restituisca l'informazione, sotto forma di parametro booleano, se l'applicazione possieda i permessi necessari per l'utilizzo del servizio, in caso contrario verrà generato un popup per un'ulteriore conferma.Questa funzione viene richiamata nel codice ogni qualvolta sia necessario una verifica sui permessi presenti nel sistema. In particolare, nel caso non siano presenti i permessi relativi alla localizzazione viene verificato che sia stata utilizzata una versione minima dalle *SDK* al fine di poter richiedere i permessi con un apposito popup, se così non fosse non sarebbe possibile continuare.

```
1  private boolean checkGpsPermission(){
2          // Controllo dei permessi
3
4          if(ActivityCompat.checkSelfPermission(this,
   Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED){
5              // Abbiamo i permessi
6              return true;
7          }
8          else{
9              // Non abbiamo i permessi
10             if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
11                 requestPermissions(new String[]
   {Manifest.permission.ACCESS_FINE_LOCATION}, PERMISSION_LOCATION);
12             }
13         }
14         return false;
15     }
```

È stata creata una apposita classe ed interfaccia per gestire le funzioni di geolocalizzazione, rispettivamente GPS.java e IGPS.java. All'interno della classe sono presenti gli oggetti fondamentali con il quale sarà possibile interagire con gli API.

```
1 LocationCallback locationCallBack;
2 LocationRequest locationRequest;
3 FusedLocationProviderClient fusedLocationProviderClient;
4 public long time = 100;
```

L'oggetto fondamentale che permetterà l'utilizzo dei servizi di geolocalizzazione viene istanziato dalla classe *FusedLocationProviderClient*[32], mentre le classi *LocationRequest*[33] e *LocationCallback*[34] verranno utilizzate rispettivamente per richiedere una determinata qualità del servizio (intervalli di richiesta e accuratezza) e per ricevere informazioni relative al cambio di posizione da parte dell'oggetto *fusedLocationProviderClient*. Al fine di generalizzare il codice, la variabile *time* indicherà, come intero, il tempo in millisecondi del valore dell'intervallo tra due richieste dovute ad un cambio di posizione.

Per consentire all'utente l'interazione con la classe, sono stati implementati due metodi che permettono la gestione delle richieste di cambio posizione, aggiornando i vari parametri nel programma, e rimuovendo tutte le richieste future nel caso di chiusura del servizio.

```
@SuppressLint("MissingPermission")
    public void Location_ON() {
        fusedLocationProviderClient =
LocationServices.getFusedLocationProviderClient(activity);
        fusedLocationProviderClient.requestLocationUpdates(locationRequest,
locationCallBack, null);
        fusedLocationProviderClient.getLastLocation().addOnSuccessListener(activity,
new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                if(location != null) {
                    iGps.updateUI(location);
                }
            }
        }); }
    public void Location_OFF() {
        fusedLocationProviderClient.removeLocationUpdates(locationCallBack);
    }
```

La parte relativa al controllo dei permessi non viene gestita nella classe *GPS.java* per cui è necessario ipotizzare che essi siano stati già forniti se il metodo *Location_ON* viene eseguito. L'oggetto *iGps*, istanziato dall'interfaccia *IGPS.java*, permetterà l'esecuzione di una funzione o metodo di un'altra classe, in questo caso della *MainActivity* per l'aggiornamento delle variabili contenenti i valori delle coordinate geografiche e della *UI*.

[32] Per consultare la documentazione completa: "*FusedLocationProviderClient* " developers.google.com

[33] Per consultare la documentazione completa: "*LocationRequest*" developers.google.com

[34] Per consultare la documentazione completa: "*LocationCallback*" developers.google.com

Per poter utilizzare la classe GPS.java per istanziare oggetti nella MainActivity è necessario configurare i parametri necessari, tra cui l'interfaccia, che per ottimizzazione del codice viene gestita da un'unica funzione nella *MainActivity*.

```
public void setupGPS(){
    igps = new IGPS() {
    @Override
    public void updateUI(Location location) {
        if(location!=null)
            GpsUpdateUI(location);
    }
    gps = new GPS(MainActivity.this, MainActivity.this, igps);
};}
```

Per eventuali sviluppi futuri sarebbe possibile importare la classe *GPS.java* ed utilizzare una funzione custom made grazie ad una corretta configurazione dell'interfaccia.

Grazie alla funzione *GpsUpdateUI,* avente in ingresso un oggetto che possiede informazioni di coordinate geografiche, vengono aggiornati i valori delle variabili relativi alla latitudine (*lat*), longitudine (*lon*) e altitudine (*alt*). Grazie alla classe *Geocoder.class* si è in grado di ricavare l'indirizzo relativo alle coordinate geografiche in ingresso; dato che la classe potrebbe portare ad un errore, questa parte del codice è stata inglobata in una struttura *try* e *catch*. Una volta aver identificato l'indirizzo, la *UI* utente viene aggiornata. Evidentemente bisogna accertarsi che il servizio sia attivo (*sw_location.isChecked()*) perché se così non fosse bisognerà terminare tutte le richieste di geolocalizzazione attive.

```
public void GpsUpdateUI(Location location){
    if(!sw_location.isChecked()){
        gps.Location_OFF();
        return;
    }
    lat = location.getLatitude();
    lon = location.getLongitude();
    alt = location.getAltitude();
    address = "-";
    geocoder = new Geocoder(this);
    try {
        List<Address> addresses = geocoder.getFromLocation(lat,lon,1);
    address = addresses.get(0).getAddressLine(0);
    }
    catch(Exception e){
        address = "-";
    }
    tv_lat.setText(lat+"");
    tv_lon.setText(lon+"");
    tv_alt.setText(alt+"");
    tv_address.setText(address+"");
}
```

4.2. *Configurazione ed utilizzo dell'accelerometro*

Le classi utilizzate le quali fungono da API che permettono un accesso ai dati dei sensori, tra cui l'accelerometro, sono *SensorManager*[35], *SensorEvent*[36], *SensorEventListener*[37] e *Sensor*[38].

Grazie alla classe *SensorManager* si è in grado di accedere alla lista completa di tutti i sensori presenti sul dispositivo e successivamente sarà possibile richiedere l'accesso ad un sensore che verrà istanziato come un oggetto *Sensor*. Il codice di setup dell'accelerometro è scritto in una semplice funzione presente nella *MainActivity*.

```
1 SensorManager sensorManager;
2 Sensor accelerometer;
3 private static final int ACCELEROMETER_PERIOD_MS = 10;
4 public void setupAccelerometer(){
5     sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
6     accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
7     sensorManager.registerListener(MainActivity.this, accelerometer,
  1000*ACCELEROMETER_PERIOD_MS);
8 };
```

Le classi API utilizzate permettono un tempo di campionamento configurabile fino all'ordine di $10^{-6}s$, ovvero microsecondi, con un passo di campionamento medio minimo pari a $10ms$ per i dispositivi utilizzati. Per evitare discrepanze nei tempi di campionamento tra i diversi dispositivi, i quali potrebbero avere un passo τ inferiore a $10ms$, si è deciso di fissarlo manualmente ad un valore pari a $10ms$ ($\tau = 10ms$)[39].

Con l'ausilio della classe *SensorEventListener* si è in grado di gestire gli eventi relativi a nuove letture avvenute sui sensori. Al fine di poter utilizzare questa classe è necessario che i suoi metodi *onAccuracyChanged* e *onSensorChanged* vengano implementati nella classe.di riferimento, in questo caso la *MainActivity*.

```
1 import android.hardware.Sensor;
2 import android.hardware.SensorEvent;
3 import android.hardware.SensorEventListener;
4 import android.hardware.SensorManager;
5 import ...
6 public class MainActivity extends Activity implements SensorEventListener{...}
```

[35] Per consultare la documentazione completa: " *SensorManager* " developers.android.com

[36] Per consultare la documentazione completa: " *SensorEvent* " developers.android.com

[37] Per consultare la documentazione completa: " *SensorEventListener* " developers.android.com

[38] Per consultare la documentazione completa: " *Sensor* " developers.android.com

[39] Nel capitolo 5 verrà affrontato un breve studio sull'analisi del tempo di campionamento τ

Il metodo *onAccuracyChanged*, anche se implementato, non è stato realmente utilizzato ed un possibile sviluppo futuro potrebbe essere utilizzare tale metodo per classificare le informazioni ottenute in base al grado di accuratezza registrato nell'istante della lettura.

Il metodo *onSensorChanged* viene chiamato in media ogni $10ms$ e come parametro in ingresso è presente un oggetto istanziato dalla classe *SensorEvent*, il quale conterrà informazioni relative all'accelerometro tra cui le variazioni di accelerazioni nelle diverse coordinate cartesiane.

Una parte del metodo di elaborazione della *non-overlapping window* viene eseguita in questa sezione del codice. Dopo l'attivazione del servizio vengono inviati a blocchi di liste contenenti 10^4 campioni al thread responsabile per l'effettivo calcolo delle finestre e aggiunta delle suddette sul database locale *Room*.

```
public void onSensorChanged(SensorEvent sensorEvent) {
    x = sensorEvent.values[0]; // x
    y = sensorEvent.values[1]; // y
    z = sensorEvent.values[2]; // z
    m = x*x+y*y+z*z; // Modulo al quadrato
    if(sw_location.isChecked()){
        if(i==0) {
            dots = new ArrayList<Dot>();
        }
        // Coordinate geografiche attendibili
        if(alt!=0.0 || lon!= 0.0 || alt!=0.0) {
            dots.add(new Dot(x, y, z, lat, lon, alt));
            i++;
        }
        if(i==10000) {
            sendToThread();
        }
    }
    else{
        if(i!=0){
            sendToThread();
        }
    }
    tv_x.setText(x+"");
    tv_y.setText(y+"");
    tv_z.setText(z+"");
    tv_m.setText(m+"");
}
```

Evidentemente per la massima precisione le variabili *x*, *y* e *z* sono dei valori di tipo *double*.

4.3. *Configurazione ed utilizzo del database locale Room*

Il database *Room* fornisce una serie di metodi i quali semplificano di molto la struttura del suo progenitore *SQLite*. Come già accennato, esso risulta essere una libreria che fornisce un'ulteriore livello di astrazione permettendo un robusto accesso al database e potendo utilizzare tutti gli assets di *SQLite*.

Prima di poter utilizzare il database è necessario che tutte le librerie necessarie siano state aggiunte appositamente nella giusta sezione del *build.gradle*.

```
// Dipendenze del database - Room
implementation "androidx.room:room-runtime:2.3.0"
annotationProcessor "androidx.room:room-compiler:2.3.0"
```

Data la semplice notazione offerta da Room, è possibile far riconoscere molto facilmente al codice le diverse strutture delle classi da utilizzare come tabelle nel database locale, le varie query, insert e delete da eseguire per interagirci ed associare i diversi parametri delle classi agli attributi delle tabelle.

Prima della creazione del database locale, sono state create le classi *Dot.java* e *Settings.java* le quali contengono rispettivamente la struttura della tabella dei campioni da inviare a Room ed una tabella supplementare di settings per sviluppi futuri relativa all'aggiunta di nuove feature.

Utilizzando le notazioni *@Entity* e *@ColumnInfo* sarà possibile configurare il nome della tabella sul database, la sua chiave primaria ed il nome delle colonne. Qualora fosse necessario sarà possibile utilizzare anche *@NonNull* per identificare valori che non devono risultare nulli nel database e *@PrimaryKey* nel caso di un solo attributo come chiave primaria della tabella.

Sotto viene riportata parte della struttura della classe *Dot.java*.

```
@Entity(tableName="dots", primaryKeys = {"time","uuid"})
public class Dot {
    @ColumnInfo(name = "x")
    public double x;
    @ColumnInfo(name = "y")
    public double y;
    @ColumnInfo(name = "z")
    public double z;
    ...
}
```

Per poter interagire con il database utilizzando un'entità generata dalla struttura di una classe, è necessario creare un file contenente un'interfaccia a cui interno verranno poste tutte query che potranno essere eseguite sottoforma di metodi dell'oggetto che permetterà la connessione al database locale. Nel caso di *Dot.java* è stato creato il file *dotDAO.java*[40], mentre per *Settings.java* è stato creato il file *settingsDAO.java*.

Nel file *DAO* sarà possibile dichiarare metodi utilizzabili dall'applicazione per eseguire codice SQL, quali operazioni di insert (*@Insert*), delete(*@Delete*) o query (*@Query*). In particolare, utilizzando l'apposita notazione per le operazioni di insert sarà possibile dichiarare la strategia da adoperare in caso di righe ripetute nella tabella.

Sotto viene riportata parte della struttura dell'interfaccia *dotDAO.java*.

```
@Dao
public interface dotDAO {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    void addDot(Dot dot);

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void forceAddDot(Dot dot);

    @Query("SELECT * FROM dots WHERE uuid = :uuid ORDER BY time")
    List<Dot> getQueuedots(String uuid);

    @Query("DELETE FROM dots WHERE uuid = :uuid AND time<=:time")
    void removeDots(String uuid,long time);
    ...
}
```

In questo caso, per le operazioni di delete si è escelto di utilizzare una versione equivalente con la notazione *@Query*, se si fosse deciso di utilizzare la notazione col *@Delete* sarebbe stato possibile utilizzare alcune strategie[41] per particolari eventi, come nel caso di *@Insert*.

I dati che verranno inseriti nel database locale corrispondono alle finestre ottenute elaborando i campioni registrati dall'accelerometro. Per evitare problemi di sincronizzazione, potranno essere eliminate finestre già ricevute e registrate sul server remoto permettendo una pulizia dei dati sul database locale.

[40] L'acronimo *DAO* sta per *Data Access Objects*, è il metodo con cui *Room* permette all'applicazione di accedere ai dati presenti sul suo database. La consulatione dei dati avviene mediante l'esecuzione di query dirette, dette anche *raw* (crude), oppure costruite mediante terzi metodi (*query builder*).

[41] Per maggiori informazioni consultare la documentazione: "*Delete*" developer.android.com

Per sviluppi futuri sarebbe possibile eliminare dati troppo datati per avere un set d'informazioni sufficientemente fresche per gli studi che ne susseguiranno.

Al fine di poter comunicare correttamente col database è necessario specificare tutte le tabelle che dovranno essere generate dalle strutture delle classi precedentemente create.

Con l'apposita notazione *@Database* è possibile specificare quali classi utilizzare per la generazione del database; per implementare le interfacce sarà necessario specificare i nomi dei metodi che faranno riferimento al file *DAO* interessato aventi il codice SQL che sarà eseguito durante l'esecuzione dell'applicazione.

Al fine di poter generare il database, è necessario che l'attuale classe erediti tutti i metodi e strutture necessarie dalla classe *RoomDatabase*[42].

```
1  @Database(entities = {Dot.class,Settings.class},version = 1,
   exportSchema = false)
2  public abstract class databasedots extends RoomDatabase {
3      private static final String DB_NAME = "Jmap";
4      private static volatile databasedots INSTANCE;
5      public abstract dotDAO dotDao();
6      public abstract settingsDAO SettingsDao();
7      static databasedots getDatabase(final Context context){
8          if(INSTANCE == null){
9              synchronized (databasedots.class){
10                 if(INSTANCE == null){
11                     INSTANCE =
   Room.databaseBuilder(context.getApplicationContext(),
   databasedots.class,DB_NAME)
12                                 .build();
13                 }
14             }
15         }
16         return INSTANCE;
17     }
18 }
19
```

Evidentemente è necessario che prima della restituzione dell'istanza dell'oggetto venga effettuato un controllo sull'esistenza del database. Nel caso in cui il metodo venga richiamato per la prima volta si occuperà della generazione del database, mentre se è già presente un database preesistente restituirà semplicemente una sua istanza.

[42] Per consultare la documentazione completa: " *RoomDatabase* " developers.android.com

4.4. *Generazione del file JSON*

Per la generazione del file *JSON* è stata utilizzata una libreria esterna chiamata *GSON*, grazie ad essa si è in grado di generare una stringa in formato *JSON* da un qualsiasi oggetto o lista di oggetti.

Per utilizzare la libreria è necessario aggiungere una dipendenza al file *build.gradle*.

```
1 // Dipendenza GSON per la conversione Object -> Json
2 implementation 'com.google.code.gson:gson:2.8.8'
```

Per la creazione del file *JSON* viene convertita una lista di tipo *List<dot>* in un array-stringa in formato *JSON* che successivamente verrà utilizzata per generare un oggetto di tipo *JSONArray* (Da non confondersi con la classe *JsonArray*).

```
1 String uuid = database.dotDao().getLatestUuidDot();
2 List<Dot> dots = database.dotDao().getQueuedots(uuid, 5000);
3 // Converto la lista in un array con sintassi JSON
4 Gson gson = new Gson();
5 String dotsJson = gson.toJson(dots);
```

Per come è strutturato il sistema e la classe *Dot.java* nello specifico, le finestre registrate ed ottenute da una chiamata al database locale possiederanno un valore dell'attributo "*uuid*" in comune per tutta la lista. Evidentemente, essendo un attributo ridondante e grazie alla notazione *JSON*, è possibile eliminare tale occorrenza dall'intero file permettendo un solo invio del parametro "*uuid*" riducendo quindi la dimensione del file.

```
1 // Rimuovo le informazioni ridondanti dall'array JSON
2 dotsJson = dotsJson.replace("\"uuid\":\"" + uuid + "\",", "");
3 dotsJson = dotsJson.replace("\"uuid\":\"" + uuid + "\"", "");
```

Per la generazione del file *JSON* vero e proprio è possibile utilizzare la classe *JSONObject* poiché permette la generazione di stringa risultate in formato *JSON*, ovvero quella che verrà effettivamente inviata al server remoto mediante un'apposita richiesta *HTTPS*.

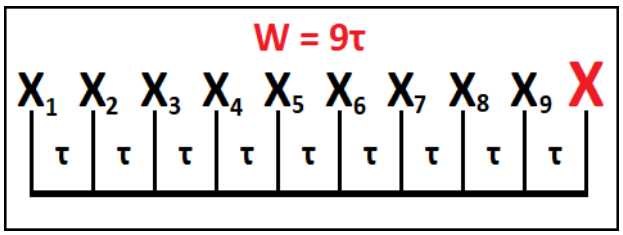
```
1 String model = Build.MODEL;
2 // Genero l'oggetto JSONObject
3 data = new JSONObject();
4 data.put("uuid", uuid);
5 data.put("model", model);
6 data.put("dots", new JSONArray(dotsJson));
7 // Converto l'oggetto in una Stringa
8 String payload = data.toString();
```

Capitolo 5

Studio e analisi dei dati

5.1. *Analisi del tempo di campionamento dell'accelerometro*

Uno dei primi problemi che si sono riscontrati fin dalle prime versioni dell'applicazione riguarda il tempo irregolare del passo di campionamento τ dell'accelerometro; in particolare esso risulta essere, in maniera casuale, contenuto in un intervallo di valori.



È stato effettuato uno studio sul tempo di generazione di ogni finestra W, salvata sul database con nome *sample.delay*. La variabile casuale W è strettamente correlata con la variabile τ poiché in ogni finestra avvengono $n-1$ campionamenti per gli n campioni a finestra[43]. Avendo impostato $T_r = 100ms$ e $T_c = 10ms$ otteniamo un valore di $n = 10$. Dato il valore di T_c, ci aspettiamo che il valor medio[44] $\bar{\tau}$ sia prossimo a circa $10ms$ ($\bar{\tau} \cong T_c$), ottenendo quindi un valor medio $\bar{W}$ che ci si aspetta essere prossimo a $90ms$ ($\bar{W} \cong 9\bar{\tau} \cong 9T_c$).

Al fine di ottenere dei risultati affidabili, l'esperimento è stato eseguito per un numero minimo di campioni pari a $3.5 \cdot 10^4$, corrispondenti a $3.5 \cdot 10^3$ finestre

[43] Nel capitolo precedente, n è stato definito pari al rapporto tra T_r e T_c ($n = T_r/T_c$)

[44] Con $\bar{\tau}$ e $\bar{W}$ indichiamo il valor medio delle variabili casuali τ e W

Inoltre, l'esperimento verrà eseguito su smartphone differenti valutando i valori al variare dell'hardware utilizzato.

5.1.1 Risultati e analisi dell'esperimento

L'esperimento è stato eseguito su tre differenti smartphone; di seguito sono riportati i risultati elaborati con un programma scritto ad hoc nel linguaggio Matlab. I dati di finestre ritenute inaffidabili sono state escluse dallo studio, di seguito vengono risportati i risultati:

Parametri\ Modello	AC2003	SM-A202F	SM-A505FN
Finestre Totali Registrate	3985	4529	5991
Finestre Eliminate	7 (0.18%)	316 (7%)	1 (0.017%)
Valor medio μ [ms]	91.470	90.352	71.696
Deviazione Standard σ [ms]	5.367	32.116	9.023
Dispersione [ms]	151	190	170

Con *Dispersione* (D) intendiamo la differenza del tempo massimo W_{Max} registrato con quella del tempo minimo W_{min}, ovvero $D = W_{Max} - W_{min}$

Durante l'avvio del servizio sono sorti dei problemi per quanto riguarda alcuni dei valori registrati, in particolare:

1- Durante l'avvio del servizio un set ristretto di finestre risultavano avere un valore di tempo W_i inferiore a 10 ms.
2- In fase di esecuzione dell'applicazione sono stati riscontrati dei picchi superiori a 100 ms rispetto al valor medio calcolato per i singoli dispositivi.

Per questo studio preliminare sono state eliminate le finestre tali che i valori dei singoli tempi W_i rispettassero la condizione $|W_i - 90| \leq 100\ ms$, ovvero è stato introdotto un valore di soglia $S = 100$ ms. Nella tabella il parametro *Finestre Eliminate* indica il numero di finestre che sono state escluse dallo studio con accanto la loro percentuale rispetto alle finestre totali registrate. Nel caso peggiore (SM-A202F) è stata registrata una diminuzione del 7% delle finestre totali mentre negli altri due casi è risultato essere un valore molto più contenuto.

Nel caso del dispositivo SM-A505FN è stato registrato un valor medio prossimo al valore di circa 71.7 ms che si discosta notevolmente dai 90 ms che ci si aspettava, questo risultato può essere frutto di un errore da parte degli API che non sono stati in grado di rispettare la condizione posta di un tempo $T_c = 10$ ms.

Possiamo affermare che i dati risultano essere sufficientemente affidabili per lo studio. Gli intervalli di tempo W_i risultano essere sufficientemente piccoli da poter permettere uno studio sulle diverse componenti registrate nel database remoto. Ai fini della tesi, questo studio non verrà approfondito ulteriormente poiché l'obbiettivo era determinare che le finestre venissero generate in un tempo sufficientemente breve. In via di sviluppi futuri, uno studio in ambito metrologico sarebbe richiesto per determinare ulteriori errori nelle letture.

Di seguito in figura 3 sono stati riportati graficamente i due dispositivi che hanno presentato le prestazioni migliori (AC2003) e peggiori (SM-A202F)

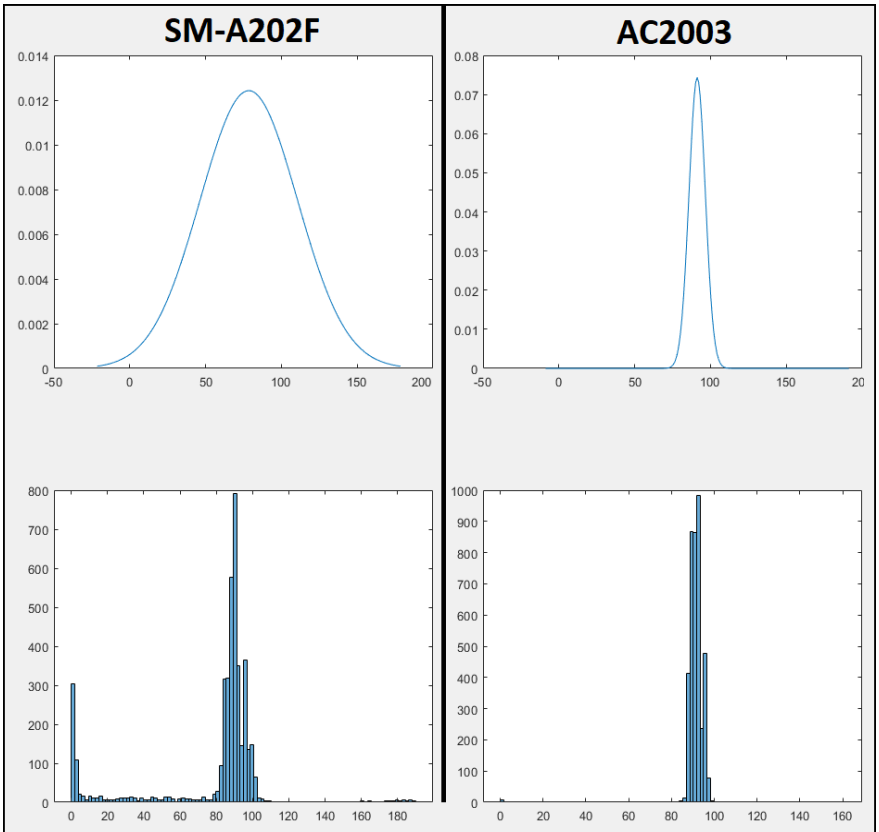


Figura 3 - Rappresentazione grafica dei dati

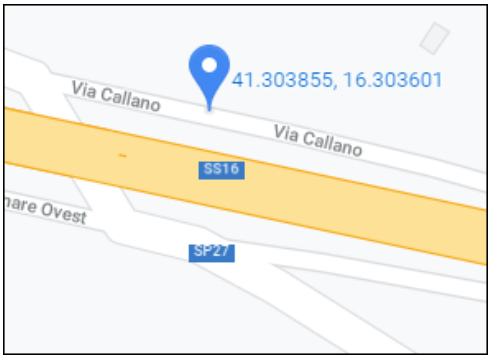
Per eventuali sviluppi futuri sarebbe possibile identificare i diversi tempi di *warm-up* per i vari modelli e dispositivi al fine di determinare un valore di soglia che risulti sufficientemente adeguato per escludere i valori non affidabili dallo studio. Sarebbe inoltre possibile instautrare un sistema di *ranking* [45] basato sulle migliori prestazioni dei diversi parametri delle finestre, tra cui il loro tempo di generazione al fine di privilegiare nello studio i dati più affidabili.

[45] Per *ranking* si intende una classifica che si aggiorna all'aumentare dei dati inseriti nel database

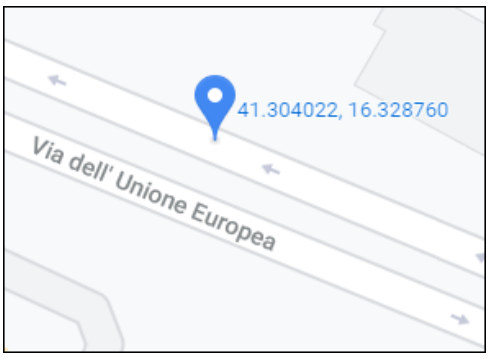
5.2. *Esperimento ed analisi dei dati su strade campioni*

Per lo studio sperimentale sono state scelte 3 differenti strade con diverse conformazioni del manto stradale. Sono stati utilizzati 3 diversi dispositivi su una vettura che viaggiava a circa 60 km/h. L'applicazione è stata utilizzata per registrare le vibrazioni causate dalle irregolarità presenti sulla superficie del manto stradale. Infine, l'app ha effettuato le operazioni di compressione delle informazioni mediante l'utilizzo del metodo della *non-overlapping window* e invio in maniera asincrona dei dati sul server remoto.

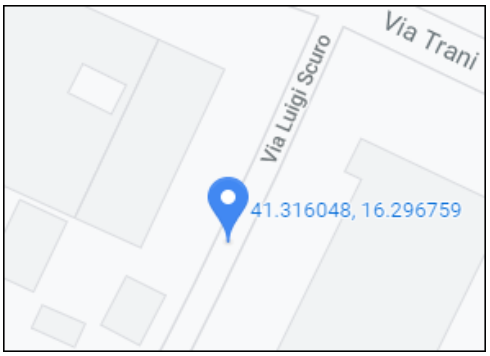
Strada A



Strada B



Strada C



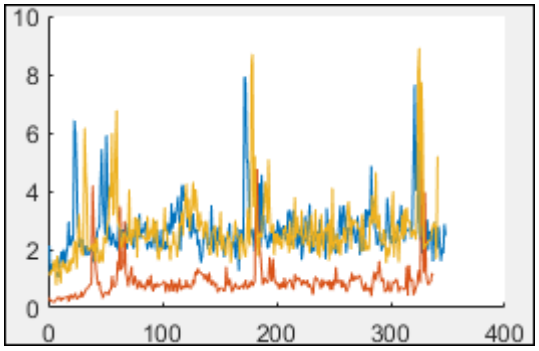
Per la rappresentazione grafica dei dati si è scelto di mostrare il valore della variabile $strength$ (S) in funzione del tempo rappresentata dalla variabile $time$. Si ricorda che la forza di vibrazione massima registrata nella finestra W è ottenuta dalla espressione:

$$S = \sqrt{(x_{Max} - x_{min})^2 + (y_{Max} - y_{min})^2 + (z_{Max} - z_{min})^2}$$

Nell'ipotesi di dati sufficientemente puliti ci si aspetta che le forze di vibrazioni minime vengano registrate nella *strada A* e le massime sulla *strada C*. La *strada B* rappresenta una via di mezzo tra le strade precedentemente elencate.

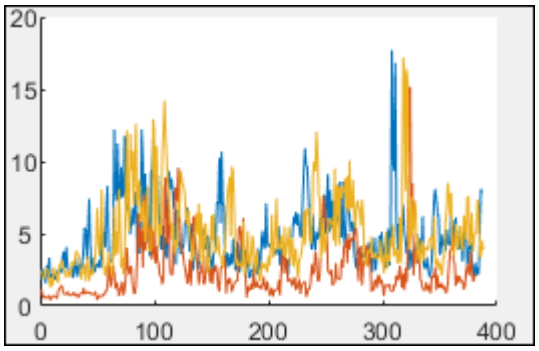
Strada A

Come prevedibile, la strada presenta i valori di vibrazioni inferiori rispetto alle altre. I picchi registrati sono presumibilmente ottenuti da piccole variazioni del manto stradale. Le registrazioni e la mappatura del manto stradale sono state effettuate nello stesso veicolo ed i grafici presentano picchi simili e vicini tra loro per i tre cellulari utilizzati.



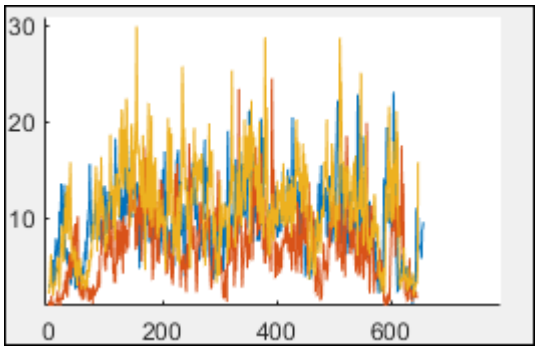
Strada B

Come prevedibile, la strada in questione non è particolarmente mal ridotta ma presenta ugualmente delle conformazioni particolari. È possibile osservare che i dati hanno un andamento simile a quelli relativi alla ***strada A*** e che a tratti si ottengono vibrazioni molto simili a quelle registrate nella ***strada C***.



Strada C

Come prevedibile, è possibile osservare che i picchi di vibrazioni massimi sono state registrati in questa strada, la più mal ridotta. Le vibrazioni sono molto violente, data la particolare conformazione del manto stradale. Con riferimento alla ***strada A***, la differenza dei dati ottenuti di S è notevole.



5.3. *Possibile analisi futura dei parametri delle finestre*

I valori $\bar{x}$, $\bar{y}$ e $\bar{z}$ indicano i valor medi delle coordinate di una generica finestra W salvata nel database, con x_i, y_i e z_i i generici parametri i-esimi $(i = 1, \dots, n)$ utilizzati per il metodo della *non-overlapping window* e ε una tolleranza fissata.

Ipotizzando che le accelerazioni siano relativamente contenute tali che:

$$|x_i - \bar{x}| \leq \varepsilon \quad |y_i - \bar{y}| \leq \varepsilon \quad |z_i - \bar{z}| \leq \varepsilon$$

1- Dalla formula della $strength\ (S)$

$$S = \sqrt{(x_{Max} - x_{min})^2 + (y_{Max} - y_{min})^2 + (z_{Max} - z_{min})^2}$$

Possiamo affermare che:

$$\begin{array}{ll} x_{Max} \leq \bar{x} + \varepsilon & x_{min} \geq \bar{x} - \varepsilon \\ y_{Max} \leq \bar{y} + \varepsilon & y_{min} \geq \bar{y} - \varepsilon \\ z_{Max} \leq \bar{z} + \varepsilon & z_{min} \geq \bar{z} - \varepsilon \end{array}$$

Da cui segue che:

$$S \leq \sqrt{\left(\bar{x} + \varepsilon - (\bar{x} - \varepsilon)\right)^2 + \left(\bar{y} + \varepsilon - (\bar{y} - \varepsilon)\right)^2 + \left(\bar{z} + \varepsilon - (\bar{z} - \varepsilon)\right)^2} =$$
$$= \sqrt{(2\varepsilon)^2 + (2\varepsilon)^2 + (2\varepsilon)^2} = 2\sqrt{3}\varepsilon$$

In conclusione otteniamo che $S \leq 2\sqrt{3}\varepsilon\ (1)$.

2- Riportata la formula della deviazione standard:

$$\sigma_x = \sqrt{\frac{\sum_{1=1}^{n}(x_i - \bar{x})^2}{n}}$$

Nelle stesse ipotesi per la formula (1) segue che:

$$\sigma_x = \sqrt{\frac{\sum_{1=1}^{n}(x_i - \bar{x})^2}{n}} \leq \sqrt{\frac{\sum_{1=1}^{n}\left((\bar{x} \pm \varepsilon) - \bar{x}\right)^2}{n}} = \sqrt{\frac{\sum_{1=1}^{n}(\pm\varepsilon)^2}{n}} = \sqrt{\frac{\varepsilon^2 n}{n}} = \varepsilon$$

Si può osservare che:

(a) il segno di ε non ha importanza per via dell'operazione di elevamento al quadrato.

(b) Il calcolo è analogo per ognuna delle diverse coordinate y e z.

In conclusione otteniamo che:

$$\sigma_x \leq \varepsilon \quad \sigma_y \leq \varepsilon \quad \sigma_z \leq \varepsilon\ (2)$$

3. Volendo calcolare la deviazione standard del modulo del vettore $\vec{m}_i = (x_i, y_i, z_i)$:

$$\sigma_m = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2}$$

Nell'ipotesi di accelerazioni contenute si può dedurre che:

$$\sigma_m = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2} \leq \sqrt{3}\varepsilon \qquad (3)$$

4. Dalla formula della deviazione standard del modulo del vettore $\vec{m}_i$:

$$\sigma_m = \sqrt{\frac{\sum_{i=1}^{n}(m_i - \bar{m})^2}{n}}$$

Per definizione, dal modulo ad istanti di tempo i nella finestra W segue che:

$$m_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

Di seguito, dato che le accelerazioni delle singole componenti sono contenute, segue che:

$$\sqrt{(\bar{x} - \varepsilon)^2 + (\bar{y} - \varepsilon)^2 + (\bar{z} - \varepsilon)^2} \leq m_i \leq \sqrt{(\bar{x} + \varepsilon)^2 + (\bar{y} + \varepsilon)^2 + (\bar{z} + \varepsilon)^2}$$

Per semplicità definiamo i valori di m_{Max} e m_{min} pari a:

$$m_{min} = \sqrt{(\bar{x} - \varepsilon)^2 + (\bar{y} - \varepsilon)^2 + (\bar{z} - \varepsilon)^2} \quad m_{Max} = \sqrt{(\bar{x} + \varepsilon)^2 + (\bar{y} + \varepsilon)^2 + (\bar{z} + \varepsilon)^2}$$

Procedendo con lo sviluppo del valor medio della variabile m_i:

$$\underbrace{\frac{\sum_{i=1}^{n} m_{min}}{n}}_{= m_{min}} \leq \underbrace{\frac{\sum_{i=1}^{n} m_i}{n}}_{= \bar{m}} \leq \underbrace{\frac{\sum_{i=1}^{n} m_{Max}}{n}}_{= m_{Max}}$$

Da cui si conclude che $m_{min} \leq \bar{m} \leq m_{Max}$ (4).

5. Ritornando alla formula della deviazione standard del modulo del vettore $\vec{m}$:

$$\sigma_m = \sqrt{\frac{\sum_{i=1}^{n}(m_i - \bar{m})^2}{n}} \leq \sqrt{\frac{\sum_{i=1}^{n}(m_{Max} - m_{min})^2}{n}} = m_{Max} - m_{min} \qquad (5)$$

5.3.1 Deduzioni dalle formule ottenute

Evidentemente, le formule (1) e (2) sono quelle utilizzabili per le analisi sui dati con la versione attuale del sistema poiché non necessitano che variabili aggiuntive vengano inserite nei parametri della finestra W da inviare successivamente al server remoto.

Le formule (3) e (5) forniscono due maggiorazioni della variabile σ_m, in termini di ε in un caso e di m_{Max} e m_{min} nell'altro. Nel caso in cui si decida di voler utilizzare la formula (5) sarebbero richiesti dei calcoli aggiuntivi per il calcolo dei valori m_{Max} e m_{min}. Sarebbe possibile effettuare uno studio su quale delle due condizioni risulti più efficace e se una delle due includa anche l'altra. Nel caso in cui si decidesse di utilizzare tali formule sarebbe richiesta l'elaborazione della variabile σ_m durante la fase della *non-overlapping window*.

Infine, la formula (4) include due condizioni sul valor medio $\overline{m}$. Come nel caso della formula (5), anche qui è richiesto il calcolo dei parametri m_{Max} e m_{min}.

Evidentemente, nel caso in cui si decidesse di calcolare il valore di deviazione standard σ_m sarebbe implicito che anche il valore di $\overline{m}$ venga calcolato. La determinazione del solo valor medio risulterebbe più leggero per evitare elaborazioni massicce di dati da parte dei singoli dispositivi. Un'altra scelta sarebbe optare per il calcolo del valore di σ_m evitando di includere la variabile $\overline{m}$ nella finestra per alleggerire il carico d'informazioni da inviare al server al prezzo di un aumento di elaborazione sui diversi dispositivi.

5.4. ***Conclusioni***

Dall'analisi effettuata con un programma scritto ad hoc realizzato in Matlab per la rappresentazione grafica dei dati, i risultati ottenuti evidenziano una sostanziale differenza nella misura dello $strength$ con il variare delle condizioni del manto stradale, indipendentemente dal cellulare utilizzato. Le differenze sostanziali tra i dispositivi riguardano i valori riscontrati; in particolare, il dispositivo contrassegnato col colore arancione risulta presentare in media le vibrazioni più basse. Nonostante questa particolarità, le vibrazioni registrate dai vari dispositivi nelle strade B e C tendono a raddoppiare e triplicare rispetto a quelle della strada A.

Poichè i dati ottenuti sono sensibili anche alla qualità, in termini di prestazioni, dell'hardware e del software del dispositivo utilizzato per effettuare le misure, un'analisi e una eventuale correzione del loro effetto sui risultati prodotti sarebbe auspicabile per evitare che i risultati finali possano diventare inaffidabili.

È stato inoltre formulato e distribuito uno sviluppo matematico (**5.3**) che potrà essere utilizzato per uno studio più organizzato e per modifiche dell'applicazione in via di sviluppi futuri.

Conclusioni

L’evoluzione in ambito tecnologico dei cellulari e dei dispositivi portatili in generale ha consentito lo sviluppo di applicazioni che vanno oltre la comunicazione o il semplice scambio di dati; in questa tesi, il ricorso ai dati forniti dai sensori interni di un comune smartphone ha prodotto la mappatura del manto stradale.

Per la struttura del sistema, si è sviluppato un prototipo in Android. Grazie alla versatilità della comunicazione mediante invio e ricezione di file *JSON* si garantisce la portabilità in diversi sistemi operativi, come ad esempio *iOS*. D’altronde, essendo Android un sistema open-source, è possibile estendere l’applicazione ad altre piattaforme.

I risultati ottenuti da questo processo sembrano incoraggiarne l’uso come strumento di misura anche se, durante la prova, i dispositivi sono stati utilizzati solo per raccogliere dati. per altre operazioni da parte dell’utente. Non è da escludere che l’utente, utilizzando normalmente il suo smartphone durante la prova, possa compromettere con il movimento la bontà dei risultati. In questo caso sarebbe opportuno migliorare il sistema così da distinguere i movimenti associati alla strada da quelli prodotti dall’utente.

La vettura su cui erano presenti gli smartphone viaggiava ad una velocità relativamente costante, essa è un fattore che influisce direttamente sulle distanze spaziali tra i diversi campioni. A parità di passo di campionamento τ, viaggiare ad una velocità v_2 maggiore di un’altra velocità v_1 comporta un aumento delle distanze tra i campioni pari a $(v_2 - v_1)\tau$. Inoltre, l’applicazione è affetta da una attenuazione delle vibrazioni da parte degli ammortizzatori delle vetture stesse, che sarebbe opportuno valutare e compensare per rendere oggettiva la prova.

Anche le condizioni meteorologiche potrebbero falsare le informazioni ottenute, in particolare nel caso di grandine o di semplice pioggia. Per aumentare l’affidabilità dei dati acquisiti sarebbe dunque necessario tener conto delle condizioni meteorologiche in un intorno del periodo in cui è svolta la prova.

Effettuare studi più approfonditi riguardanti le caratteristiche metrologiche permetterebbe un’analisi al fine di determinare ulteriori forme di rumore a cui gli smartphone sono

soggetti. Dai risultati ottenuti è stato solo possibile filtrare finestre generate da campioni non sufficientemente correlabili tra loro.

Tenendo conto dell'inevitabilità di alcuni imprevisti che possono presentarsi in determinati contesti, il sistema si è dimostrato in grado di sfruttare gli smartphone come strumenti di supervisionamento della superficie del manto stradale.

Il sistema realizzato si propone come una ulteriore feature di *Google maps*, che si aggiungerebbe a quelle di geolocalizzazione e navigazione già implementati. Unendo le funzionalità di mappatura del manto stradale con quelle dell'applicazione *maps*, potrebbe essere possibile, da parte dell'utente, attraverso le informazioni relative alle conformazioni stradali, la scelta di un percorso per indice di confort. Sarebbe inoltre possibile fornire una mappa d'interesse per servizi comunali e provinciali al fine di monitorare e manutenere particolari strade d'interesse.

In conclusione, lo smartphone è un componente molto utile che potrebbe trovare numerose applicazioni in diversi ambiti. Utilizzare gli smartphone al massimo delle proprie capacità e possibilità potrebbe comportare lo sviluppo di applicazioni utili e significative per l'utente finale, aprendo la strada ad un mondo più semplice e a realtà ancora inesplorate. In particolare, le applicazioni pratiche degli smartphone in ambito di controllo stradale potrebbero migliorare la viabilità del traffico cittadino. Grazie alle potenzialità del GPS e con la giusta app, sarebbe possibile avere una mappa in tempo reale di potenzialmente tutte le vetture in circolo, le quali potrebbero essere smistate in diverse strade per evitare ingorghi. Inoltre, conoscendo i vari flussi del traffico, sarebbe possibile rendere dinamiche le varie situazioni semaforiche per permettere un'equa distribuzione del traffico in presenza di semafori. Con le funzionalità dell'applicazione realizzata nello svolgimento della tesi, una giusta distribuzione del traffico permetterebbe un'equa mappatura delle condizioni del manto stradale. Avendo quindi il controllo del flusso stradale con una sufficiente mappatura della superficie del suo manto sarebbe possibile cambiare i sensi di marcia o aggiungere strade di supporto per favorire la viabilità cittadina e potendo effettuare manutenzioni mirate, qualora fossero necessarie.

Bibliografia

[1] Joint Task Force, "*Security and Privacy Controls for Federal Information Systems and Organizations",* 2015, pagina 94
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

[2] J. Sinclair, "*Collins COBUILD Advanced Learner's Dictionary*" Zanichelli, 2018

[3] "*Mobile Operating System Market Share Worldwide*" consultato il giorno 25/08/2021 https://gs.statcounter.com/os-market-share/mobile/worldwide

[4] Mauro Capelli, "*portabilita*", Treccani Enciclopedia della Scienza e della Tecnica., 2008, https://www.treccani.it/enciclopedia/portabilita_%28Enciclopedia-della-Scienza-e-della-Tecnica%29/

[5] K. Lamhaddab , M. Lachgar , K. Elbaamrani, "*Porting Mobile Apps from iOS to Android: A Practical Experience*", Hindawi Publishing Corporation , 2019

https://www.hindawi.com/journals/misy/2019/4324871/

[6] B.Stewart, "*Portability Study of Android and iOS*", 2012

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.923&rep=rep1&type=pdf

[7] Treccani "*open source*", consultato il giorno 30/08/2021
https://www.treccani.it/enciclopedia/open-source

[8] "*Codenames, Tags, and Build Numbers*", consultato il giorno 31/08/2021

https://source.android.com/setup/start/build-numbers?hl=en

[10] Roberto Steindler, "*Sensori*", Treccani Enciclopedia della Scienza e della Tecnica., 2008, https://www.treccani.it/enciclopedia/sensori_%28Enciclopedia-della-Scienza-e-della-Tecnica%29/

[11] " Manish Kumar Mishra, Vikas Dubey, P. M. Mishra and Isharat Khan, "*MEMS Technology: A Review*", 2019,
https://www.journaljerr.com/index.php/JERR/article/view/16891/31405

[12] "*Sensor*", consultato il giorno 02/09/2021
https://developer.android.com/reference/android/hardware/Sensor

[13] "*Mobile & Tablet Android Version Market Share Worldwide*", consultato il giorno 02/09/2021 https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[14] T. Kuhlmann, P. Garaizar & U. Reips , "*Smartphone sensor accuracy varies from device to device in mobile research: The case of spatial orientation*" 2020 https://link.springer.com/article/10.3758/s13428-020-01404-5

[15] "*Sensor types*", consultato il giorno 05/09/2021 https://source.android.com/devices/sensors/sensor-types

[28] Q. Ma, H. Yang, A. Mayhue, Y. Sun, Z. Huang, Y. Ma , "*Accident Analysis and Prevention* ", 2021, https://www.sciencedirect.com/journal/accident-analysis-and-prevention

[29] "*Simple, battery-efficient location"*, consultato il giorno 15/09/2021 https://developers.google.com/location-context/fused-location-provider

[32] *"FusedLocationProviderClient"*, consultato il giorno 17/09/2021 https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

[33] *"LocationRequest"*, consultato il giorno 17/09/2021 https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

[34] *"LocationCallback"*, consultato il giorno 17/09/2021 https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

[35] "*SensorManager*", consultato il giorno 29/09/2021

https://developer.android.com/reference/android/hardware/SensorManager

[36] "*SensorEvent*", consultato il giorno 29/09/2021

https://developer.android.com/reference/android/hardware/SensorEvent

[37] "*SensorEventListener*", consultato il giorno 29/09/2021

https://developer.android.com/reference/android/hardware/SensorEventListener

[38] "*Sensor*", consultato il giorno 29/09/2021

https://developer.android.com/reference/android/hardware/Sensor

[41] "*Delete*", consultato il giorno 30/09/2021

https://developer.android.com/reference/androidx/room/Delete

[42] "*RoomDatabase*", consultato il giorno 30/09/2021

https://developer.android.com/reference/androidx/room/RoomDatabase