Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica

Progettazione ed implementazione di un sistema di navigazione in realtà aumentata per motocicli

Relatore:
Chiar.mo Prof.
Marco Di Felice
Correlatore:
Ing. Francesco Rambaldi

Presentata da:
Francesco Draisci

III Sessione
Anno Accademico 2014/2015

La mente che si apre ad una nuova idea
non torna mai alla dimensione precedente.
Albert Einstein

Introduzione

Durante gli ultimi decenni lo sviluppo dei sistemi informatici ha subito un forte incremento. Questo processo di sviluppo tecnologico ha dato la possibilità di aumentare le capacità di gestione delle informazioni. In particolare, oggi abbiamo la possibilità di ricevere costantemente un enorme quantità di informazioni di nostro interesse, grazie ai dispositivi mobile costantemente connessi alla rete Internet.

Con l'aumentare delle informazioni a disposizione dell'utente, si presenta il bisogno di nuove tecnologie per la loro fruizione in modo efficiente e veloce. Nascono, così, nuovi metodi di impiego dei dati, a volte distanti dal concetto di utilizzo di un personal computer. Un esempio è l'*Ubiquitus Computing* (o ubicomp): un modello di interazione uomo-macchina, in cui l'elaborazione delle informazioni è stata completamente integrata negli oggetti di vita quotidiana. L'ubicomp comprende un'ampia gamma di argomenti di ricerca, tra cui il calcolo distribuito, il mobile computing, i Wireless Sensor Network (o WSN), l'interfaccia uomo-macchina e l'intelligenza artificiale.

Ad oggi le tecnologie utilizzate per il controllo e l'elaborazione dei dati sono molte e sempre più alla portata di tutti. Quanto appena detto, ci permette di introdurre l'argomento di studi di questa tesi: la realtà aumentata.

La realtà aumentata (o Augmented Reality, abbreviato AR), da la possibilità di arricchire la visione del mondo fisico (reale, percepito) con informazioni digitali (virtuali), rendendo le informazioni stesse parte del mondo che ci circonda. Anche se questa tecnologia può sembrare abbastanza recente, per circa 40 anni è stata oggetto di studio da parte di molti ricercatori ed ha

stuzzicato l'immaginazione in molte pellicole di fantascienza.

L'ubiquitous computing e i dispositivi indossabili (weareble device) sembrano essere il veicolo perfetto per la diffusione della realtà virtuale e della realtà aumentata.

"Wearable computing and ubiquitous computing are the latest solutions in mobile virtual reality. These technologies together with wireless data transmission enable both augmented reality and multimedia in mobile environments. Wearable and ubiquitous computing are energetically finding their position in the future information society."[SSVK99] ["Il wearable computing e l'ubiquitous computing sono le più recenti soluzioni in ambito di realtà virtuale mobile. Queste tecnologie, insieme alla trasmissione di dati senza fili consentono sia la realtà aumentata che la multimedialità in ambienti mobile. Il wearable computing e l'ubiquitous computing stanno energicamente trovando la loro posizione nella futura società dell'informazione."]

È di facile intuizione che il concetto di realtà aumentata sia molto vicino alla realtà virtuale, tecnologia attualmente utilizzata in ambito video-ludico e di simulazioni grafiche. La principale differenza tra realtà virtuale e realtà aumentata sta nel modo in cui le informazioni vengono presentate all'utente. La realtà virtuale, infatti, utilizza un mondo digitale immersivo, creato per ingannare i sensi di chi lo utilizza, facendogli vivere una *realtà* (Il termine realtà qui non rappresenta il mondo fisico) creata dalla macchina. La realtà aumentata utilizza invece la realtà fisica che circonda l'utente, catturata tramite videocamera o con particolari visori, e proietta in essa oggetti digitali.

Oggi la realtà aumentata non è più un sogno fantascientifico. Immaginare di camminare per strada ampliando la quantità di informazioni a nostra disposizione, semplicemente guardando oggetti o trovandosi in un certo punto geografico, non ci sembra più così fuori dal comune. Un esempio può essere quello dei *Google Glass*, prodotti dal colosso di Mountain View. Degli occhiali *smart* in grado di interagire con lo smartphone e di visualizzare tutti i dati di interesse. Molto simile è il progetto *HoloLens* dell'azienda Microsoft. Tali

strumenti vengono chiamati *smart-glasses,* una delle più conosciute tipologie di *wearable device.* Le possibili applicazioni di questa innovativa tecnologia sono molto numerose.

0.1 Obiettivi

Questa tesi ha l'obiettivo di mostrare i fondamenti per lo sviluppo di un sistema di navigazione per caschi motociclistici in realtà aumentata[1].

L'applicazione implementata sfrutta i concetti principali di realtà aumentata *sensor based,* cioè basata su geo-localizzazione, al fine di fornire i dati di interesse all'interno del campo visivo del guidatore. Lo scopo del progetto è di realizzare un sistema in grado di interagire con l'utente attraverso i suoi movimenti, e rendere fruibili le informazioni riguardanti la navigazione all'interno di un casco.

Non sono pochi i vantaggi che questi strumenti potrebbero introdurre nella guida veicolare, anche in ambito di sicurezza stradale. Infatti, in questo modo, l'utilizzatore del casco non sarà più costretto a distrarsi dalla guida per consultare le informazioni del percorso da seguire, ma avrà la possibilità di vederle proiettate direttamente all'interno del suo campo visivo.

Tutte le informazioni che oggi siamo abituati a ricevere da un comune navigatore satellitare (o dal nostro smartphone), saranno disponibili nella visione reale del mondo che ci circonda in modo rapido e intuitivo.

Si è scelto di utilizzare *Android* come sistema operativo per lo sviluppo del sistema, utilizzando la libreria *droidAR* per la realtà aumentata. Il progetto è composto da due applicazioni. La prima, installata nel casco, svolge il ruolo di visualizzatore delle informazioni in AR. La seconda, installata in uno smartphone, fornisce le informazioni necessarie. Le due parti utilizzano

[1]Il progetto ARMH (Augmented Reality Motorbike Helmet) è nato da un'idea di ricerca e sperimentazione presso l'azienda Raleri srl, che da anni opera nel settore ottico-sportivo, con l'intento di proporre soluzioni efficaci e tecnologicamente innovative.

un canale di comunicazione wireless per lo scambio di dati relativi a geolocalizzazione, destinazione, percorso, notifiche e settaggi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

La realtà aumentata

All'interno di questo capitolo, verrà fatta una breve panoramica sull'evoluzione della realtà aumentata, cercando di sottolineare le differenze che la separano dalla realtà virtuale. Verranno, inoltre, descritti i diversi metodi di sviluppo e le tecnologie attualmente a disposizione. Senza tralasciare alcuni esempi di impiego.

1.1 Cenni storici

La realtà aumentata fa la sua prima comparsa nel 1957 , con lo sviluppo da parte del regista, filosofo e inventore Morton Heilig di una macchina chiamata *Sensorama*[1]. Fu brevettata nel 1962 [Hei61]. Si trattava di un simulatore multi-sensoriale cinematografico, in grado di dare all'utente l'illusione di essere effettivamente nella proiezione. Era provvisto di un sistema di proiezione delle immagini in 3D, vibrazioni, audio stereofonico, vento, riproduzione di stimoli olfattivi e tattili. A causa del suo costo eccessivo, il progetto restò in fase di sperimentazione.

[1] http://www.mortonheilig.com/InventorVR.html

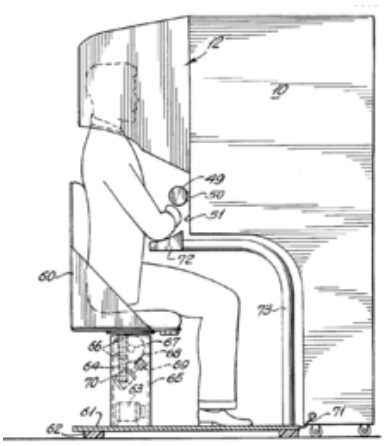

Figura 1.1: The Sensorama,da U.S. Patent #3050870

Nel 1966 Sutherland crea il primo prototipo di Head-Mounted Display (HMD). Viene considerato il primo dispositivo indossabile per la realtà virtuale e la realtà aumentata. Un visore da indossare sulla testa, dotato di display ottici integrati. Lo strumento era progettato per visualizzare le immagini provenienti da un computer. Tali immagini venivano elaborate e mostrate in base ai movimenti effettuati dall'utente. Il prototipo creato da Sutherland era primitivo, sia in termini di interfaccia utente, sia per quanto riguarda il realismo. Tra l'altro, il peso eccessivo costringeva all'utilizzo di un braccio meccanico come supporto all'apparecchio. Tale tecnologia si è sviluppata negli anni ed è impiegata oggi in ambito militare, civile e ludico.

Figura 1.2: Un esempio di Head-Mounted display (HMD)

Negli anni 70 Myron Krueger crea Videoplace, in grado di realizzare una *realtà artificiale*[2] fruibile senza l'ausilio di visori, dando la possibilità a più utenti contemporaneamente di interagire con oggetti virtuali. La figura 1.3 illustra Videoplace in esecuzione.

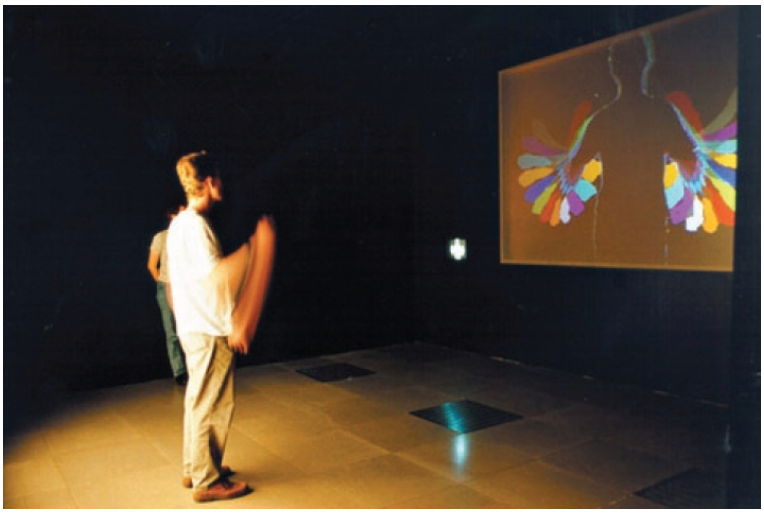

Figura 1.3: Immagine tratta da una dimostrazione di Videoplace, da: `http://people.ucsc.edu/~joahanse/onlineexhibit/videoplace/`

> "In VIDEOPLACE il computer aveva controllo sull'interazione tra il partecipante e gli oggetti grafici sullo schermo. Il computer poteva coordinare il movimento di un oggetto con le azioni del partecipante senza considerare necessariamente i limiti della realtà fisica: se la gravità agisce sul corpo fisico, nella *realtà artificiale* non riesce a controllare o a confinare l'immagine che avrebbe potuto galleggiare se fosse stato necessario.[...]"[3]

Anche se da più di 40 anni studiosi e ricercatori sono a lavoro su questo tipo di tecnologia, il concetto di *Realtà Aumentata* (Augmented-Reality) viene

[2]Termine coniato dallo stesso Krueger.

[3]

Tratto da: `http://www.edueda.net/index.php?title=Videoplace`

coniato solo nel 1990, dal ricercatore Tom Caudelle durante il suo lavoro presso l'azienda Boeing.

Nel corso degli anni 90 molti ricercatori cercano di dare alla realtà aumentata una definizione effettiva che la distinguesse dalla realtà virtuale. Uno dei contributi più importanti venne dato nel 1994 da Paul Milgram, docente dell'Università di Toronto, e Fumio Kishino, dell'Università di Osaka. I quali cercarono di spiegare il rapporto tra le due tecnologie collocandole all'interno di un *Reality-Virtuality Continuum* mostrato in figura 1.4, creando il concetto di *Mixed Reality*.

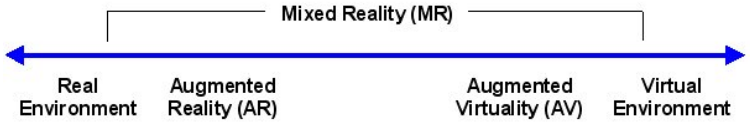

Figura 1.4: Il Reality-Virtuality Continuum di Milgram-Kishino, da: `https://en.wikipedia.org/wiki/Reality-virtuality_continuum`

All'interno del Continuum si collocano agli antipodi il mondo reale, quello che tutti conosciamo composto da oggetti fisici, e il mondo virtuale, che non esiste nella realtà poiché è stato creato da un calcolatore elettronico.

Tra i due vertici c'è la Realtà Mixata (o mista), che mescola oggetti fisici della realtà con oggetti virtuali creati dal calcolatore. Muovendosi all'interno del continuum, la realtà mista più prossima al mondo reale viene definita realtà aumentata, mentre all'opposto troviamo la virtualità aumentata.

Nel 1992 Louis Rosenberg, agli USAF Armstrong Labs, crea *Virtual Fixtures,* il primo sistema totalmente immersivo di Realtà Aumentata. Il progetto era composto da un visore indossabile e un esoscheletro governabile attraverso i movimenti delle braccia. L'esoscheletro permetteva di gestire delle braccia robotiche, mostrate dal visore. Il sistema dava la possibilità a chi lo utilizzava di incrementare le performance nello svolgimento di task locali o remoti. Dispositivi simili sono oggi impiegati per effettuare compiti che richiedo un'estrema precisione, anche a distanza. Ad esempio, è note-

vole l'ausilio in ambito chirurgico di tali prodotti, grazie al quale è possibile limitare il margine d'errore umano, oppure operare pazienti a chilometri di distanza.

1.2 La realtà aumentata oggi

Negli anni il legame tra la realtà aumentata e i dispositivi indossabili si è fatto sempre più saldo. Questi dispositivi infatti rappresentano uno dei più efficaci strumenti per sfruttare tale tecnologia.

Di seguito verranno illustrati brevemente alcuni progetti che illustrano l'evoluzione dell'AR negli ultimi anni.

1.2.1 Google Glass

Google Glass[4] è un programma di ricerca e sviluppo dell'azienda Google Inc., che ha come obiettivo lo sviluppo di un paio di occhiali in grado di sfruttare la realtà aumentata. I primi occhiali prodotti sono stati resi disponibili per gli sviluppatori nell'anno 2013. Nel 2014, invece, il prodotto è stato reso disponibile al pubblico negli Stati Uniti, e successivamente in Europa.

I Google Glass sono basati sul sistema operativo Android. Sono dotati di un display dalle dimensioni molto ridotte e una risoluzione elevata, tramite il quale l'utente riceve le informazioni visive. Inoltre, sono presenti: una fotocamera, per l'acquisizione di immagini e video, un impianto audio a conduzione ossea, connettività WiFi e Bluetooth.

[4]Pagina del progetto: `https://www.google.com/glass/start/`

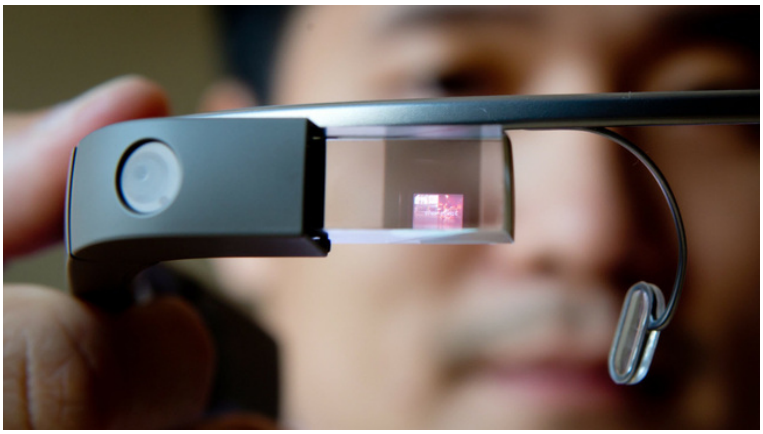

Figura 1.5: Google Glass, da: `http://www.wired.it/attualita/tech/2014/10/15/dice-davvero-studio-sulluomo-dipendente-dai-google-glass/`

L'interazione tra l'utente e il dispositivo può avvenire tramite comandi vocali o touchpad (posizionato sull'astina destra dell'occhiale).

Chi li indossa potrà effettuare ricerche e visitare siti Web, leggere le notizie online, controllare i social network, avviare videoconferenze, telefonare, visualizzare i messaggi , scattare fotografie, registrare video e molto altro ancora.

1.2.2 HoloLens

HoloLens[5] è un progetto sviluppato da Microsoft, avviato nel 2010 con il nome di *Progetto Baraboo.*

Il device è progettato per sfruttare le potenzialità di *Windows Holografic*, una piattaforma di realtà aumentata creata dalla stessa Microsoft. HoloLens è basato sul sistema operativo Windows 10 e caratterizzato da un insieme di sensori avanzati. È infatti corredato di un display ottico 3D ad alta definizione ed un sistema di scansione spaziale dei suoni, i quali consentono a chi li indossa di poter fruire di applicativi di realtà aumentata attraverso una

[5]Pagina del progetto: `https://www.microsoft.com/microsoft-hololens/en-us`

interfaccia olografica, con la quale è possibile interagire mediante sguardo, voce e gesti delle mani (Figura 1.6). HoloLens, all'interno del suo comparto hardware, possiede una Holographic Processing Unit (HPU), un coprocessore che integra i dati dai vari sensori e gestisce attività come la mappatura del territorio, il riconoscimento dei gesti ed il riconoscimento vocale.

Figura 1.6: Microsoft HoloLens, da: `https://www.microsoft.com/microsoft-hololens/en-us`

1.2.3 Skully AR-1

Skully AR-1[6] è stato progettato dall'azienda californiana Skully Helmet e sviluppato grazie ad una campagna di *crowdfunding* sulla nota piattaforma Indigogo. Si tratta di un casco hi-tech che permette al motociclista di consultare informazioni senza staccare gli occhi dalla strada.

[6]Sito del produttore: `http://www.skully.com/`

Figura 1.7: Skully AR-1, da: `http://www.jebiga.com/skully-ar-1-motorcycle-helmet/`

Il casco smart è basato sul sistema operativo Android. È inoltre caratterizzato da un HUD (heads-up display) trasparente integrato, che permette di consultare informazioni sovrapposte al campo visivo (Figura 1.7). In questo modo è possibile ricevere indicazioni sulla direzione da seguire per giungere a destinazione, o visualizzare le notifiche sulle chiamate ricevute. La comunicazione con lo smartphone avviene attraverso il modulo Bluetooth incluso.

Le funzionalità implementate in Skully AR-1 sono:

- un sistema HUD che mostra le informazioni ad una distanza percettiva di circa 3 metri di fronte al motociclista.
- visione posteriore, attraverso una fotocamera con campo di acquisizione di 180°.
- navigazione GPS turn-by-turn, con possibilità di visualizzare una mappa.
- controllo vocale per le interazioni con l'untente.

Si tratta del primo casco al mondo ad impiegare la realtà aumentata come supporto alla guida motociclistica.

1.3 Come funziona la realtà aumentata

Tutte le applicazioni di realtà aumentata, per quanto diverse possano essere, sono accomunate da alcune caratteristiche [BBKM02]:

- Un software per il rendering grafico degli oggetti virtuali.
- Un sistema di tracking e/o sensing.
- Uno o più oggetti virtuali da inserire nella visualizzazione.
- Un dispositivo in grado di visualizzare contenuti digitali.

Attualmente le metodologie di sviluppo per questa tecnologia possono essere classificate in due categorie, distinte dalla tipologia di tracking utilizzata:

- Visual-tracking : basato sul riconoscimento di figure stilizzate (marker) o oggetti, a tal fine vengono impiegate tecniche di analisi dell'immagine e *feature detection* [7].
- Sensor-tracking : si utilizzano i sensori di posizionamento, per lo più GPS, per collocare oggetti in determinate aree geografiche.

Le due categorie non differiscono solo per i caratteri implementativi, ma anche per le tipologie di applicazione. Mentre la prima è di più ampio utilizzo in ambito di *comunicazione,* soprattutto pubblicitaria. La seconda ha un impiego circoscritto alle applicazioni *outdoor,* proprio a causa dei limiti ereditati dalla tecnologia di posizionamento GPS, spesso utilizzata in ambito di navigazione. Questa divisione grossolana, però, non costituisce una regola.

[7] In computer vision, e nell'elaborazione digitale delle immagini, il concetto di rilevamento di caratteristiche (feature detection) o riconoscimento di caratteristiche racchiude una serie di metodi per l'estrapolazione di informazioni da una immagine e per prendere decisioni locali sull'esistenza o meno di una caratteristica in quel determinato punto. Le caratteristiche risultanti saranno un sottoinsieme del dominio dell'immagine, spesso in forma di punti isolati, curve continue o regioni connesse. [`https://it.wikipedia.org/wiki/Rilevamento_di_caratteristiche`]

1.3.1 Realtà aumentata visual-based

Molti degli applicativi disponibili per l'utilizzo di tale tecnologia, sono basati sul concetto di *fiducial marker*[8]. Questo tipo di soluzione è considerata necessaria per gli applicativi pensati per l'utilizzo indoor. Infatti, in tali contesti, è quasi impossibile utilizzare la localizzazione satellitare, a causa delle barriere fisiche rappresentate dagli edifici. Esistono, però, metodi alternativi alla localizzazione in ambienti chiusi. Come il posizionamento tramite reti wireless, ma non entreremo nel dettaglio.

I marker sono figure bidimensionali (vedi figura 1.8), di solito in bianco e nero, e permettono il tracking degli oggetti virtuali all'interno di uno spazio tridimensionale.

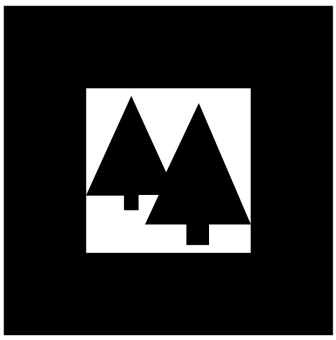

Figura 1.8: Marker

Il dispositivo è in grado di leggere il marker, tramite l'uso della fotocamera, e associa ad esso dei contenuti 3D virtuali. Il software elabora i movimenti del marker all'interno di un sistema di riferimento, così da riposizionare di conseguenza l'oggetto associato nello spazio.

[8]Concetto introdotto da Reikimoto nel 1996.

Figura 1.9: Esempio di utilizzo dei marker

Alcune softwarehouse hanno sviluppato librerie che integrano sistemi di *Visual Search* e *Natural Feature Tracking*, che vanno ben oltre il riconoscimento di figure stilizzate. Nel primo caso, il software è in grado di reperire informazioni su un oggetto, inviando delle immagini ad un *visual search engine* (un esempio è l'applicazione Google Goggles). Il Natural Feature Tracking può essere utilizzato in alternativa al fiducial marker, ma richiede risorse di calcolo più elevate. Questa tecnologia è in grado di posizionare il dispositivo all'interno del sistema di riferimento, attraverso il riconoscimento delle caratteristiche fisiche degli oggetti (come forma e dimensioni).

Figura 1.10: Esempio di Natural Feature tracking

1.3.2 Realtà aumentata sensor-based

L'esempio più comune di sensor-tracking è il Global Positioning System (GPS), un sistema di posizionamento terrestre particolarmente preciso. Il funzionamento del GPS è basato su un trasmettitore radio, il quale determina la posizione geografica attraverso l'interrogazione di diversi satelliti[9] (maggiore è il numero di satelliti agganciati, maggiore è la precisione).

Questa tecnologia venne creata per utilizzi militari, dal Ministero della Difesa degli Stati Uniti d'America, e successivamente introdotto nella navigazione veicolare.

[9]Per poter stabilire la posizione, sarà necessario interrogare almeno tre satelliti.

Figura 1.11: Immagine di un satellite GPS, da: `http://www.navcen.uscg.gov/ftp/gps/ggeninfo/gps-iif.tif`

Il sistema GPS permette di utilizzare le coordinate geografiche come sistema di riferimento. Attraverso il quale è possibile posizionare il device e gli oggetti virtuali nello spazio, mediante la latitudine e la longitudine. L'uso delle coordinate geografiche rende il tracking della realtà aumentata molto più esteso rispetto all'impiego del fiducial marker. Infatti, il tracking GPS non necessita di riferimenti visivi per inserire gli oggetti nell'AR, i quali potranno essere posizionati anche fuori dal campo visivo e ad una distanza elevata. Inoltre maggiore è l'accuratezza del GPS, maggiore sarà la precisione nel posizionamento degli oggetti all'interno dello spazio.

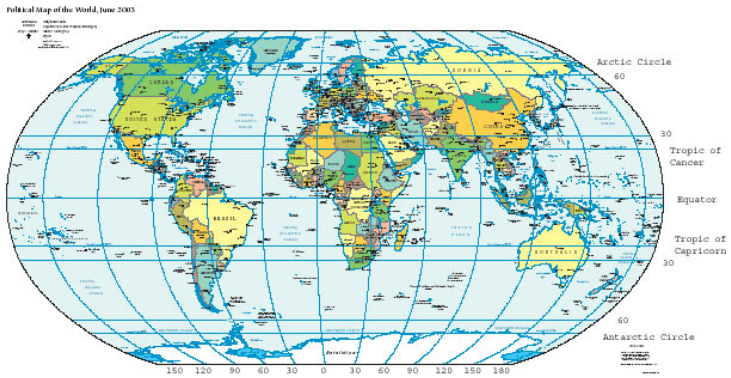

Figura 1.12: Mappa terrestre. Le linee verticali curve sono dette meridiani e servono a misurare la latitudine, le linee orizzontali sono dette paralleli e servono a misurare la longitudine., da: `https://commons.wikimedia.org/w/index.php?curid=561108`

Il sistema GPS è in grado di stabilire la posizione del trasmettitore in uno spazio bidimensionale. Infatti, in tale contesto il globo terrestre è considerato piatto, all'interno di un sistema a due assi (vedi figura 1.12). Per poter ampliare tale sistema in uno a tre assi, è necessario inserire una terza dimensione, che, in termini di coordinate spaziali terrestri, è indicata dall'altitudine. La misurazione dell'altitudine può essere effettuata attraverso dei sensori dedicati (altimetro). In alternativa può essere ottenuta consultando dei servizi online, creati per fornire l'elevazione di un punto geografico rispetto al *livello zero*[10], indicando latitudine e longitudine di tale locazione.

Infine, per rilevare la direzione verso la quale il dispositivo è orientato, vengono sfruttati una serie di sensori MEMS[11]. In particolare i sensori inerziali IMU (Internal Measurement Unit), integrati nella maggior parte dei dispositivi mobile. Questi sensori hanno la capacità di rilevare i movimenti che il dispositivo compie in uno spazio tridimensionale. Nello specifico, verranno utilizzati per calcolare le rotazioni rispetto ai tre assi del sistema di riferimento, così da riposizionare gli oggetti virtuali di conseguenza.

[10]Nella maggior parte dei casi questo è il livello del mare (m s.l.m.).

[11]`https://it.wikipedia.org/wiki/MEMS`

Capitolo 2

Stato dell'arte e librerie di realtà aumentata

2.1 Architettura di un'applicazione di realtà aumentata

Molte delle applicazioni riguardanti la realtà aumentata, seppur molto diverse nel dettaglio, condividono un'architettura generica di base . Al fine di delineare una struttura valida per ogni applicativo, Asa MacWilliams, Thomas Reicher, Gudrun Klinker e Bernd Bruegge progettarono e pubblicarono nel 2004 un *Design Pattern* [1] per i software di AR [BBKM02, BBKM03].

2.1.1 Design pattern per la realtà aumentata

Le funzionalità che accomunano ogni applicazione di questo genere sono: tracking della posizione dell'utente, mixing degli oggetti virtuali e reali e interazione con l'utente.

[1] `http://campar.in.tum.de/pub/macwilli2004patterns/macwilli2004patterns.pdf`

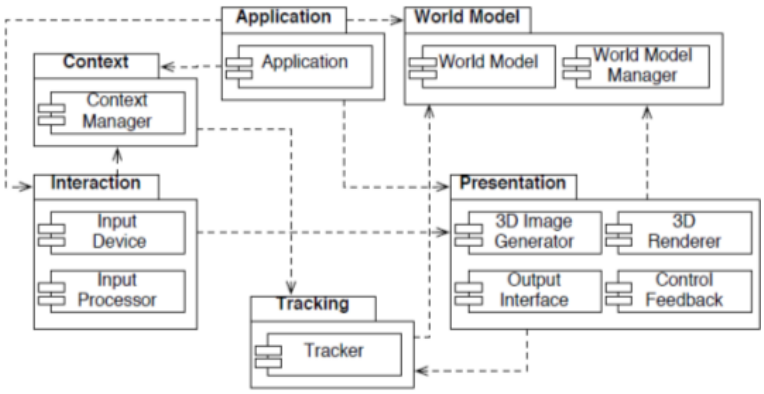

Figura 2.1: Struttura generica di un'applicazione di Realtà Aumentata

Il pattern fa riferimento a diversi *sottosistemi,* che nel complesso rappresentano un sistema di AR:

Application: Rappresenta l'astrazione dell'intera applicazione.

Interaction: L'*Interaction Subsystem* recupera tutti gli input generati dall'utente e li elabora.

Presentation: Mostra all'utente l'output grafico della realtà aumentata. Comprende immagini 3D, testo e audio.

Tracking: Il *Tracking Subsystem* è responsabile del tracciamento della posizione dell'utente, la elabora e la rende disponibile per gli altri subsystem.

Context: Raccoglie le varie tipologie di dati e le rende disponibili per i diversi subsystem.

World-Model: In un sistema di realtà aumentata, l'utente si muove all'interno del mondo reale e ottiene informazioni legate agli oggetti reali che ne fanno parte o alla sua posizione.Queste informazioni vengono memorizzate ed elaborate nel *World Model Subsystem.*

Come mostrato nella figura 2.1 ogni subsystem è composto da diversi componenti. Le linee tratteggiate rappresentano delle dipendenze tra subsystem, le quali indicano un'interfaccia tra gli stessi.

La struttura illustrata è molto simile al comune pattern *Model-View-Controller* (MVC) , il quale separa i compiti fra i tre moduli: *Model* fornisce i metodi per accedere ai dati utili all'applicazione, *View* visualizza i dati contenuti nel model e si occupa dell'interazione con l'utente, *Controller* gestisce i comandi dell'utente [DeS10].

Il design pattern in questione rappresenta un modello generico. In caso di funzionalità specifiche il pattern può essere ridimensionato in base alle esigenze dettate dallo sviluppo.

2.2 Librerie per la realtà aumentata

Negli ultimi anni sono state pubblicate numerose librerie e framework come ausilio alla creazione di applicativi di realtà aumentata. Quest'ultimi sempre più orientati al mondo dei sistemi per dispositivi mobile. Grazie, infatti, alla loro dotazione hardware e alla loro larga diffusione, smartphone e tablet rappresentano un ottimo vettore per lo sviluppo della tecnologia in questione.

Tra le varie librerie a disposizione degli sviluppatori, molte sono basate su progetti multi piattaforma. Offrono, quindi, dei tool in grado di sviluppare applicativi per sistemi desktop (Windows, Linux, OS X, ecc...) o sistemi mobile (Android, iOS).

Come già specificato, le tipologie di AR possono essere classificate come marker-based o sensor-based. Allo stesso modo esistono librerie orientate ad una o altra tipologia. Poche sono quelle in grado di sopperire ad un utilizzo polivalente, ancor meno quelle freeware o open-source.

Nella fase preliminare del progetto è stata svolta un'analisi sulle funzionalità messe a disposizione dalle librerie principali. Di seguito sono elen-

cati i criteri di valutazione, i quali rappresentano i principali fattori che influenzeranno lo sviluppo:

- Licenza: allo scopo prefisso saranno preferite librerie rilasciate con licenza open-source o freeware.

- Piattaforma software: come già specificato, molte librerie sono disponibili per diverse piattaforme software. La libreria scelta dovrà essere compatibile con il Sistema Operativo Android adoperato per lo sviluppo.

- GPS: l'applicativo che si andrà a sviluppare sarà basato sulla geolocalizzazione. Tutte le librerie prese in considerazione dovranno gestire in modo efficiente il sistema GPS.

- Internal Measurement Unit (IMU): il posizionamento degli oggetti virtuali e alcune delle interazioni con l'utente saranno gestite in base ai movimenti del dispositivo. A tal scopo, la presenza di moduli dedicati alla gestione dei sensori inerziali (giroscopio, accelerometro e magnetometro) sarà ritenuto un fattore fondamentale.

- Documentazione: una ricca documentazione offre ausilio nello studio di tale tecnologia, oltre allo sviluppo dei progetti. Sarà considerato un valore aggiunto.

2.2.1 ArToolKit

ArToolKit[2] è uno dei principali framework gratuiti di realtà aumentata, disponibile per diversi ambienti di sviluppo. Tra i quali Mac OS X, Windows, Linux, Android e iOS. Venne realizzato e rilasciato da Hirokazu Kato e Mark Billinghurst nel 1999 e può essere considerata uno dei primi strumenti di sviluppo di realtà aumentata. Un esempio di applicazione di

[2] `http://artoolkit.org/`

ARToolKit è *ARQuake*, una versione in AR del popolare videogioco *Quake*, sviluppato presso i Wearable Computer Lab dell' University of South Australia nel 2000 [Tho02].

Le caratteristiche di ARToolKit sono:

- Visual-tracking, compreso l'utilizzo di natural feature.
- Calibrazione della fotocamera.
- Tracking simultaneo e supporto alla fotocamera stereoscopica.
- Multi-linguaggio.
- Ottimizzazione per i dispositivi mobile.
- Supporto di Unity3D e OpenSceneGraph.

Questa libreria attualmente supporta solo il tracking attraverso fiducial marker o natural feature. ArToolKit elabora i frame ricevuti dalla fotocamera alla ricerca di una corrispondenza con il marker, in caso di successo renderizza e sovrappone ad esso un immagine tridimensionale.

La documentazione contiene una descrizione completa della libreria ARToolKit, come installarlo e come utilizzare le funzionalità. Questa libreria è open-source e rilasciata con doppia modalità di licenza, GPL e commerciale.

Sebbene l'affinità di tale libreria con gli obiettivi del progetto siano minimi, è stato ritenuto opportuno analizzarla al fine didattico.

2.2.2 Metaio SDK

Metaio SDK[3] è uno dei framework modulari più completi per lo sviluppo della realtà aumentata. Sviluppata dalla omonima azienda, fondata nel 2003. Il Software Developement Kit è disponibile per diversi ambienti di

[3] `https://my.metaio.com/dev/sdk/documentation/metaio-sdk-framework/index.html`

sviluppo in una versione gratuita (con restrizioni) e una versione a pagamento. A maggio del 2015 Metaio è stata acquisita dalla Apple Inc, che ne ha bloccato il rilascio e gli aggiornamenti.

Metaio SDK permette la realizzazione di applicazioni basate su tracking GPS, riconoscimento di marker, analisi di immagini e modelli 3D. Metaio oltre al SDK mette a disposizione anche altri strumenti di sviluppo. Junaio è un *Augmented Reality Browser* in grado di visualizzare contenuti di realtà aumentata reperibili tramite internet. Metaio Creator da la possibilità di creare applicazioni di realtà aumentata in modo veloce ed intuitivo, anche senza nessuna conoscenza di programmazione.

La struttura del framework è costituita da diversi moduli:

- Capturing component: si occupa della gestione delle immagini catturate dalla fotocamera.
- Sensor interface component: si occupa della gestione dei sensori e dei dati da essi forniti.
- Rendering component: renderizza le immagini virtuali da visualizzare nella realtà aumentata.
- Tracking component: monitora la posizione del dispositivo e degli oggetti virtuali nel sistema di riferimento.
- Metaio SDK interface: è l'interfaccia con il quale l'applicazione ha la possibilità di interagire con gli altri quattro componenti.

La figura 2.2 mostra la struttura del framework.

In base a questa struttura, lo sviluppatore non ha un accesso diretto all'implementazione delle componenti. Infatti il loro utilizzo è basato su delle semplici API del SDK, che permetto l'integrazione di tali funzionalità all'interno dell'applicazione.

La compatibilità di Metaio SDK con gli ambienti di sviluppo è molto vasta e comprende Android, IOS, Unity3D e Windows.

La documentazione della libreria è ricca e dettagliata. Comprende, infatti, descrizioni e esempi per le diverse architetture.

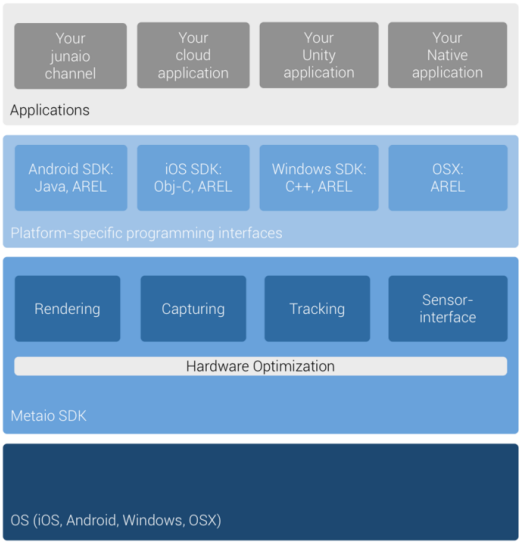

Figura 2.2: Struttura del framework Metaio SDK, da: `https://my.metaio.com/dev/sdk/documentation/metaio-sdk-framework/index.html`

2.2.3 Wikitude SDK

Wikitude SDK[4] è stato progettato e sviluppato da Wikitude GmbH. È disponibile in versione trial (con notevoli limitazioni) o commerciale. Anche Wikitude offre la possibilità di sviluppare applicazioni basate sul posizionamento geografico del device e sul riconoscimento di immagini (target). In particolare, il sistema di riconoscimento delle immagini, sfrutta il natural feature tracking, ma non è disponibile un visual search engine. Il sistema di localizzazione è basato sul posizionamento geografico del device attraverso GPS o WiFi, oltre ai sensori IMU. Sfruttando la localizzazione e i sensori, vengono elaborate le relative posizioni degli oggetti.

[4] `http://www.wikitude.com/`

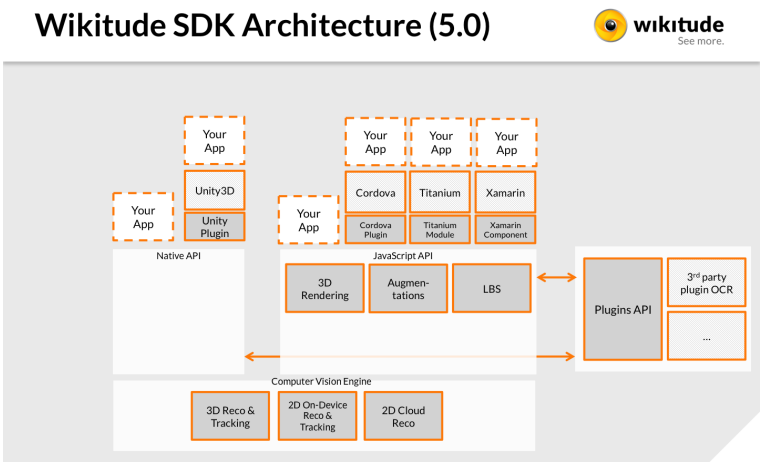

Figura 2.3: Struttura del framework Wikitude SDK, da: `http://www.wikitude.com/developer/documentation/android`

La figura 2.3 mostra gli aspetti dell'architettura utilizzata dal Software Developer Kit Wikitude.

Le componenti di tale struttura sono:

- Computer Vision Engine: Il motore di computer vision è un componente fondamentale della Wikitude SDK e utilizzato da tutte le piattaforme. Non è direttamente accessibile, ma va sfruttato attraverso l'API nativa o l'API JavaScript.
- API Native: Fornisce l'accesso al motore di computer vision nativo per Android (Java) e iOS (objc). È anche possibile caricare i plugin tramite l'API.
- API JavaScript: permette di costruire mondi di realtà aumentata sulla base di HTML e JavaScript. È disponibile per Android e iOS. L'API JavaScript consente di accedere alle funzionalità del motore di computer vision, location-based AR, l'API di Plug-in e funzionalità di rendering dedicato.
- Plugin API: una API per collegare i propri plugin al Wikitude SDK.

- Cordova Plugin: In cima alla API JavaScript, il plugin Cordova permette di utilizzare Wikitude SDK in combinazione con Apache Cordova.
- Titanium Module: In cima alla API JavaScript, permette di utilizzare Wikitude SDK in combinazione con Titanium.
- Unity Plugin: In cima alla API nativo, il plugin Unity permette di utilizzare Wikitude SDK in combinazione con Unity.
- Xamarin Componenent: In cima alla API JavaScript, permette di utilizzare Wikitude SDK in combinazione con Xamarin.

Anche in questo caso sono disponibili diversi strumenti di supporto allo sviluppo. Wikitude Studio, ad esempio, consente di creare ed organizzare i contenuti virtuali della realtà aumentata. Mentre il Target Manager da la possibilità di salvare in un database le immagini target attraverso un'interfaccia web.

2.2.4 OpenCV

OpenCV[5] (Open Source Computer Vision Library), fu rilasciato ufficialmente nel 1999. Sviluppato inizialmente da Intel Research, come ausilio alle applicazioni CPU-intensive, basate soprattutto su tracking e visione tridimensionale.

Si tratta di una libreria scritta in C e C++ per l'elaborazione di funzionalità di *computer vision* e *image detection*. Perciò, OpenCV non può essere definita una vera e propria libreria di sviluppo per la realtà aumentata. Il suo utilizzo è orientato alla creazione di applicazioni di *visione artificiale* che ,attraverso l'acquisizione, l'elaborazione e l'analisi delle immagini, sono in grado di effettuare un tracking real-time degli oggetti (vedi figura 1.10).

L'interesse verso questa libreria nasce dalla possibilità di integrare le sue funzionalità all'interno della libreria droidAR, descritta nella sezione seguente.

[5] `http://opencv.org/`

La libreria è rilasciata gratuitamente con licenza *BSD Open-Source* ed è disponibile in diverse versioni per molti ambienti di sviluppo.

OpenCV offre un notevole supporto allo sviluppatore, fornendo una documentazione ricca e provvista di tutorial.

2.2.5 DroidAr

DroidAr[6] [Hei13] è un framework di realtà aumentata per Android sviluppato da Simon Heinen, successivamente il progetto è stato portato avanti dalla società BitStar. DroidAR è in grado di combinare tracking GPS e tracking con fiducial marker. È possibile utilizzare i sensori come giroscopio e accelerometro per percepire i movimenti del device e di conseguenza elaborare la relativa posizione degli oggetti virtuali. Questo framework non è in grado di sfruttare il tracking di natural feature. Si può, però, integrare questa funzionalità, come anche il *face-detection*, attraverso l'uso di OpenCV.

Il funzionamento di droidAR, si basa principalmente su tre Thread:

- UI Thread: si occupa dell'interfaccia utente e contiene tutti i thread dedicati alla sua inizializzazione e gestione.
- Render Thread: ha il compito di renderizzare gli oggetti grafici virtuali ed inserirli all'interno della visualizzazione tridimensionale.
- Update Thread: gestisce il posizionamento di tutti gli oggetti grafici in relazione al sistema di riferimento.

Vedremo più nel dettaglio il funzionamento delle componenti della libreria nella sezione dedicata allo sviluppo del progetto.

La libreria è sviluppata in Java, in parte in C++, ed è disponibile solo per piattaforma Android. Il framework è gratuito e open-source, rilasciato con licenza GNU GPL v3. È, però, possibile ricevere una versione commerciale su richiesta. È provvisto di una documentazione scarna, ma dotata di diverse demo.

[6] `http://bitstars.github.io/droidar/`

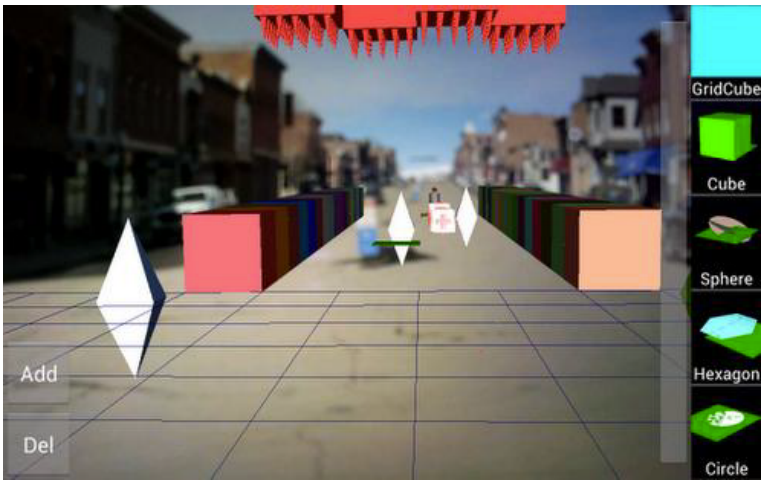

Figura 2.4: Un esempio di applicazione sviluppata con droidA, da: `http://bitstars.github.io/droidar/`

I vantaggi di questa libreria, dovuti al suo codice open-source, sono molteplici. Infatti, lo sviluppatore ha così la possibilità di adattare al meglio la libreria alle funzionalità desiderate. Non di poco conto è la possibilità di studiare il codice ai fini didattici.

Durante i test sono state rilevate, però, alcune problematiche, legate ad alcuni strumenti utilizzati per lo sviluppo. L'ultimo aggiornamento disponibile, infatti, risale al 2013. Ciò ha creato alcune incompatibilità con le nuove API di Google per Android, riguardanti la geo-localizzazione. È stato necessario, ai fini dello sviluppo del progetto, modificare e adattare alcuni moduli della libreria, affinché risultasse compatibile con le ultime versioni di API Android.

2.3 Ricapitolando

La tabella 2.3 mostra uno schema riassuntivo delle caratteristiche analizzate nelle varie librerie di realtà aumentata.

Libreria	Licenza	Ambiente	Marker	GPS	Narural Feature	Face Tracking	Sensori IMU	Doc.
ArToolKit	Free + Commercial	Mobile	Si		Si			Base
Metaio SDK	Free + Commercial (*)	Mobile + Desktop	Si	Si	Si	Si	Si	Buona
Wikitude SDK	Free + Commercial	Mobile	Si	Si	Si		Si	Buona
OpenCV	Free	Mobile + Desktop			Si	Si	Si	Base
DroidAr	Free	Mobile (Android)	Si	Si	Si(**)	Si(**)	Si	Base

(*) Da maggio 2015 il rilascio è bloccato.

(**) Attraverso l'integrazione di OpenCV.

Tabella 2.1: Comparativa Librerie di Realtà Aumentata

Si evince che Metaio SDK risulta essere la libreria attualmente più evoluta. Però, non è stato possibile impiegarla ai fini dello sviluppo. Questo, a causa dell'interruzione del suo rilascio e dei limiti dovuti ad un sorgente chiuso.

Anche Wikitude SDK offre molte funzionalità. Inoltre, la dettagliata documentazione e la ricca community, offrono un ottimo supporto alla programmazione attraverso questo framework. Tuttavia, le limitazioni di utilizzo imposte nella versione trial, costituiscono un fattore penalizzante.

La libreria droidAR, grazie alle sue caratteristiche e alla disponibilità del codice sorgente, è risultata essere la scelta più opportuna ai fini dello sviluppo e dello studio della realtà aumentata.

Capitolo 3

Android

In questo capitolo verranno introdotti gli aspetti principali del sistema operativo Android. Sarà fatta una descrizione generale sulla sua evoluzione. Di seguito, verranno illustrate l'architettura del sistema e i suoi componenti principali. Mostreremo, inoltre, il ciclo di vita di un'activity, uno dei principali elementi della programmazione su piattaforme Android, e le API utilizzate nello sviluppo del progetto.

3.1 Cos'è Android

ANDROID[1] è un sistema operativo nato per i dispositivi mobile, ma successivamente introdotto anche in altri settori: come wearable, netbook, smart-TV, elettrodomestici, ecc... Si tratta di uno dei *mobile OS* più diffusi, presente ormai in in più del 50%[2] di device, come smartphone e tablet. Android nasce nel 2003 da un progetto dello sviluppatore statunitense Andy Rubin, presso la Android Inc. Successivamente Google proseguì lo sviluppo acquisendo Android. Nel 2007 venne distribuito il primo SDK, mentre nel 2008 venne rilasciata la prima versione del sistema operativo Android 1.0.

[1] `https://www.android.com/intl/it_it/`

[2] Fonte: `https://www.netmarketshare.com/`

Il sistema è open-source e basato sulla versione 2.6 del kernel Linux, distribuito gratuitamente con licenza Apache 2.0. Ciò ha sicuramente favorito l'enorme diffusione, agevolando la crescita di una numerosa community di sviluppatori. Le applicazioni sono scritte prevalentemente in Java, un linguaggio di programmazione ad alto livello *object-oriented* molto diffuso.

Il principale tallone d'Achille della piattaforma è sicuramente la sua elevata frammentazione,come mostrato in figura 3.1. Cioè l'utilizzo di differenti versioni installate contemporaneamente su device diversi. La frammentazione delle distribuzioni è certamente una conseguenza della sua natura open-source, per il quale le diverse aziende produttrici di dispositivi sono libere di utilizzare una vasta gamma di hardware per i loro prodotti, magari personalizzando il sistema. Ciò non è d'ausilio per la programmazione di applicazioni da parte degli sviluppatori.

Attualmente la versione più aggiornata di Android è la 5.1(Lollipop), ma la più diffusa resta la 4.4 (KitKat).

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.4%
4.1.x	Jelly Bean	16	11.4%
4.2.x		17	14.5%
4.3		18	4.3%
4.4	KitKat	19	38.9%
5.0	Lollipop	21	15.6%
5.1		22	7.9%

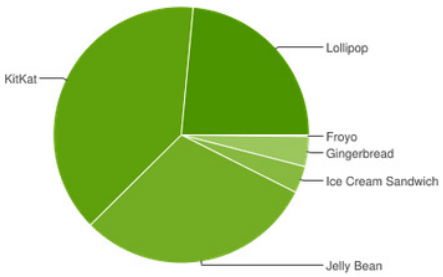

Figura 3.1: Distribuzione delle versioni di Android, da: `http://developer.android.com/about/dashboards/index.html#Platform`

3.2 Architettura

Il Sistema Operativo è strutturato in modo gerarchico, stratificato in *layer,* nel quale i livelli superiori offrono un grado di astrazione più alto ai

livelli inferiori, come descritto in [Car13, Goo]. Ogni layer è costituito da diversi componenti, adibiti alla gestione di specifiche funzionalità. Analizziamo i vari livelli mostrati in figura 3.2.

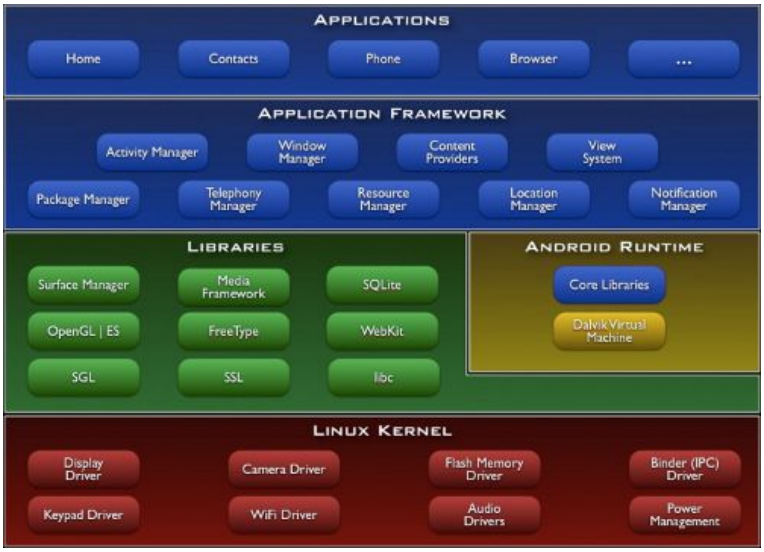

Figura 3.2: Architettura di Android

3.2.1 Linux Kernel

Lo strato più basso dell'architettura è rappresentato dal kernel Linux. Il compito del kernel è quello di fornire gli strumenti necessari per la virtualizzazione dell'hardware sottostante attraverso dei driver dedicati.

In particolare possiamo notare la presenza dei driver per la gestione del display, dell'alimentazione, della connessione WiFi e dei componenti multimediali.

Inoltre è presente un *Builder Driver* (IPC) dedicato alla gestione delle comunicazioni tra processi diversi. L'importanza di questo driver è fondamentale per permettere la comunicazione di componenti diversi in un ambiente dove ogni applicazione è eseguita in un processo.

3.2.2 Librerie

Al di sopra del livello kernel, è presente il vero e proprio *core* del Sistema Operativo Android, costituito da un insieme di librerie native realizzate in C e C++.

Surface Manager

Il Surface Manager ha il compito di gestire le *View*, cioè l'insieme dei componenti che costituiscono l'interfaccia grafica di un'applicazione. Come già specificato, ogni applicazione sarà eseguita in un processo diverso, così il Surface Manager si occupa della corretta visualizzazione delle diverse finestre visualizzate dalle applicazioni, utilizzando un buffer per l'organizzazione delle schermate da visualizzare. Inoltre il Surface Manager ha accesso ai componenti grafici del device, per permettere la visualizzazione sul display di oggetti grafici 2D e 3D di ogni applicazione.

Open GL ES

Si tratta di una versione ridotta di *Open GL*, realizzata per i dispositivi mobile. Open GL è un'importante libreria per la visualizzazione di contenuti in grafica tridimensionale, in grado di sfruttare le funzionalità dell'accelerazione grafica hardware.

SGL

La Scalable Graphics Library (SGL) rappresenta il motore per la grafica bidimensionale, utilizzato sopratutto dal Window Manager e dal Surface Manager. Insieme ad Open GL ES costituisce l'intero motore grafico 2D e 3D del Sistema Operativo Android.

Media Framework

Il Media Framework ha il compito di gestire i vari CODEC per i diversi formati di riproduzione e acquisizione multimediale, realizzando così un'astrazione per l'hardware dedicato alle funzionalità audio e video.

FreeType

FreeType è un motore di rendering di font, ottimizzato per dispositivi mobile, caratterizzato dall'alta portabilità, efficienza e piccole dimensioni.

SQLite

SQLite è una libreria che implementa un Database Management System (DBMS) relazionale, caratterizzato dal fatto di essere molto compatto e di non necessitare di particolari configurazioni. In questo modo è possibile memorizzare le informazioni desiderate in modo permanente.

WebKit

WebKit è un framework fondamentale per l'utilizzo delle funzionalità del Web 2.0. Si tratta di un *browser engine* utilizzato anche in browser di vasta distribuzione come *Chrome* e *Safari*. WebKit si basa sulle tecnologie HTML, CSS, DOM e Javascript.

SSL

Secure Socket Layer (SSL) è una libreria di ormai vasta distribuzione, dedicata alla gestione di connessioni sicure in reti TCP/IP.

Libc

Libc è un'implementazione della libreria standard C *libc,* in questo caso, ottimizzata per i sistemi basati su Linux embedded.

Android Runtime

Costituita dalle Core Libraries e dalla Dalvik Virtual Machine (DVM), è il componente fondamentale per l'esecuzione di codice scritto in Java. Infatti la DVM realizza una compilazione *Just-in-Time,* al fine di tradurre il *bytecode* Java memorizzato nel file *.dex* in codice macchina eseguibile. Dalla versione 4.4 (KitKat) di Android, è stato introdotto Android Run Time (ART) in sostituzione della Dalvik Virtual Machine. ART , a differenza della DVM, crea un eseguibile in formato *.elf* del bytecode al momento della compilazione, che verrà memorizzato e sarà disponibile per ogni esecuzione. Questo tipo di approccio prende il nome di *Ahead-of-Time.*

3.2.3 Application Framework

Si tratta di un insieme di API e componenti per l'esecuzione di funzionalità fondamentali per ogni applicazione Android. L'Application Framework utilizza le librerie dello strato inferiore per l'esecuzione dei task richiesti dalle applicazioni presenti al livello superiore. Di seguito verrà svolta una introduzione delle varie componenti.

Activity Manager

L'Activity Manager si occupa della gestione di ogni *activity.* In seguito verrà illustrato nel dettaglio come l'Activity Manager implementa il ciclo di vita di un'activity. Possiamo immaginare un'activity come la schermata contenente l'interfaccia grafica di una applicazione, in grado di dare all'utente la possibilità di interagire con l'applicazione. Ogni applicazione può essere composta da più activity. L 'Activity Manager organizza tutte le schermate in uno stack, in base all'ordine di visualizzazione sul device.

Package Manager

Il Package Manager svolge un ruolo fondamentale nella gestione dei processi di installazione delle applicazioni nel dispositivo.

Ogni applicazione, infatti, è caratterizzata da un'insieme di informazioni di vario genere, ad esempio: sul layout, sulla versione, sulle varie activity e sulla sicurezza. Tali informazioni sono memorizzate in un file XML indicato come AndroidManifest. Il Package Manager analizza tale file, al fine di permettere una corretta gestione del ciclo di vita dell'applicazione.

Window Manager

Il Window Manager permette di gestire le schermate delle varie applicazioni, gestite dai diversi processi, sul display di un unico dispositivo.

Telephony Manager

Questa componente si occupa della corretta gestione delle funzionalità riguardanti la comunicazione telefonica, come l'avvio di una chiamata telefonica o la verifica dello stato della chiamata.

Content Provider

Il Content Provider funziona come un repository condiviso tra le diverse applicazioni, permettendo così la comunicazione e la condivisione di dati tra le stesse. Questo componente è fondamentale in un Sistema Operativo come Android, al fine di permettere lo scambio dei dati e la cooperazione tra processi.

Resource Manager

Ogni applicazione è costituita da diversi file, ad esempio: file di layout, immagini, file di configurazione, file di properties, ecc... Il compito del Resource Manager è la corretta gestione e ottimizzazione di queste informazioni.

View System

Il View System è responsabile della renderizzazione dei componenti di ogni activity e degli eventi ad essi associati.

Location Manager

Questo componente, di grande importanza nei device mobile, permette di usufruire dei dati riguardanti la localizzazione geografica del dispositivo, alle applicazioni che sfruttano tali informazioni.

Notification Manager

Il Notification Manager ha il compito di inviare delle notifiche al dispositivo, le quali verranno mostrate all'utente mediante icone, messaggi o segnali acustici.

3.3 Ciclo di vita di un'Activity

Come è stato precedentemente illustrato, possiamo considerare un'activity come una finestra contenente la parte visiva di un'applicazione. Questa finestra può trovarsi in diversi stati. In particolare un'activity può essere avviata o meno, può essere visualizzata in primo piano, nel caso in cui l'utente stia interagendo con tale activity, o in background, nel caso in cui fosse avviata ma l'utente stesse interagendo con un'altra applicazione. Possiamo delineare i vari stati in base agli eventi definiti dalla classe Activity, e mostrati nell'illustrazione 3.3.

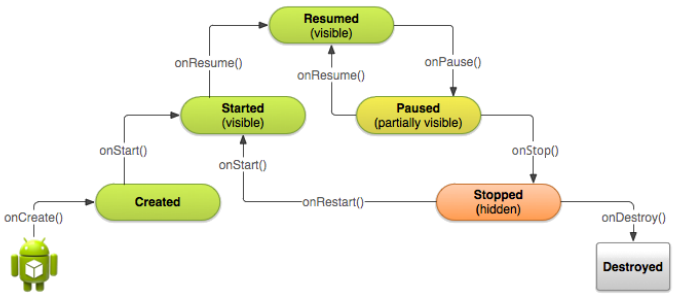

Figura 3.3: Ciclo di vita di un'activity Android, da: `http://developer.android.com/training/basics/activity-lifecycle/starting.html`

Analizziamo brevemente i principali eventi:

- onCreate(Bundle *savedInstanceState*): questo evento viene invocato quando l'activity è lanciata per la prima volta, o dopo essere stata chiusa definitivamente. Si occupa di inizializzare l'activity, di inserire il layout grafico, creare i riferimenti ai vari elementi presenti sulla schermata e avviare i servizi. Il parametro *savedInstanceState* di tipo Bundle, contiene le informazioni necessarie a riportare l'activity allo stato dell'istanza precedente.

- onStart(): viene chiamato non appena l'activity è stata creata e si occupa di registrare eventuali BroadcastReceiver e di rendere visibile la schermata; da questo momento in poi l'activity è in status *Visible*.

- onResume(): l'evento in questione identifica lo stato in cui l'activity è pronta per interagire con l'utente.

- onPause(): viene chiamato quando l'applicazione apre una nuova Activity, lo schermo viene spento o l'applicazione perde il focus e si occupa di liberare le risorse non necessarie quando l'applicazione viene messa in pausa; prima della sua chiamata viene eseguito il metodo onSaveInstanceState() che si occupa di salvare lo stato attuale dell'activity; in seguito a questo metodo l'activity risulta in status *Paused*.

- onStop(): l'activity non è visibile all'utente e non è possibile interagire con essa, ma è ancora presente in memoria. Si occupa di liberare tutte le risorse non più necessarie, de-registrare i BroacastReceiver; in seguito a questo metodo l'activity risulterà *Stopped* e potrà essere riavviata tramite onRestart(), oppure distrutta chiamando onDestroy()

- onDestroy(): l'evento identifica lo stato in cui l'activity viene distrutta e la memoria da essa occupata viene de-allocata. Questo evento può essere invocato dall'utente, ma anche dal Sistema Operativo nel caso in cui l'applicazione fosse in background e la memoria necessaria non sufficiente per altre applicazioni.

3.4 API Android utili alla realizzazione del progetto

Le Application Programming Interface (API) sono librerie messe a disposizione degli sviluppatori come supporto alla programmazione. Le API di Google a supporto della programmazione Android sono numerose e molto efficienti. Tanto da permettere la creazioneo di quasi ogni tipo di applicazione. Infatti, tali strumenti semplificano molto la programmazione e offrono un potenziale enorme per lo sviluppo di applicativi anche molto complessi. Di seguito verranno descritte le API Android utilizzate all'interno del progetto.

3.4.1 Google Maps API for Android

Con l'API di Google Maps per Android [Goo16] è possibile aggiungere le mappe basate su Google Maps all'interno di un'applicazione Android. L'API gestisce autonomamente l'accesso ai server del servizio Maps, il download dei dati relativi alle mappe e alla posizione, la visualizzazione della mappa e l'interazione con l'utente.

Google Maps permette all'utente, tramite l'inserimento di un indirizzo, di individuare la località richiesta all'interno di una mappa.

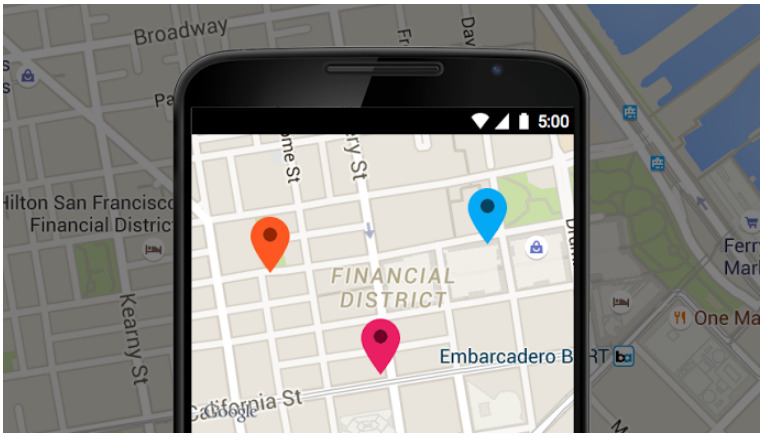

Figura 3.4: Google Maps per Android, da: `developers.google.com`

Lo sviluppatore ha a disposizione diversi strumenti per semplificare l'interazione dell'utente con la mappa presente nella sua applicazione. Ad esempio è possibile inserire nella mappa diversi marker, caratterizzati da una localizzazione, un'icona e una finestra per le informazioni nel caso l'utente selezioni il marker. Inoltre, è possibile sovrapporre alla mappa dei poligoni per delimitare un'area, oppure delle linee per indicare un percorso. Le mappe messe a disposizione da Google sono di tre tipi: mappa standard, immagini satellitari o ibrida, cioè un mix di entrambi. Lo sviluppatore ha la possibilità di scegliere quale delle tre inserire nella sua applicazione o lasciare all'utente la scelta, inserendo l'apposito selettore.

Per poter utilizzare le API di Google Maps, e le API descritte di seguito, è necessario ottenere da Google una *API Key*[3], utilizzata per identificare l'applicazione e lo sviluppatore, oltre a validare le richieste effettuate.

Le API di Google Maps per Android sono disponibili in una versione gratuita senza limiti nelle richieste di localizzazione.

3.4.2 Google Places API for Android

Google Places API [Goo16-3] è un servizio messo a disposizione da Google per ricevere informazioni riguardo a località geografiche, edifici e punti di interesse, denominati Places. Ogni Place è caratterizzato da: il nome del luogo e il suo indirizzo, collocazione geografica, ID univoco, il numero di telefono, tipo di posto, URL del sito web, e altro ancora. Tali informazioni possono essere sfruttate attraverso una mappa, arricchendo la quantità di dati messa a disposizione dell'utente.

Gli strumenti offerti da Places sono:

- PlacePicker UI Widget: è una finestra di dialogo che consente all'utente di scegliere un luogo utilizzando una mappa interattiva.

[3]La guida per ottenere una API Key è descritta alla pagina: `https://developers.google.com/maps/documentation/android-api/signup`

- Autocomplete UI Widget: un widget che offre la possibilità di inserire il nome di un punto di interesse, un indirizzo o una località. Il widget semplifica la ricerca attraverso l'auto-completamento del testo inserito, effettuando, ad ogni modifica, delle richieste ai server dedicati.

- GeoDataApi: fornisce l'accesso al database di Google per recuperare tutte le informazioni necessarie su un Place.

- PlaceDetectionApi: fornisce un rapido accesso alle informazioni riguardanti la posizione geografica del device.

Google Places API dovrà essere interrogato attraverso delle richieste HTTP. In ogni richiesta è necessario specificare la posizione geografica desiderata attraverso la latitudine e la longitudine.

Google Places API è gratuito, ma impone un limite di 1.000 richieste in 24 ore. È possibile aumentare il limite delle richieste fornendo a Google i dati di una Carta di Credito, utilizzati per l'identificazione.

Le funzionalità di Places API sono state utilizzate nel progetto per offrire all'utente la possibilità di impostare la destinazione desiderata, inserendo l'indirizzo o il nome della locazione specifica.

Esempio di richiesta HTTP per Places API

Di seguito viene mostrata una porzione di codice come esempio di richiesta al server di Google Places API. Si noti che all'interno della richiesta è possibile specificare il formato della risposta tra JSON e XML.

```
1  ...
2
3  /*  Definizione dei parametri per la richiesta  */
4  GooglePlacesAutocompleteAdapter adapter;
5  private static final String PLACES_API_BASE = "https://maps.googleapis.com/\
       maps/api/place";
6  private static final String TYPE_AUTOCOMPLETE = "/autocomplete";
7  private static final String OUT_JSON = "/json";
8  private static final String API_KEY = "KEY_XXX";
9  private String country = "it";
10
```

```
11  ...
12
13  StringBuilder sb = new StringBuilder(PLACES_API_BASE + TYPE_AUTOCOMPLETE + \
        OUT_JSON);
14  sb.append("?key=" + API_KEY);
15  sb.append("&components=country:" + this.country);
16  sb.append("&input=" + URLEncoder.encode(location, "utf8"));
17  /* location è la stringa dell'indirizzo di interesse */
18
19  /*  Creazione dell'indirizzo URL per effettuare la richiesta  */
20  URL url = new URL(sb.toString());
21  conn = (HttpURLConnection) url.openConnection();
22  InputStreamReader in = new InputStreamReader(conn.getInputStream());
23
24  ...
```

L'URL così formato avrà la forma seguente:

```
https://maps.googleapis.com/maps/api/place/autocomplete/outputFormat?parameters
```

Esempio di risposta di Places API

Di seguito è riportato un esempio di risposta in linguaggio JSON, presente sul sito `https://developers.google.com/places/web-service/autocomplete`. È possibile ricevere la risposta in formato XML, specificando il parametro all'interno della richiesta URL.


```
{
  "status": "OK",
  "predictions" : [
      {
         "description" : "Paris, France",
         "id" : "691b237b0322f28988f3ce03e321ff72a12167fd",
         "matched_substrings" : [
            {
               "length" : 5,
               "offset" : 0
            }
         ],
         "place_id" : "ChIJD7fiBh9u5kcRYJSMaMOCCwQ"
      "reference": "\
          CjQlAAAA_KB6EEceSTfkteSSF6U0pvumHCoLUboRcDlAH05N1pZJLmOQbYmboEi0SwXBSoI2EhAhj249tFDCVh4
          -PXZkPK8GhTBmp_6_lWljaf1joVs1SH2ttB_tw",
         "terms" : [
            {
               "offset" : 0,
```

```
            "value" : "Paris"
         },
         {
            "offset" : 7,
            "value" : "France"
         }
      ],
      "types" : [
         "locality", "political",
         "geocode"
      ]
   },
   {
      "description" : "Paris Avenue, Earlwood, New South Wales, Australia\
          ",
      "id" : "359a75f8beff14b1c94f3d42c2aabfac2afbabad",
      "matched_substrings" : [
         {
            "length" : 5,
            "offset" : 0
         }
      ],
      "place_id" : "ChIJrU3KAHG6EmsR5Uwfrk7azrI",
   "reference": "CkQ2AAAARbzLE-\
       tsSQPgwv8JKBaVtbjY48kInQo9tny0k07FOYb3Z_z_yDTFhQB_Ehpu-\
       IKhvj8Msdb1rJlX7xMr9kfOVRIQVuL4tOtx9L7U8pC0Zx5bLBoUTFbw9R2lTn_EuBayhDvugt8T0Oo\
       ",
      "terms" : [
         {
            "offset" : 0,
            "value" : "Paris Avenue"
      },
         {
            "offset" : 14,
            "value" : "Earlwood"
         },
         {
            "offset" : 24,
            "value" : "New South Wales"
         },
         {
            "offset" : 41,
            "value" : "Australia"
         }
      ],
      "types" : [ "route", "geocode" ]
   },
```

...

3.4.3 Google Directions API

Direction API [Goo16-1] è un servizio che fornisce dati riguardo ad un percorso tra due o più località geografiche, fornendo informazioni sulla rotta da seguire, attraverso richieste HTTP. Si può richiedere il percorso specifico per diversi mezzi di trasporto, come: mezzi pubblici, automobile, a piedi o bicicletta. Le posizioni di partenza, destinazione e dei *waypoint*[4] possono essere specificate attraverso l'uso delle coordinate geografiche, o con il testo che identifica l'indirizzo. La risposta alla richiesta può contenere un percorso formato da diversi sotto-percorsi, delimitati dai waypoint.

Questo strumento è stato progettato per il calcolo di una rotta tra indirizzi statici, conosciuti in anticipo. L'utilizzo per il calcolo dinamico dei percorsi, caratterizzato dalla risposta in tempo reale alle modifiche degli estremi della rotta, è sconsigliato. Poiché l'elaborazione della rotta, attraverso questi metodi, è oneroso dal punto di vista del costo computazionale.

Google Direction API definisce due tipi di limiti alle richieste:

Standard Usage Limit Per utenti delle API standard:

1. 2.500 richieste gratuite al giorno.
2. Fino ad un massimo di 23 waypoint consentiti in ogni richiesta contenente una API Key, o fino a 8 waypoint quando non è fornita la API Key, o fino a 8 waypoint quando si utilizza il servizio Indicazioni in Google Maps API JavaScript.
3. I waypoint non sono disponibili per le indicazioni di trasporto pubblico.
4. Un massimo di 10 richieste al secondo.

[4]Sono punti di transito, distribuiti lungo la rotta compresa tra i due estremi. Questi ultimi non considerati dei waypoint.

Premium Usage Limit Per i clienti *Google Maps APIs Premium Plan*:

1. 100.000 richieste in 24 ore.
2. Massimo 23 waypoint per ogni richiesta.
3. I waypoint non sono disponibili per le indicazioni di trasporto pubblico.
4. Massimo 50 richieste al secondo.

Esempio di richiesta HTTP per Directions API

Il codice JAVA seguente mostra un esempio di richiesta a Direction API, per ottenere un percorso in formato JSON. È possibile richiedere i dati formattati in linguaggio XML.

```
1  ...
2
3  /*  Definizione dei parametri per la richiesta  */
4  params.add(new BasicNameValuePair("origin",startPoint.latitude + "," + \
       startPoint.longitude));
5  params.add(new BasicNameValuePair("destination",destinationPoint.latitude +\
       "," + destinationPoint.longitude));
6  params.add(new BasicNameValuePair("sensor", "false"));
7  params.add(new BasicNameValuePair("alternatives",
8          String.valueOf(alternativeReq)));
9  params.add(new BasicNameValuePair("language", language));
10 params.add(new BasicNameValuePair("key", myKey));
11
12 ...
13
14 String input = URLEncodedUtils.format(params, "utf-8");
15 URL requestUrl = new URL("https://maps.googleapis.com/maps/api/directions/\
       json?" + input    );
16 HttpURLConnection connection = (HttpURLConnection)requestUrl.openConnection\
       ();
17 connection.setRequestMethod("GET");
18 connection.connect();
19
20 ...
```

Di seguito viene indicato un esempio generico dell'URL di richiesta a Direction API:

```
https://maps.googleapis.com/maps/api/directions/outputFormat?parameters
```

Esempio di risposta di Direction API

Le risposte sono inviate nel formato specificato dal parametro *outputFormat* all'interno della richiesta URL. Di seguito è mostrato un esempio di risposta in formato JSON, dal sito `https://developers.google.com/maps/documentation/directions/`.


```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJyYfhZ79ZwokRMtXcL6CYxkA",
         "types" : [ "premise" ]
      },
      {
         "geocoder_status" : "OK",
         "partial_match" : true,
         "place_id" : "ChIJ8YWMWnz4wokRCOVf1CcJCbY",
         "types" : [ "street_address" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 40.8171321,
               "lng" : -73.99449150000001
            },
            "southwest" : {
               "lat" : 40.7416627,
               "lng" : -74.0728354
            }
         },
         "copyrights" : "Map data ©2015 Google",
         "legs" : [
            {
               "distance" : {
                  "text" : "9.7 mi",
                  "value" : 15653
               },
               "duration" : {
                  "text" : "25 mins",
```

```
      "value" : 1480
   },
   "end_address" : "1 MetLife Stadium Dr, East Rutherford, NJ \
       07073, USA",
   "end_location" : {
      "lat" : 40.814505,
      "lng" : -74.07272910000002
   },
   "start_address" : "75 Ninth Ave, New York, NY 10011, USA",
   "start_location" : {
      "lat" : 40.7428759,
      "lng" : -74.00584719999999
   },

         ...
```


3.4.4 Google Elevation API

Elevation API [Goo16-2] fornisce informazioni sull'altitudine per qualunque posizione geografica sulla superficie terrestre, compresi i fondali marini. Nel caso in cui una valutazione precisa dell'elevazione di un determinato luogo non fosse disponibile, viene effettuata una stima approssimata basandosi sulle quattro misurazioni più vicine. Attraverso le API di Google Maps Elevation, è possibile arricchire ulteriormente i dati riguardanti un percorso o un punto d'interesse, utilizzando le informazioni sull'elevazione. I dati sull'elevazione vengono richiesti attraverso un'interfaccia HTTP. All'interno di una richiesta possono essere specificati uno o più punti geografici (che identificano un percorso), utilizzando latitudine e longitudine.

Anche per Elevation API sono previsti due diversi limiti di utilizzo:

Standard Usage Limit Per utenti delle API standard:

1. 2.500 richieste gratuite al giorno.
2. Fino a 512 locazioni per ogni richiesta.
3. Un massimo di 10 richieste al secondo.

Premium Usage Limit Per i clienti *Google Maps APIs Premium Plan*:

1. 100.000 richieste in 24 ore.
2. Fino a 512 locazioni per ogni richiesta.
3. Massimo 50 richieste al secondo.

Esempio di richiesta HTTP per Elevation API

Il codice che segue mostra un esempio di richiesta di altitudine a Elevation API:

```
1
2  ...
3
4  HttpClient httpClient = new DefaultHttpClient();
5  HttpContext localContext = new BasicHttpContext();
6  String url = "http://maps.googleapis.com/maps/api/elevation/" + "xml?\
       locations=" + String.valueOf(latitude) + "," + String.valueOf(longitude\
       ) + "&sensor=true"  ;
7  HttpGet httpGet = new HttpGet(url);
8  try {
9                  HttpResponse response = httpClient.execute(httpGet, \
                       localContext);
10                 HttpEntity entity = response.getEntity();
11                 if (entity != null) {
12                         ...
13         }
14         } catch (ClientProtocolException e) {}
15
16 ...
```

L'URL della richiesta a Elevation API è qui mostrata come esempio:

```
https://maps.googleapis.com/maps/api/elevation/xml?locations
    =39.7391536,-104.9847034|36.455556,-116.866667&key=YOUR_API_KEY
```

Esempio di risposta di Elevation API

Il formato della risposta potrà essere XML o JSON, specificandolo preventivamente nella richiesta. La risposta in formato XML della richiesta mostrata precedentemente sarà:

```
{
   "results" : [
      {
         "elevation" : 1608.637939453125,
         "location" : {
            "lat" : 39.73915360,
            "lng" : -104.98470340
         },
         "resolution" : 4.771975994110107
      },
      {
         "elevation" : -50.78903579711914,
         "location" : {
            "lat" : 36.4555560,
            "lng" : -116.8666670
         },
         "resolution" : 19.08790397644043
      }
   ],
   "status" : "OK"
}
```

Capitolo 4

Progettazione e sviluppo di ARMH (Augmented Reality Motorbike Helmet)

In questo capitolo verranno mostrati gli aspetti fondamentali della progettazione e dell'implementazione del sistema ARMH.

Innanzitutto saranno analizzati i requisiti necessari allo sviluppo, compreso ambiente di sviluppo, hardware necessario per i dispositivi e componenti software in grado di offrire determinate funzionalità. Successivamente sarà illustrata la struttura del progetto, la comunicazione tra le diverse componenti e il modo in cui le informazioni vengono elaborate e visualizzate in realtà aumentata. Infine verranno descritti possibili miglioramenti e implementazioni future.

4.1 Analisi dei requisiti

ARMH (Augmented Reality Motorbike Helmet) è un progetto sperimentale nato in ambito di ricerca presso la Raleri s.r.l., azienda attiva nel settore ottico, che da anni studia nuove tecnologie in campo ottico-sportivo. L'obiettivo è quello di sviluppare un sistema di navigazione basato sulla realtà

aumentata, che possa essere sfruttato all'interno di un casco motociclistico. Le informazioni saranno proiettate attraverso un sistema HUD (Head-Up-Display) e comprendono: tracciato del percorso tra due punti, velocità di marcia, indicazioni stradali, un grafico dell'altitudine del percorso, una mappa e le notifiche avviate dallo smatphone collegato.

L'insieme delle informazioni dovrà essere presentato all'utilizzatore in modo poco invasivo, al fine di non compromettere la sicurezza alla guida. Sarà, inoltre, necessario implementare un canale di comunicazione con un secondo device (smartphone), al quale verrà affidato il compito di interagire con l'utente e di procurare le informazioni.

Al fine di garantire un costante scambio di dati con lo smartphone, il device installato nel casco, dovrà prevedere componenti hardware in grado di sfruttare la tecnologia Bluetooth o WiFiDirect. Oltre a ciò, dovranno essere presenti dei sensori inerziali IMU (Internal Measurement Unit), come accelerometro e giroscopio, che saranno utilizzati per intercettare l'orientamento e i movimenti del casco. Una GPU in grado di fornire accelerazione hardware, seppur non necessaria, sarà utile nel rendering grafico di oggetti 3D.

Per rendere possibile la progettazione e l'implementazione, è stato necessario *emulare* il dispositivo installato nel casco utilizzando un comune tablet. Al fine di tralasciare alcune problematiche non inerenti la materia di studio. Tale scelta non comporta rilevanti modifiche concettuali per la realizzazione dell'intero sistema.

Grazie alla sua grande versatilità e alla predisposizione in ambito mobile, si è scelto di utilizzare Android come sistema operativo per l'intero progetto. Inoltre l'opportunità di modificare il codice sorgente, offre allo sviluppatore la possibilità di adattare al meglio il sistema operativo alle funzionalità richieste. L'ambiente di sviluppo utilizzato è Android Studio nella versione 1.4 , un' IDE basato su *IntelliJ IDEA*, fornito da Google per lo sviluppo su piattaforma Android. Android Studio è disponibile gratuitamente per Windows, Mac OS X e Linux, rilasciato con licenza Apache 2.0. La versione del sistema operativo utilizzato per la componente che sfrutterà la realtà aumentata è

Android 5.1 (Lollipop). La seconda applicazione potrà essere installata su dispositivi con versioni di Android pari o superiori alla 4.1.

L'elaborazione e la gestione degli oggetti virtuali 3D sarà affidata ad una libreria per la realtà aumentata, in grado di posizionarli nello spazio grazie alle coordinate geografiche. A tal fine è stata scelta la libreria gratuita e open-source droidAR, che è caratterizzata da una grande potenzialità e dall'assenza di limiti di utilizzo. Seppur corredata di una scarna documentazione, la possibilità di studiarne il codice e il funzionamento l'ha resa utile ai fini didattici.

Il sistema sviluppato sarà in grado di fornire le seguenti funzionalità:

- Calcolo del percorso tra due punti, eventualmente offrire la possibilità di scegliere il percorso preferito.
- Calcolo dell'altitudine lungo il percorso.
- Visualizzazione delle informazioni in realtà aumentata.
- Visualizzazione della rotta attraverso una mappa e un tracciato in AR.
- Navigazione turn-by-turn con indicazioni stradali.
- Connessione Bluetooth tramite moduli integrati nei dispositivi.
- Interazione con l'utente mediante il rilevamento dei movimenti e l'applicazione dedicata, installata su uno smartphone.
- Ricezione delle notifiche avviate dallo smartphone collegato.

4.2 Struttura

Il progetto è composto da due applicazioni Android:

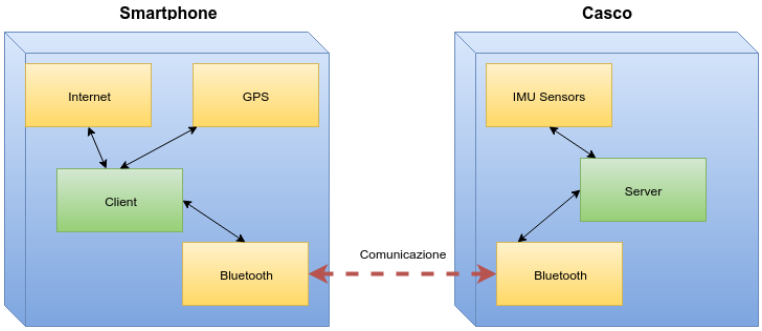

Figura 4.1: Struttura del progetto

- Un client installato su uno smartphone, il quale ha il compito di offrire la possibilità all'utente di pianificare un percorso e visualizzarlo su una mappa. Successivamente, i dati relativi a tale percorso, saranno inviati al casco attraverso il canale di comunicazione prestabilito. In questo caso è stata scelta una comunicazione Bluetooth. Oltre al percorso, il client, dovrà stabilire e fornire la corretta posizione geografica del device.

- Un server installato nel dispositivo all'interno del casco. Il suo compito è quello di ricevere le informazioni del client, elaborarle e visualizzarle. Inoltre, analizza i movimenti del casco, posizionando di conseguenza tutti gli oggetti virtuali nel campo visivo.

4.2.1 Struttura del client

La classe principale dell'applicazione client è costituita dall'activity **MapsActivity.java**, la quale ha il compito di inizializzare la mappa, avviare i vai Thread e Service, gestire gli elementi da visualizzare all'interno della mappa e interagire con l'utente. Quando viene avviato il metodo onCreate(), l'activity si occupa di impostare l'interfaccia grafica dell'applicazione, compreso il fragment adoperato per la visualizzazione della mappa. Successivamente saranno registrati il LocationListener per la geo-localizzazione e il client per le richieste a Google Place e l'auto-completamento dell'indirizzo.

La connessione mediante interfaccia Bluetooth è stabilita dalla classe **BluetoothChatService.java**, la quale avvia il Thread *ConnectThread*, in grado di essere eseguito in modo parallelo dal resto dell'applicazione. Il Thread ha il compito, una volta avviato, di instaurare una connessione stabile e duratura con il server, al fine di assicurare un pronto scambio di dati. Al termine della creazione della connessione, il Thread viene stoppato. Lo stato della connessione sarà fornito dal Thread *ConnectedThread,* che sarà in esecuzione durante tutta la sua durata, e lo scambio di messaggi sarà gestito dalla classe **BluetoothMessenger.java**.

Il calcolo del percorso e dell'altitudine relativa ai punti di transito sono affidati alle rispettive classi **Routing.java** e **Elevation.java**. Queste classi utilizzano le API di Google per ottenere i dati richiesti. Grazie alle classi DirectionUtils.java e AltitudeUtils.java, sono in grado di analizzare i dati ricevuti, estrapolarli da un file JSON e inserirli in una lista.

La decodifica e la formattazione di file in formato JSON sono effettuati attraverso le classi **JsonFileReader.java** e **JsonFileWriter.java.**

È inoltre presente un Service, chiamato **NLService.java**, in grado di intercettare le notifiche testuali avviate dallo smartphone, come: chiamate in arrivo, chiamate perse, messaggi testuali, stato della connessione GPS e altro. Tali notifiche saranno elaborate e inviate all'applicazione server per essere comunicate all'utente.

4.2.2 Struttura del server

Il core dell'applicazione server è rappresentato dalla libreria droidAR, che si occupa della visualizzazione degli elementi virtuali in realtà aumentata, del loro corretto posizionamento nello spazio, dell'elaborazione dei dati ottenuti dai sensori IMU e dell setup dell'interfaccia utente. In particolare, l'activity **ARActivityPlusMaps.java** crea e gestisce una mappa contenente il percorso stabilito, ed è il contenitore degli elementi di realtà aumentata. Questa activity viene avviata con una classe di setup, **NewSetup.java**, che estende la classe **MyDefaultARSetup.java**, dalla quale ne eredita le pro-

prietà. Quest'ultima è la classe in grado di impostare, modificare e eliminare elementi dell'interfaccia utente e della realtà aumentata.

Oltre alla mappa, l'interfaccia utente è provvista di un grafico dinamico, creato mediante la classe **plotDynamic.java**. Il grafico mostra i dati relativi all'altitudine del percorso e al suo avanzamento. Grafico e mappa possono essere alternati nella visualizzazione attraverso i movimenti del dispositivo, intercettati dalla classe **TiltSensor.java,** la quale implementa un *SensorEventListener*.

La comunicazione Bluetooth è stabilita dalla classe **BluetoothChatService.java**, che avvia il Thread *AcceptThread*, in attesa di una connessione in ingresso da parte del device accoppiato. Anche in questo caso, a connessione avviata, il Thread verrà bloccato e il controllo dello stato della connessione passerà al Thread *ConnectedThread.* La ricezione dei messaggi avviene attraverso la classe **BluetoothReceiver.java**, che si occupa dell'elaborazione e identificazione dei frame ricevuti.

La figura 4.2 mostra il diagramma delle classi di tale applicazione, illustrandone i componenti fondamentali e le loro relazioni.

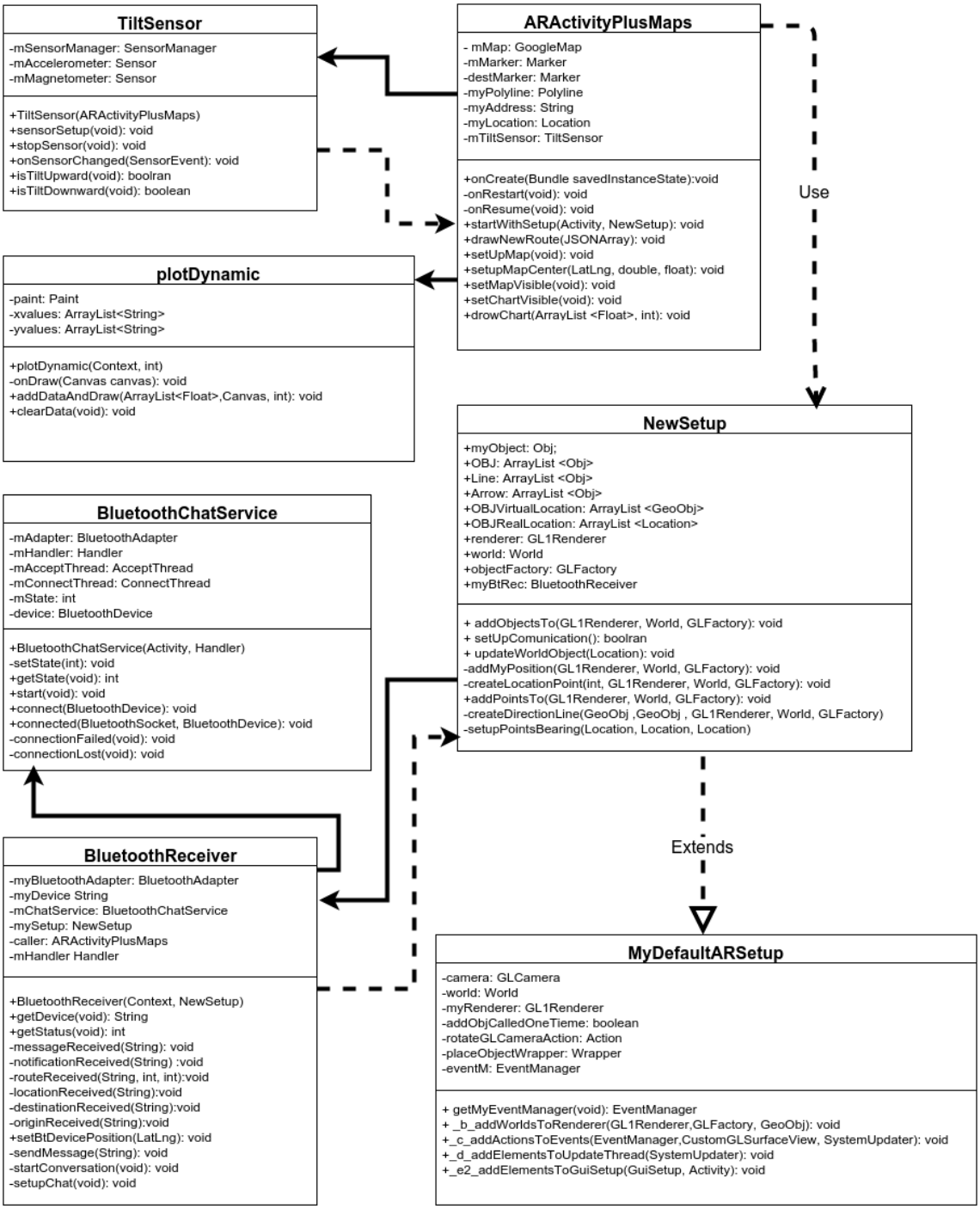

Figura 4.2: Diagramma delle classi del server

4.3 Pianificazione del percorso

In questa sezione vedremo come l'applicazione client utilizzi il sistema di posizionamento geografico e le API di Google Maps, per ottenere diverse possibilità di percorso tra la posizione attuale e una destinazione scelta dall'utente. I vari percorsi vengono mostrati all'utente, indicando i dati relativi alla lunghezza del tracciato, tempo presunto di percorrenza e altitudine.

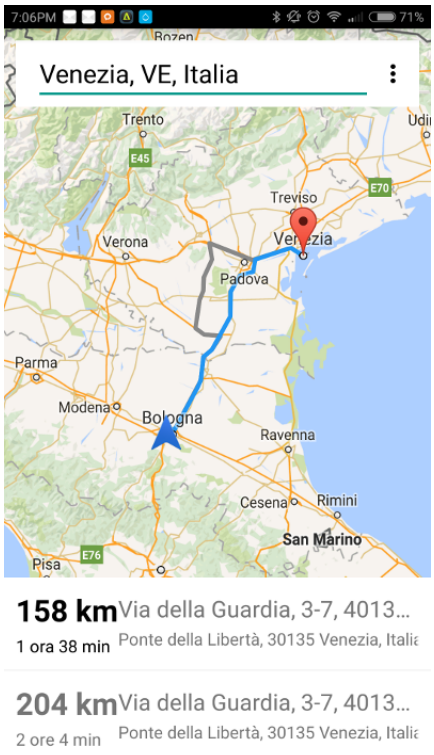

Figura 4.3: Client per la pianificazione del percorso

4.3.1 Geo-localizzazione

La Geo-localizzazione consiste nel rilevare la posizione geografica di un dispositivo, esprimendola nelle coordinate denominate latitudine e longitudine. Questi valori sono dei numeri frazionari che indicano un punto preciso sulla superficie terrestre, ipotizzando che questa sia piatta, ossia bidimensionale.

Esistono diversi metodi per la geo-localizzazione dei dispositivi mobile, qui vedremo i principali:

GPS: è il sistema più affidabile e utilizzato in ambito di navigazione, basato su informazioni provenienti da una rete di satelliti. Molto accurato e comunque disponibile sulla maggior parte dei dispositivi Android, senza dover usare una connessione dati.

Network-Based: sfrutta i dati provenienti dalle reti mobile locali (WiFi o GSM) per ottenere informazioni geografiche. Non è particolarmente accurata, ma è veloce e disponibile su praticamente tutti i dispositivi, a patto di avere una connessione dati attiva.

La localizzazione del dispositivo è stabilita attraverso il servizio *FusedLocationApi*. Questo fa uso di entrambi i metodi di posizionamento geografico appena descritti, per ottenere le coordinate geografiche del terminale secondo i parametri stabiliti.

Nel seguente esempio, viene mostrato il codice Java relativo all'impostazione dei parametri per la richiesta di posizionamento:

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      ...
4
5      /***    Start LocationListener      ***/
6      client = new GoogleApiClient.Builder(this.\
           getApplicationContext())
7          .addApi(LocationServices.API)
8          .addConnectionCallbacks(this)
9          .addOnConnectionFailedListener(this)
10         .build();
11     mLocationRequest = LocationRequest.create()
12         .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
13         .setInterval(1000)
14         .setFastestInterval(500)
15         .setSmallestDisplacement(1f);
16
```

```
17      ...
18  }
19
20   @Override
21  public void onConnected(Bundle bundle) {
22      LocationServices.FusedLocationApi.requestLocationUpdates(
23          client, mLocationRequest, this);
24  }
25
26   @Override
27  public void onLocationChanged(Location location) {
28      this.myLocation=location;
29      setupMapCenter(new LatLng(
30              location.getLatitude(),
31              location.getLongitude()
32          ),
33          location.getAccuracy(),
34          location.getBearing());
35
36      ...
37  }
38  ...
```

setPriority() : indica la priorità di aggiornamento della posizione, in questo caso settata con il parametro PRIORITY_HIGH_ACCURACY, cioè richiede la massima accuratezza possibile per la localizzazione.

setInterval() : imposta l'intervallo di tempo degli aggiornamenti in ordine di millisecondi. La richiesta mostrata è inizializzata con un intervallo di un secondo.

setFastestInterval() : indica il minimo intervallo di tempo nel quale verrà inviato l'aggiornamento della posizione. In questo modo, il sistema fornirà al massimo un aggiornamento ogni 500 millisecondi.

setSmallestDisplacement() : definisce la minima distanza in metri tra un aggiornamento e il successivo. Se questo requisito non è soddisfatto, il sistema non invierà nessun aggiornamento della posizione.

Per ottenere i dati richiesti sulla posizione geografica del dispositivo, è necessario inserire i seguenti permessi all'interno dello specifico file AndroidManifest.xml:

```
<!-- Permesso per le richieste di localizzazione GPS -->
<uses-permission andorid:name="android.permission.ACCESS_FINE_LOCATION" />
<!-- Permesso per le richieste di localizzazione Network-Based -->
<uses-permission andorid:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Sfruttare la geo-localizzazione in Android vuol dire, generalmente, gestire un dialogo tra le due componenti:

LocationManager: è un servizio di sistema, rintracciabile mediante Context, che svolge il ruolo di gestore unico dei sistemi di localizzazione.

LocationListener: è il ruolo svolto dall'elemento della nostra applicazione, che desidera ricevere informazioni sulla posizione del dispositivo. Tipicamente tale ruolo viene svolto dall'activity che pertanto dovrà implementare l'interfaccia LocationListener.

4.3.2 Impostazione della destinazione

L'applicazione offre la possibilità all'utente di impostare una destinazione in due modi differenti:

1. Attraverso la selezione di un punto sulla mappa.
2. Mediante l'utilizzo di Places API e dello strumento Autocomplete UI Widge. Si ha, così, la possibilità di inserire l'indirizzo della località desiderata o il nome di un punto di interesse.

Autocomplete UI Widge effettua una richiesta di informazioni ai server di Google Places ad ogni modifica del testo, per ottenere una serie di indi-

rizzi compatibili con il testo inserito. Non è necessario immettere l'indirizzo completo, poiché il servizio così implementato offre un metodo di autocompletamento del testo. L'utente potrà selezionare la destinazione desiderata, tra quelle suggerite dall'applicazione.

In entrambi i casi la destinazione sarà caratterizzata dalle coordinate geografiche al quale verrà assegnato un *marker* che identifica la posizione scelta.

4.3.3 Impostazione del percorso

Un percorso è costituito da una serie di punti geografici, costituiti da latitudine e longitudine, con il quale è possibile costruire un tracciato. Più i punti sono vicini tra loro, maggiore sarà la precisione del tracciato.

Ogni tracciato è visualizzato attraverso lo strumento *Polyline* di Google Maps, mentre un widget mostra una lista dei possibili percorsi selezionabili.

Nello specifico, mediante le informazioni riguardanti la posizione del device e le coordinate della destinazione scelta, si potrà procedere con la richiesta a Direction API di uno o più percorsi tra le due tappe. La richiesta è impostata per ottenere i dati riguardo ad un tragitto in automobile. Nel caso fossero disponibili diversi percorsi, questi saranno organizzati in base ad una priorità ottenuta dal tempo di percorrenza e dalla lunghezza del tragitto.

Ricevuta la risposta contenente le informazioni richieste, verrà elaborata, al fine di estrarre tutti i punti di transito necessari per la costruzione del tracciato sulla mappa e comunicare all'utente le possibili soluzioni (vedi figura 4.3).

Selezionando tra le alternative il percorso desiderato, verrà mostrato un grafico dell'altitudine dell'intero tracciato. I valori dell'altitudine sono ottenuti effettuando una richiesta a Elevation API. All'interno della richiesta verranno inseriti tutti i punti geografici del percorso. Se il numero dei punti è maggiore di 512 , stabilito dal limite di utilizzo standard, l'insieme dei punti sarà suddiviso e verranno effettuate diverse richieste. Successivamente i dati

ottenuti saranno ricombinati per essere aggiunti al percorso memorizzato in formato JSON.

Il grafico dell'elevazione è realizzato mediante il servizio Google Charts[1], al quale viene effettuata una richiesta HTML contenente i valori da visualizzare, cioè l'altitudine. Google Chart restituisce un'immagine del grafico elaborato, che l'utente potrà visualizzare all'interno di un widget.

4.4 Comunicazione

Lo standard Bluetooth fornisce un metodo economico e sicuro per lo scambio di dati tra dispositivi diversi, attraverso una frequenza radio a corto raggio. Il campo d'azione di questa tecnologia è ristretto ad una decina di metri[2], comunque sufficiente all'utilizzo effettuato all'interno del progetto. Inoltre tutti i moderni dispositivi mobile integrano tale connettività.

La connessione tra le due componenti del progetto è stabilita attraverso il protocollo Bluetooth Serial Port Profile (SPP). Il server , ad ogni avvio, si mette in attesa di una richiesta di connessione del client associato ad un identificativo (UUID). Mentre il client da la possibilità all'utente di ricercare e selezionare il device accoppiato.

La comunicazione vera e propria dei messaggi avviane attraverso le due classi: **BluetoothMessenger.java** e **BluetoothReceiver.java**, implementate rispettivamente nel client e nel server.

All'interno delle due classi è definito un Handler, attraverso il quale è possibile smistare i messaggi in ingresso e in uscita all'interno della connessione. I messaggi sono suddivisi in frame di lunghezza massima di 512 byte. Ogni frame contiene il nome del dispositivo client, un identificativo per il tipo di messaggio e la stringa del messaggio da trasmettere. Successivamente i frame dei messaggi saranno ricomposti per ricostruire l'informazione originaria.

I tipi di messaggio abilitati nella comunicazione sono:

[1] Un servizio per creare grafici di vario tipo, inviando al server i dati, verrà restituito un grafico costruito secondo i parametri richiesti. `https://developers.google.com/chart/`

[2] Il raggio d'azione dipende dalla tipologia di dispositivi bluetooth utilizzati.

1. LOCATION_FRAME: identifica i messaggi che riguardano l'aggiornamento della posizione dell'utente.

2. NOTIFICATION_FRAME: contiene il tipo e il testo di una notifica comunicata sullo smartphone dell'utente. Può riferirsi a chiamate, messaggi, aggiornamenti dello stato della connessione, riproduzione musicale, ecc...

3. ROUTE_FRAME: rappresenta il file in formato JSON contenente tutte le informazioni riguardanti la rotta da seguire, come le coordinate geografiche e le relative altitudini dei punti del percorso.

4. DESTINATION_FRAME: è la stringa di testo dell'indirizzo di destinazione.

5. ORIGIN_FRAME: contiene la stringa di testo dell'indirizzo di partenza.

6. SETTING_FRAME: contiene i messaggi relativi al settaggio dell'applicazione server, attraverso il quale l'utente può modificare determinate componenti.

4.5 Visualizzazione delle informazioni

In questa sezione verranno illustrate le tecniche adoperate per la visualizzazione delle informazioni in realtà aumentata. Quindi parleremo di come l'applicazione server organizzi l'interfaccia grafica e gli oggetti virtuali in base a determinati eventi.

La creazione e l'aggiornamento del contenuto in realtà aumentata è svolto dalla libreria droidAR, attraverso un render grafico per contenuti OpenGL e l'utilizzo di sensori integrati nel dispositivo.

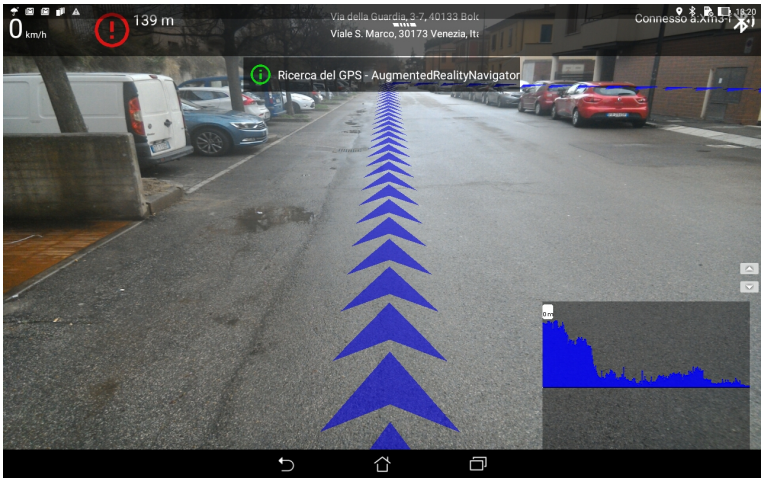

Figura 4.4: Esempio di visualizzazione della rotta attraverso la realtà aumentata in ARMH

4.5.1 Impostazione della UI e creazione dell'ambiente di realtà aumentata

All'avvio, l'applicazione si occupa di inizializzare l'interfaccia grafica di realtà aumentata attraverso le procedure definite nelle classi NewSetup e MyDefaultARSetup. L'intera visualizzazione è composta da tre strati: CameraViewOverlay, GLSurfaceOverlay e GUIOverlay.

4.5.1.1 CameraViewOverlay

CameraViewOverlay è il livello più basso, costituito dalle immagini catturate dalla fotocamera. Rappresenta il livello base, la visione del mondo fisico, sul quale verrà creata la realtà aumentata, vedi figura 4.4.

La classe **CameraView.java**, si occupa della creazione e inizializzazione di tale livello. Inoltre, implementa un SurfaceHolder, il quale da la possibilità di bloccare lo streaming grafico della fotocamera in caso di perdita di focus dell'activity. Questo livello è impostato per permettere la visualizzazione

delle immagini provenienti dalla fotocamera posteriore del dispositivo, in posizione *landscape*.

Si noti che questo strato è ritenuto necessario ai fini dell'emulazione del casco. In caso di implementazione in un vero casco, tale livello sarebbe sostituito dalla visione reale, attraverso la visiera trasparente.

4.5.1.2 GLSurfaceOverlay

GLSurfaceOverlay rappresenta lo strato intermedio, il quale ha il compito di renderizzare e mostrare il contenuto digitale della realtà aumentata, rappresentato dal tracciato di colore blu visibile in figura 4.4. Tutti gli oggetti virtuali vengono inseriti in un contenitore, chiamato *world*, il quale crea uno spazio virtuale tridimensionale. Attraverso un Thread supplementare, *worldThread*, sarà possibile aggiornare ogni elemento digitale inserito nella visualizzazione, in risposta a determinati eventi.

Il livello è creato e inizializzato dalla classe **CustomGLSurfaceView.java**, la quale estende la classe GLSurfaceView, utilizzata per il rendering grafico dei contenuti OpenGL.

All'interno del mondo tridimensionale virtuale sono visualizzati i vari punti geografici relativi al percorso, attraverso dei marker 3D, e il tracciato che unisce i vari punti, costituito da frecce 3D orientate nel senso di marcia. Tali oggetti sono inseriti nella visualizzazione in base all'avanzamento del percorso. Nello specifico, l'applicazione mostra il tracciato che collega i tre marker successivi alla posizione attuale. Quando un nuovo punto di transito viene raggiunto, al tracciato ne viene aggiunto un altro, fino al raggiungimento della destinazione.

4.5.1.3 GUIOverlay

GUIOverlay è lo strato relativo all'interfaccia grafica utente. Questo strato contiene tutti gli elementi visualizzati in modo statico e bidimensionale, i quali non fanno parte della realtà aumentata ma arricchiscono la quantità di informazioni a disposizione dell'utente.

All'interno della libreria droidAR è definita la classe **GuiSetup.java**, in cui sono presenti i metodi per l'inserimento di oggetti all'interno della UI. Questa classe è stata opportunamente modificata, al fine di permettere l'inserimento delle informazioni definite nei requisiti del progetto.

In particolare la UI è composta da diversi LinearLayout, posizionati lungo il perimetro dell'area di visualizzazione, vedi figura 4.4. In alto è presente una barra, nel quale vengono mostrate all'utente le informazioni sulla velocità di marcia, sulla direzione da seguire, gli estremi del percorso e lo stato della connessione Bluetooth. In basso a destra sono presenti due contenitori, utilizzati rispettivamente per la visualizzazione della mappa, con relativo tragitto, e il grafico dell'altitudine. Le notifiche vengono mostrate in una barra dinamica, posizionata in alto, visibile solo in caso di una nuova notifica.

4.5.2 Risposta agli eventi

Gli eventi che possono influenzare il posizionamento degli oggetti all'interno del campo visivo sono diversi. Possiamo dividere questi eventi come legati alla posizione dell'utente o all'orientamento e ai movimenti del casco.

Analizzeremo le varie tipologie di eventi e le reazioni che ne scaturiscono.

4.5.2.1 Aggiornamento della posizione del dispositivo

La localizzazione del sistema è stabilita dall'applicazione client, come precedentemente specificato, in base ai parametri temporali e spaziali delle richieste. Ogni qualvolta il client riceva un aggiornamento della posizione geografica, le coordinate corrispondenti vengono inviate al server.

Il server usa le coordinate trasmesse dallo smartphone per calcolare lo spostamento nello spazio rispetto alla precedente posizione. Infatti il mondo virtuale definito per la realtà aumentata utilizza come riferimento un sistema di assi cartesiani a tre dimensioni, in grado di simulare il mondo reale.

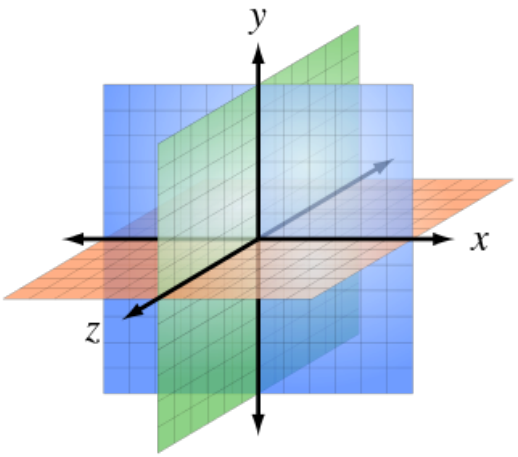

Figura 4.5: Un sistema di assi cartesiani a tre dimensioni

In questo modo lo spostamento lungo il percorso viene elaborato come un vettore nello spazio tridimensionale (figura 4.6):

$$\overrightarrow{r} = \hat{i}r_x + \hat{j}r_y + \hat{k}r_z \tag{4.1}$$

Dove $\hat{i}, \hat{j}, \hat{k}$ indicano i versori dello spazio vettoriale e r_x, r_y, r_z gli spostamenti caratterizzati rispettivamente da latitudine, longitudine e altitudine.

Il vettore è decomposto nella matrice R, affinché sia possibile l'elaborazione da parte del sistema.

$$R = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{4.2}$$

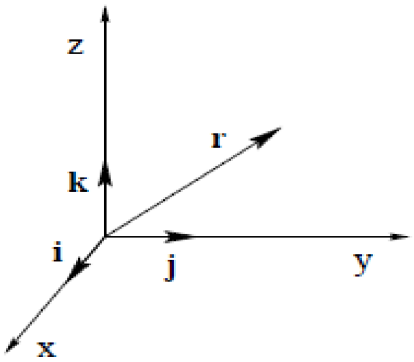

Figura 4.6: Un vettore nel sistema di assi cartesiani

Lo spostamento così calcolato, servirà, alla modifica della posizione attuale all'interno della mappa, al calcolo dell'avanzamento lungo la rotta e alla distanza dal prossimo punto di transito.

Sarà inoltre possibile comunicare all'utente le indicazioni riguardo ad una svolta o una curva, utilizzando un valore che in navigazione prende il nome di *bearing*. Il bearing indica l'angolo sotteso tra il vettore corrente e una retta immaginaria passante tra il punto di destinazione e il nord magnetico. Viene calcolato attraverso la formula:

$$\beta = \arctan(a, b) \tag{4.3}$$

Dove a e b sono le componenti del vettore nello spazio bidimensionale[3]. Il vettore è considerato in un sistema di riferimento costruito sugli assi x e y, tali che l'asse y sia parallelo alla retta passante per la posizione attuale e il nord magnetico (come mostrato in figura 4.7)

[3]In questo caso si tiene conto solo della latitudine e della longitudine.

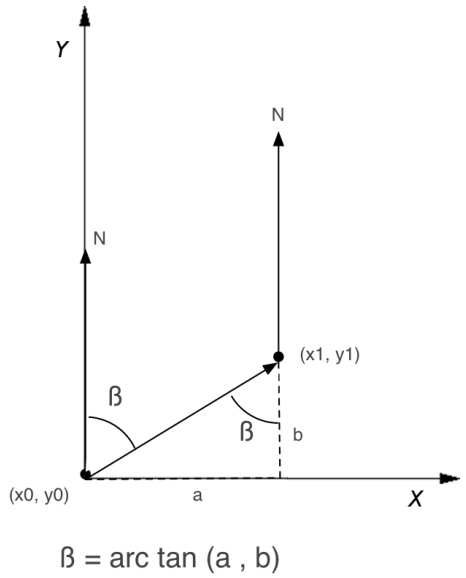

Figura 4.7: Calcolo del bearing tra due punti

La direzione della svolta o della curva è calcolata mediante gli angoli di bearing. Cioè tramite la differenza del bearing del vettore che collega il punto attuale al prossimo punto di transito e il bearing del vettore passante per il punto attuale e il punto successivo al prossimo punto di transito.

Ad esempio, siano p_0, p_1, p_2 tre punti in sequenza nel percorso all'interno dello spazio in questione. Consideriamo i vettori V_0, V_1 che collegano rispettivamente p_0, p_1 e p_0, p_2. Indichiamo con α e β i rispettivi angoli di bearing. Se il valore indicato da $\gamma = \alpha - \beta$ è negativo, la svolta avverrà a destra, altrimenti verrà comunicato di svoltare a sinistra.

L'indicazione di una svolta verrà comunicata al raggiungimento di una distanza minore o uguale di 150 metri dalla sua posizione.

La velocità di marcia è calcolata in relazione al tempo trascorso tra l'aggiornamento attuale e quello precedente.

$$V = \frac{\Delta S}{\Delta T} \tag{4.4}$$

ΔS indica la variazione di spazio tra la posizione attuale e l'ultima posizione registrata, cioè la lunghezza del vettore tra i due punti. ΔT è la differenza tra l'istante attuale e l'istante dell'ultimo aggiornamento.

4.5.2.2 Rotazione rispetto agli assi cartesiani

Vediamo ora in che modo il sistema è in grado di percepire i movimenti di rotazione rispetto agli assi descritti precedentemente. Nel caso di spostamenti lineari, abbiamo visto che, è l'applicazione client ad aggiornare il server sulla sua posizione. Per quanto riguarda invece i movimenti del casco, sarà l'applicazione server ad intercettare tali spostamenti, attraverso l'utilizzo di sensori inerziali, quali: accelerometro, giroscopio e magnetometro.

Introduciamo tre termini utilizzati per definire tali rotazioni:

Imbardata(Yaw): definisce la rotazione lungo l'asse verticale z. Utilizzata per intercettare gli spostamenti durante un movimento rotatorio del casco verso destra o sinistra.

Rollio(Roll): indica la rotazione lungo l'asse orizzontale x. Verrà utilizzato per rilevare l'inclinazione del casco verso destra o sinistra.

Beccheggio(Pitch): è la rotazione lungo l'asse orizzontale y. Tale rotazione sarà necessaria per intercettare l'inclinazione verso il basso o verso l'alto del casco.

Accelerometro

È un sensore in grado di rilevare e misurare l'accelerazione subita dal dispositivo in cui è installato. L'accelerometro misura l'accelerazione lungo i tre assi espressa in m/s^2. La misurazione dell'accelerazione dipende dalla forza di gravità che influisce sul dispositivo:

$$a = -g - \sum F/m \tag{4.5}$$

Dove a rappresenta l'accelerazione del dispositivo, g è l'accelerazione gravitazionale $(9,81 m/s^2)$, F e m sono rispettivamente la forza applicata e la massa al dispositivo. $\sum$ indica la sommatoria dei valori su tutte le componenti x,y,z.

La matrice che determina i valori del vettore di accelerazione è così composta:

$$A = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{4.6}$$

Dove a_x, a_y, a_z indicano rispettivamente le componenti dell'accelerazione sugli assi x,y e z.

Gli angoli di rotazione attorno gli assi x, y e z saranno misurati attraverso le seguenti formule (descritte in [ShRo11, CoEr11]):

$$\theta_x = \arctan(a_y, a_z) \tag{4.7}$$

$$\theta_y = \arctan(a_x, a_z) \tag{4.8}$$

$$\theta_z = \arctan(a_y, a_x) \tag{4.9}$$

Giroscopio

serve per rilevare i movimenti attorno ai tre assi del sistema tridimensionale. Spesso viene utilizzato insieme all'accelerometro per aumentare la precisione della stima dei movimenti. I valori del giroscopio sono indicati in radianti al secondo e misurano la velocità angolare di rotazione attorno i tre assi. La rotazione è considerata positiva se destro-versa, cioè in senso orario. Nel caso in cui il dispositivo si trovasse in una posizione statica, il giroscopio restituisce un valore approssimativamente pari a zero. Affinché sia possibile calcolare l'angolo del moto di rotazione, è necessario fare una valutazione discreta nel tempo:

$$\theta_n = \sum_{i=1}^{n} \omega_i * \Delta t \tag{4.10}$$

Dove θ_n indica l'angolo all'n-esima misurazione, ω_i è il vettore della velocità angolare all'i-esima misurazione e Δt è la variazione di tempo [ShRo11, CoEr11].

Il vettore di velocità angolare sarà identificato dalla matrice :

$$W = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} \tag{4.11}$$

Magnetometro

Permette di stabilire la rotazione rispetto all'asse del nord magnetico. Ciò rende questo sensore molto simile ad una bussola. Il magnetometro misura la forza del campo magnetico ambientale in micro-Tesla (μT) lungo gli assi cartesiani x,y e z. I valori rilevati dal magnetometro, come quelli dell'accelerometro vengono utilizzati per stimare la direzione.

Il seguente vettore indica i valori del campo magnetico misurato lungo i tre assi x,y e z:

$$U = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{4.12}$$

Attraverso le formule trigonometriche e la conoscenza degli angoli θ_x e θ_y (stabiliti attraverso la misurazione degli altri sensori), è possibile determinare la rotazione di imbardata θ_z [ShRo11, CoEr11]:

$$M_x = u_x * \cos(\theta_y) + u_z * \sin(\theta_y) \tag{4.13}$$

$$M_y = u_x * \sin(\theta_x) * \sin(\theta_y) + u_y * \cos(\theta_x) - u_z * \sin(\theta_x) \cos(\theta_y) \tag{4.14}$$

$$\theta_z = \arctan(M_y, M_x) \tag{4.15}$$

Le misurazioni effettuate dal magnetometro possono essere soggette ad errori, causati dall'interferenza di campi magnetici presenti nell'ambiente circostante.

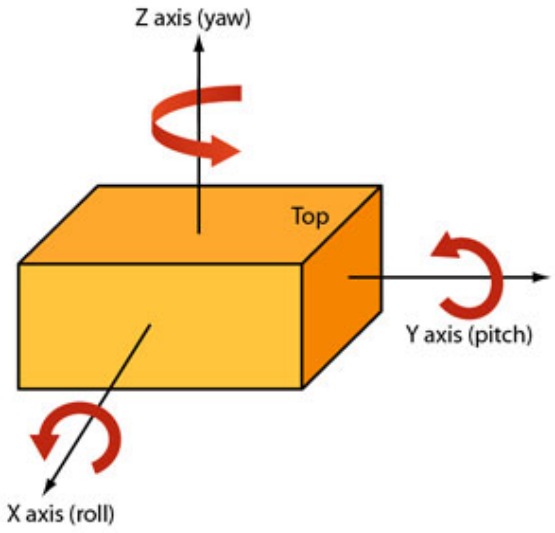

Figura 4.8: Rotazione rispetto agli assi cartesiani

Gli angoli di rollio, imbardata e beccheggio rilevati dai tre sensori serviranno al riposizionamento degli oggetti virtuali, perciò tali movimenti dovranno essere elaborati con estrema rapidità. Infatti, un ritardo nei movimenti del mondo virtuale rispetto a quelli del mondo reale, potrebbe disorientare l'utente.

Mentre imbardata e rollio vengono utilizzati esclusivamente al fine di garantire una corretta disposizione dei contenuti digitali 3D, il beccheggio è anche utilizzato per alternare la visualizzazione della mappa a quella del grafico dell'altitudine. La classe **TiltSensor.java** si occupa, infatti, di misurare tale rotazione, al fine di mostrare la mappa se l'utente inclina il casco verso il basso, altrimenti sarà visualizzato il grafico (un esempio nelle figure 4.9

e 4.10). La misurazione di tale rotazione dovrà soddisfare alcuni requisiti, come: l'angolo di inclinazione e il tempo. In particolare, per visualizzare la mappa il casco dovrà essere inclinato di un angolo minore o uguale della soglia α per un un numero maggiore o uguale a n rilevamenti del giroscopio. Il valore n verrà stabilito in base alla sensibilità del sensore installato nel dispositivo[4].

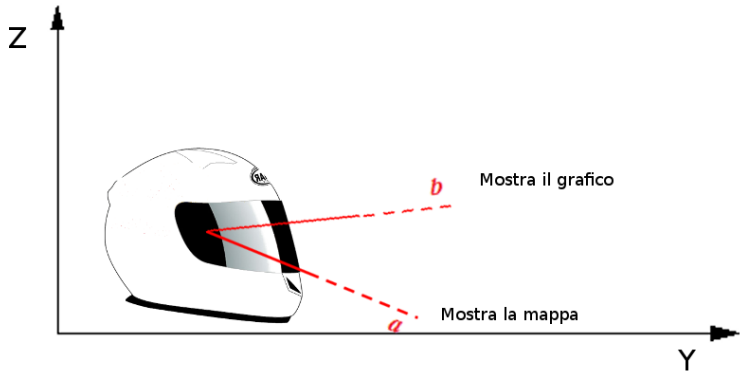

Figura 4.9: Un esempio di reazione dovuta all'inclinazione del casco.

[4]Maggiore è la sensibilità del sensore, maggiore dovrà essere il valore di n.

Figura 4.10: Visualizzazione della mappa nella componente di realtà aumentata di ARHM

4.6 Ottimizzazioni

In questa sezione si vuole fare una panoramica sulle possibili ottimizzazioni delle features, atte a migliorare l'efficienza del sistema. Analizzeremo le strategie adottate per la comunicazione dei dati e le modifiche opportune. Infine, saranno mostrate delle valutazioni sul metodo di elaborazione della rotta.

4.6.1 Ottimizzazione della comunicazione dati

Durante i test dell'applicazione sono stati riscontrati dei rallentamenti dovuti all'eccessivo carico della connessione Bluetooth. In particolare, utilizzando, in aggiunta alla comunicazione dell'applicazione, il Tethering Bluetooth della connessione internet. L'applicazione server, infatti, è stata progettata per sfruttare la connessione internet dello smartphone al quale è collegata, utilizzando l'unico canale di comunicazione presente. Ciò comporta un notevole carico della connessione tra i due dispositivi, con il risultato di un calo di prestazioni dell'applicazione client installata sullo smartphone.

La soluzione a questo problema potrebbe essere l'utilizzo di un secondo canale dedicato alla condivisione della rete internet, oppure l'utilizzo del Wi-Fi Direct.

Standard	Velocità massima	Raggio d'azione massimo
WiFi Direct	300 Mbit/s	150 metri
Bluetooth	25 Mbit/s	10˜100 metri*

(*) Il raggio d'azione varia in base al tipo di dispositivo bluetooth impiegato. Ad esmpio i dispositivi di Classe 2 e 3 pongono un il limite di circa 10 metri. Mentre per i dispositivi di Classe 1 sale circa a 100 metri.

Tabella 4.1: Comparativa tra comunicazione Bluetooth e WifiDirect

Lo standard WiFi Direct permette a due o più dispositivi di connettersi attraverso una rete WiFi, senza l'utilizzo di access point. Questo poiché sono i dispositivi stessi a fornire il servizio di access point. Questa tecnologia risulta essere circa 12 volte più veloce dello scambio dati tramite i comuni dispositivi Bluetooth.

4.6.2 Ottimizzazione del percorso

Le API di Google Maps, per la richiesta di informazioni riguardo ad una rotta, si sono rivelate molto utili ai fini del progetto.

Non sono assenti però limitazioni d'uso. Inoltre, si è notato che la precisione del calcolo della rotta diminuisce con l'aumentare della sua lunghezza. Cioè, mentre per tragitti di piccola e media lunghezza, la quantità di punti restituiti dal server di Google Maps è sufficiente per garantire un buon grado di precisione del tracciato. In caso di percorsi di lunga percorrenza, il numero di punti non aumenta di conseguenza, ma aumenta la distanza tra di loro. Questo si ripercuote nella visualizzazione del tragitto, che in alcuni casi risulta molto imprecisa.

Sono state adottate delle contromisure a questo tipo di problematica, come la suddivisione del tracciato in diversi segmenti, così da effettuare più richieste contemporaneamente per tracciati più corti. Ciò, però, aumenta i tempi di calcolo del percorso e il numero delle richieste non può superare il limite di 10 al secondo.

La figura 4.11 mostra le differenze nella quantità di punti ottenuti tramite richieste singole di percorso e richieste multiple di segmenti di percorso. I grafici mettono in relazione il numero dei punti (sull'asse delle ordinate) e i chilometri del percorso (sull'asse delle ascisse).

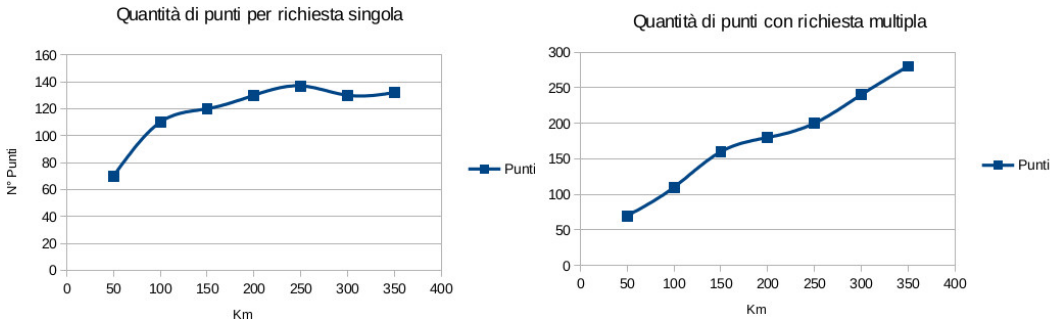

Figura 4.11: Differenza di punti tra richieste singole e richieste multiple

I test sono stati effettuati utilizzando gli stessi percorsi per entrambe le tipologie di richiesta. Poiché, a tragitti differenti, seppur di egual lunghezza, può corrispondere un numero di punti diverso. Come si può vedere dai grafici, l'utilizzo delle richieste multiple aumenta considerevolmente il numero dei punti all'interno del percorso totale. Migliorando di conseguenza la precisione del tracciato visualizzato e delle indicazioni stradali. I tempi di attesa della risposta, però, aumentano proporzionalmente rispetto al numero di richieste effettuate.

Utilizzando un Thread per ogni richiesta, è possibile diminuire il tempo di attesa per l'elaborazione della risposta. Tali Thread verranno eseguiti in parallelo. In questo modo, il tempo di attesa totale sarà pari al tempo di elaborazione della risposta più onerosa. Ma bisognerà, comunque, attendere

l'arrivo di tutte le risposte, affinché sia possibile elaborare i dati ricevuti. Inoltre, così facendo, è molto semplice superare il limite di richieste eseguibili in un secondo.

Un metodo di ottimizzazione sarebbe quello di creare dei percorsi a tappe. Cioè impiegando il metodo appena descritto, ma effettuando la richiesta del percorso n-esimo solo quando il dispositivo ha quasi terminato quello precedente. In questo modo, verrà diminuito il costo computazionale e aumentata la precisione della rotta. Bisognerà considerare, però, i problemi dovuti ad una scarsa o assente connessione alla rete mobile. Durante il viaggio, infatti, potrebbero verificarsi assenze di segnale GSM, causate dal passaggio in zone non coperte.

Capitolo 5

Conclusioni

Abbiamo visto come la realtà aumentata si sia evoluta nell'ultimo mezzo secolo e come si stia affermando nell'era dell'informazione. Si è mostrato, che alcune apparecchiature hanno svolto un ruolo fondamentale in questo, come i dispositivi indossabili.

Il progetto illustrato ha lo scopo di studiare le tecnologie disponibili, al fine di valutare quali siano gli aspetti implementativi e i vantaggi dell'utilizzo della realtà aumentata in ambito di navigazione. In particolare, utilizzando un casco motociclistico dotato di un HMD. Sono stati descritti alcuni dei principali tool di sviluppo disponibili ed è stata analizzata la struttura di una tipica applicazione di AR. Allo stesso modo, è stata illustrata una delle principali piattaforme in grado di sfruttare la visione in realtà aumentata: Android.

Il sistema sviluppato è stato descritto in ogni sua componente, analizzandone le funzionalità e valutandone le prestazioni.

ARMH apporta alcuni vantaggi rispetto alla classica navigazione. L'utilizzo di un visore per AR, infatti, permette di non distogliere lo sguardo dalla strada, diminuendo il rischio di incidenti stradali. Il progetto non è esente da miglioramenti e bug-fix. Ad esempio, l'impiego di un canale di comunicazione più efficiente e l'ottimizzazione delle richieste per la rotta sono alcuni degli obiettivi futuri, mirati a migliorare le prestazioni del sistema.

Infatti, abbiamo visto come i limiti imposti dalle API di Google influiscano sulla precisione del percorso.

A questo punto, verranno mostrati i possibili sviluppi futuri del progetto, in aggiunta a quelli già espressi.

Ricalcolo dinamico del percorso

Il calcolo dinamico del percorso è una funzionalità molto utile in termini di navigazione stradale. Si tratta di offrire la possibilità di cambiare rotta in caso di percorsi più veloci, oppure ricalcolare un percorso se l'utente decidesse di deviare quello già stabilito.

Gli strumenti messi a disposizione da Google Maps, consentono di ricevere informazioni relative al traffico o incidenti stradali di un determinato tragitto. Tali strumenti sono però disponibili solo per la navigazione negli Stati Uniti e in Europa.

L'implementazione di tale funzionalità, richiede una connessione costante alla rete internet mobile e l'utilizzo di algoritmi ottimizzati per il ricalcolo del percorso.

Integrazione di un sistema di riconoscimento degli oggetti

La principale caratteristica del tracking basato su natural feature è la possibilità di riconoscere gli oggetti attraverso la loro struttura fisica, seguirne i movimenti o analizzare la loro posizione. La libreria droidAR è sprovvista di tale funzionalità. Perciò è stata considerata la possibilità di integrare OpenCV all'interno del progetto. Al fine di offrire un sistema di navigazione evoluto, che sia in grado di tracciare i veicoli circostanti, i segnali stradali e i pedoni. In un ipotetico scenario d'uso, chi indossa il casco, sarà avvertito in caso un ostacolo o un pedone venga rilevato davanti a lui. O, ancora, effettuando il tracking dei veicoli che lo precedono, potrà essere analizzata

la loro velocità di marcia, indicando all'utente la distanza di sicurezza da mantenere.

Bibliografia

[BBKM02] Bauer M., Brugge B., Klinker G., MacWilliams A., "Design of a component-based augmented reality framework", *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, IEEE, New York, NY, 2001, 45 - 54.

[BBKM03] Bauer M., Brugge B., Klinker G., MacWilliams A., "Design Patterns for Augmented Reality Systems", 2003, `http://www.macwilliams.de/asa/macwilli2004patterns.slides.pdf`, 15/02/2016.

[Car13] Carli M., *Android 4, Guida per lo sviluppatore*, APOGEO, 2013

[CoEr11] Comotti D., Ermidoro M., "Progetto di Microelettronica Sviluppo di algoritmi per la stima dell'orientamento di un sensore inerziale", 2011, `https://9dof-orientation-estimation.googlecode.com/files/ProgettoMicroelettronica_Relazione_V16.pdf`, 17/02/2016

[DeS10] De Sio Cesari C., "Model View Controller Pattern (MVC)", `http://www.claudiodesio.com/ooa&d/mvc.htm`, 17/02/2016

[Goo16] Google Inc., "Google Maps Android API", 2016, `https://developers.google.com/android/`, 17/02/2016

[Goo16-1] Google Inc., "Google Maps Directions API", 2016, `https://developers.google.com/maps/documentation/directions/`, 17/02/2016

[Goo16-2] Google Inc., "The Google Maps Elevation API", 2016, https://developers.google.com/maps/documentation/directions/, 17/02/2016

[Goo16-3] Google Inc., "Google Places API for Android", 2016, https://developers.google.com/places/android-api/, 17/02/2016

[Goo] Google Inc., "Android Interfaces and Architecture", Android Open Source Project, https://source.android.com/devices/

[Hei61] Heilig Morton L., "Sensorama Simulator", 1961, http://www.mortonheilig.com/SensoramaPatent.pdf , 27/02/2016.

[Hei13] Heinen S., "DroidAR Mobile Locationbased Augmented Reality Framework for Android", https://github.com/simon-heinen/droidar, 2013, 10/12/2015

[ShRo11] Shala U., Rodriguez A., "Ubejd Shala Angel Rodriguez ", 2011, http://hkr.diva-portal.org/smash/get/diva2:475619/FULLTEXT02.pdf, 27/02/2016.

[Sut68] Sutherland I. E., "A head-mounted three dimensional display", 1968, Proceedings of AFIPS 68, 757-764. http://design.osu.edu/carlson/history/PDFs/p757-sutherland.pdf

[SSVK99] Sydiinheimo L., Salmimaa M., Vanhala J., Kivikoski M.; "Wearable and ubiquitous computer aided service, maintenance and overhaul", *Communications, 1999. ICC '99. 1999 IEEE International Conference on*, vol.3, IEEE, Vancouver, BC, 1999, 2012-1017.

[Tho02] Thomas B. et al. "First Person Indoor/Outdoor Augmented Reality Application: ARQuake", 2002, http://www.tinmith.net/papers/thomas-puc-2002.pdf, 10/12/2015

Ringraziamenti

Di seguito desidero ringraziare tutti coloro che mi hanno aiutato nella realizzazione della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore contenuto in essa.

Il Prof. Marco Di Felice, relatore del mio elaborato. L'Ing. Francesco Rambaldi, correlatore e tutor aziendale. Grazie ai loro insegnamenti e alla loro disponibilità è stato possibile realizzare questo progetto.

Un ringraziamento particolare va ai colleghi ed agli amici che mi hanno incoraggiato o che hanno speso parte del proprio tempo per discutere con me le bozze del lavoro.

Vorrei inoltre ringraziare e dedicare questa tesi a chi in questi anni ha dato un contributo fondamentale alla mia formazione. I miei genitori e mia sorella, che per primi mi hanno incoraggiato e supportato. Senza i loro insegnamenti e il loro aiuto, non sarei riuscito a raggiungere questo traguardo. Angela, che più di tutti mi è stata vicina e mi ha incoraggiato in questi anni. Infine, tutti i miei amici, per il sostegno offerto in questi anni di studio.