POLITECNICO DI TORINO

Dipartimento di Automatica e Informatica

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Percipio: sviluppo di applicazioni mobili a supporto dell'esplorazione condivisa di ambienti fisici

Relatore:

ing. Giovanni Malnati

Candidato:

Michele Dottorini

Dicembre 2017

Ringraziamenti

In questo bellissimo momento di conclusione del mio percorso universitario, ringrazio innanzitutto Dio per avermi dato la forza e la speranza nel superare le difficoltà degli studi, ed una famiglia meravigliosa, che grazie ai suoi sacrifici mi ha permesso di studiare e seguire sempre i miei sogni.

Ringrazio quindi la mia famiglia, sempre presente e pronta a sostenermi in qualsiasi momento; il suo prezioso amore è per me un'immensa fortuna.

Voglio ringraziare il mio relatore, Giovanni Malnati, per i suoi consigli e l'aiuto offertomi in questo lavoro di Tesi.

Un grazie speciale alla mia fidanzata, Mariagrazia, per essere entrata nella mia vita dandomi tanto amore e serenità, ed avermi sostenuto in questi miei ultimi sforzi, insieme alla sua bellissima famiglia che tanto ringrazio; sei il regalo più grande che potessi desiderare.

INDICE

CAPITOLO 1

INTRODUZIONE

Il lavoro di Tesi ha riguardato lo studio di tecniche di computer vision e la progettazione di applicazioni per dispositivi mobili incentrate sull'analisi dell'ambiente circostante e la condivisione in mobilità delle sue caratteristiche principali mediante uno streaming dati effettuato in tempo reale.
Tutto questo lavoro è stato incentrato sul concetto fondamentale di "percezione ambientale". Ogni persona è consapevole di essere in un certo ambiente quando si trova fisicamente in esso: questa situazione rappresenta l'esperienza di massima comprensione del luogo, in quanto tutti i suoi cinque sensi (vista, udito, olfatto, tatto, gusto) danno un contributo significativo a percepire dove egli sia, a riconoscere le caratteristiche fisiche dello spazio e ad orientarsi all'interno di esso, avendo perciò la sicurezza di comprendere tutto ciò che lo circonda.
Questa condizione, quindi, rappresenta il caso ideale, dove non serve nient'altro che i propri sensi non riescano a cogliere per avere un'assoluta ed infallibile consapevolezza dell'ambiente circostante, senza quindi il bisogno di alcun aiuto esterno. Purtroppo non sempre questo è realizzabile: si possono presentare, infatti, casi in cui, per svariate ragioni, vi è l'impossibilità di essere presenti fisicamente in un certo ambiente di interesse specifico.
Per esempio, in ambito industriale possono esserci contesti di utilizzo di determinati macchinari o apparati pericolosi per l'uomo, così come in ambito agricolo, dove la necessità di usare in determinati ambiti dei prodotti chimici nocivi per gli esseri umani previene la possibilità di essere fisicamente presenti, previa accurate misure cautelari. Un altro esempio importante riguarda le forze di sicurezza, come nel caso di disinnesco di ordigni esplosivi, dove è chiara la pericolosità della presenza fisica in prossimità di essi; infine si possono citare altri ambiti, come la presenza all'interno di un luogo di interesse comune quale può essere un sito archeologico, in cui è possibile che la partecipazione fisica sia limitata per problemi di sicurezza.
Altri casi importanti da considerare sono quelli dove la presenza reale non comporta particolari rischi ma l'ambiente in cui ci si trova presenta problemi di altra natura, come la difficoltà ad orientarsi; casi di questo genere possono capitare, ad esempio, in un ambiente sotterraneo o in generale al chiuso, dove i normali sensori esterni, come il GPS, non riescono a fornire assistenza.

Tutti gli esempi finora citati conducono al problema fondamentale che è stato affrontato in questo lavoro di Tesi: la mancanza di una percezione completa dell'ambiente di interesse; questo fatto, quindi, comporta la difficoltà da parte delle persone di avere una comprensione ottimale e necessaria di esso, in modo da poterle supportare nell'agire. In questo contesto, perciò, le soluzioni software progettate sono state pensate proprio per poter fornire, nelle situazioni prese in esame, una percezione ambientale maggiore.
In questo lavoro sono stati sperimentati diversi approcci per raggiungere questo obiettivo, con lo scopo di poter affrontare queste problematiche usando degli strumenti che siano il più possibile a basso costo. Si è deciso di assegnare a questo insieme di tecnologie hardware e software il nome "*Percipio"*, prima voce singolare del verbo latino "Percĭpĕre", con il significato di percepire, sentire, provare: sono infatti le sensazioni che si hanno in un contesto generale di appartenenza fisica ad un ambiente.

Inizialmente si è dedicato ampio studio ad uno specifico smartphone Android, chiamato *Lenovo Phab2 Pro*[1]. Questo dispositivo, lanciato sul mercato ad Agosto 2016, possiede caratteristiche del tutto innovative rispetto ai comuni smartphone che si possono acquistare: esso, infatti, è il primo ad essere commercializzato con a bordo una serie di sensori in grado di implementare la piattaforma di realtà aumentata sviluppata da Google, chiamata *Project Tango* (da qui semplicemente Tango). Lasciando nei capitoli successivi di questa Tesi una descrizione dettagliata delle sue caratteristiche, si può dire che il dispositivo possiede le seguenti funzionalità:

1. capacità di effettuare navigazione indoor registrando con accuratezza gli spostamenti all'interno di esso, senza bisogno di segnali esterni.

2. capacità di ricordare un ambiente precedentemente visitato e di poterne memorizzare e condividere le caratteristiche.

3. capacità di rilevare dimensioni, superfici e profondità spaziale dell'ambiente nel quale è immerso.

[1] https://www3.lenovo.com/it/it/smartphones-and-watches/lenovo/phab-series/Lenovo-PB2-690M/p/ZZITZTPPB4M

Oltre allo studio e sviluppo software realizzato su questo dispositivo, il secondo approccio che si è scelto riguarda l'uso dello *streaming multimediale*: è stata infatti progettata un'infrastruttura telematica con lo scopo di far fruire in mobilità i contenuti catturati da una sorgente video a tutti gli utenti appartenenti ad una stessa rete locale, sfruttando la tecnologia Wi-Fi. In questo modo si è affrontato il problema dell'assenza fisica: con questa soluzione, le persone interessate possono prendere parte a tutto quello che la sorgente video è impegnata a catturare, seguendo quindi i suoi spostamenti, grazie all'uso di dispositivi mobili, quali smartphone o tablet, con sistema operativo Android. Tutto questo presuppone determinati requisiti, primo fra tutti la latenza: è fondamentale infatti, per poter garantire una certa interazione con l'ambiente, avere il ritardo tra sorgente e destinazione il più basso possibile, ed infatti questo aspetto è stato di cruciale importanza durante lo sviluppo, indirizzando quindi tutto lo studio verso una soluzione che implementasse uno streaming in tempo reale.
Questo approccio è consistito nella progettazione di due soluzioni le quali si differenziano dall'uso di una diversa sorgente video. Nel primo caso è stata utilizzata una tradizionale webcam collegata ad una scheda elettronica molto diffusa da qualche anno a questa parte, il Raspberry Pi: essa è stata impiegata per poter catturare i dati della fotocamera e spedirli via wireless ai destinatari, utilizzando il modulo Wi-Fi integrato nella scheda. Tutto questo è stato possibile sfruttando una libreria multimediale open source chiamata *GStreamer*, disponibile sia per piattaforme basate su Linux (come il Raspberry Pi) che per quelle basate su Windows o Mac.
Il secondo caso ha riguardato l'uso della visione artificiale, in modo da poter distribuire informazioni diverse rispetto a quelle generate da fotocamere tradizionali. A questo scopo si è utilizzato il dispositivo Tango come sorgente video: esso infatti possiede una seconda camera posteriore, chiamata *Depth Camera*, ed un emettitore di luce infrarossa; l'uso combinato dei due sensori crea una cosiddetta nuvola di punti (nota come *Point Cloud*) ossia un insieme di punti, generati nella porzione di spazio verso il quale i sensori sono rivolti, dove per ciascuno di essi vengono calcolate le coordinate spaziali. Si è perciò deciso di inviare tramite streaming questi dati agli utenti destinatari, rappresentando ciascun punto con un colore diverso a seconda della loro distanza dal dispositivo. In questo modo è possibile riconoscere immediatamente la profondità spaziale dell'ambiente anche non potendoci essere fisicamente presenti.

Tutte queste soluzioni, come accennato, possono essere sfruttate in totale mobilità, tramite il sistema operativo Android. Al giorno d'oggi, data la grande diffusione di dispositivi mobili, tecnologie simili a quelle che sono state fin qui brevemente descritte, ovvero lo streaming dati, sono ampliamente diffuse e numerose app sono disponibili nei principali store digitali. Si può citare, una fra tutte, l'applicazione

Periscope[2], la quale infatti permette di distribuire globalmente contenuti streaming in diretta tramite la fotocamera del dispositivo.
Questa applicazione, come altre sue simili, possiedono però delle caratteristiche diverse rispetto alle soluzioni che sono state sviluppate in questa Tesi. Innanzitutto, la sorgente video dello streaming coincide sempre con la fotocamera principale, operante nello spettro della luce visibile, mentre in questo lavoro si è sfruttato anche il dispositivo Tango per permettere di distribuire i dati della camera di profondità, operante invece nello spettro invisibile dell'infrarosso. Inoltre, la trasmissione dei contenuti in diretta avviene sempre passando da diversi server che distribuiscono i dati agli utenti finali; l'approccio che è stato adottato, invece, consente la comunicazione diretta tra sorgente e destinazione, tramite il solo uso di un router Wi-Fi, indispensabile per l'instradamento dei dati. In questo modo si garantisce teoricamente una latenza molto minore rispetto alla presenza di server tra i nodi di rete partecipanti allo streaming, potendo così realizzare un'interattività maggiore.

In questo lavoro di Tesi, come anticipato, sarà innanzitutto descritto lo studio riguardante la piattaforma Tango: oltre ad analizzare in modo più approfondito i suoi aspetti generali, verrà discussa un'applicazione Android appositamente creata per poter implementare in situazioni reali le funzionalità principali di questo framework ed evidenziarne così i pregi ed i difetti maggiori.
Successivamente si parlerà dello streaming dati in tempo reale, dove verranno analizzate le due soluzioni progettate: si affronterà perciò in dettaglio l'implementazione sviluppata sia per lo smartphone Tango che per il Raspberry Pi, consistente per entrambe nella creazione di un protocollo di comunicazione ad hoc, il quale ha richiesto la modifica della libreria open source GStreamer. Infine si concluderà la Tesi riassumendo i pregi ed i limiti delle implementazioni sviluppate, suggerendo quindi dei possibili sviluppi futuri.

[2] https://www.periscope.tv/

CAPITOLO 2

INTRODUZIONE A PROJECT TANGO

2.1 L'aspetto hardware

Project Tango [1] è una piattaforma hardware e software sviluppata da Google nel 2014 capace di dotare i dispositivi mobili sui quali è installata, quali smartphone e tablet, funzionalità avanzate di computer vision. Come accennato nell'introduzione alla Tesi, il dispositivo utilizzato per sfruttare questa nuova tecnologia è lo smartphone Lenovo Phab2 Pro (figura 1).

Figura 1 - Smartphone Lenovo Phab 2 Pro

Si presenta in realtà come un “phabet”, possedendo infatti una dimensione considerevole dello schermo (ben 6,4 pollici), ma abilitato ad effettuare chiamate, avendo la possibilità di inserire due nano SIM al suo interno; il suo prezzo al lancio è di 499 €. Le peculiarità di questo dispositivo si manifestano già solo visibilmente nella sua parte posteriore (figura 2).

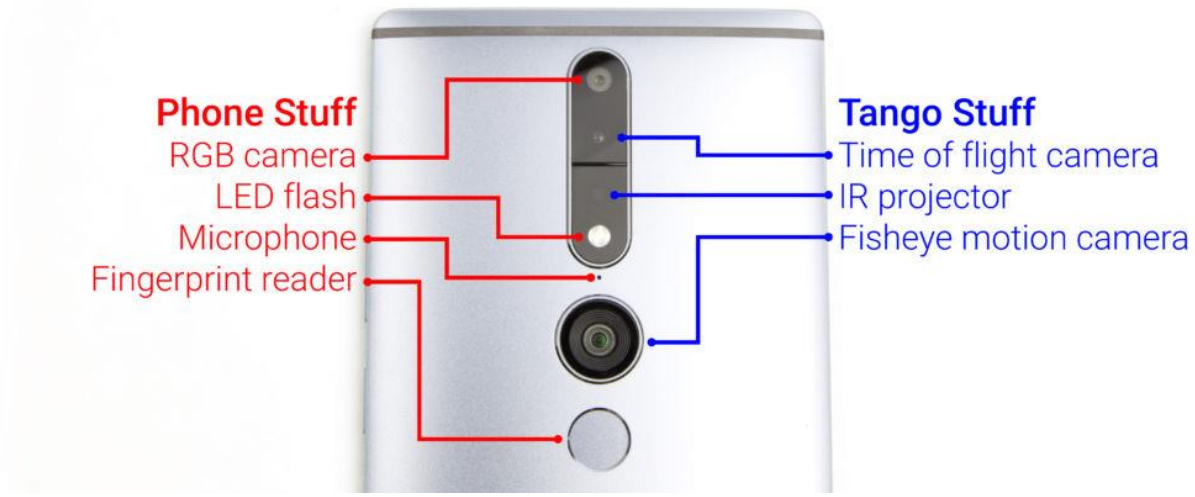

Figura 2 – Insieme dei sensori esterni presenti nello smartphone Lenovo Phab 2 Pro

Si può notare come siano ovviamente presenti componenti standard sulla maggioranza degli smartphone oggi in commercio, quali una fotocamera per scattare foto e registrare video, flash LED, microfono ed un sensore di impronte digitali. A questa componentistica standard si affianca quella caratteristica della tecnologia Tango: è presente una camera grandangolare (Fisheye Camera), una camera cosiddetta “a tempo di volo” (Time of Flight, ovvero la Depth Camera) ed infine un emettitore di luce ad infrarosso. Questo hardware permette, assieme a delle API e librerie software installate nel dispositivo, di sfruttare concretamente il potenziale della tecnologia Tango, la quale si compone di tre tecnologie chiave: *Motion Tracking*, *Area Learning*, *Depth Perception*.

2.2 Motion Tracking

Questa funzionalità permette al dispositivo di tenere traccia dei suoi movimenti nelle tre dimensioni, riuscendo quindi a calcolare in tempo reale la traslazione e la rotazione dello smartphone rispetto all'origine di un sistema di coordinate. La tecnologia alla base di Motion Tracking si chiama Visual-Inertial Odometry (VIO), la quale si differenzia dalle tecniche standard di visual odometry dove, per poter calcolare il cambiamento di posizione, vengono usate diverse immagini della camera e ne vengono selezionate delle caratteristiche chiave, al fine di rilevare cambiamenti di posizione (si pensi, ad esempio, di scattare una foto di un oggetto da lontano ed una più vicino: il cambiamento di dimensione e posizione dell'oggetto permette di calcolare la distanza percorsa dalla fotocamera). Visual-Inertial Odometry, invece, si serve sempre di una camera per rilevare gli spostamenti dei punti chiave di un'immagine (tecnica nota come Feature Tracking), che in questo caso è la camera grandangolare la quale, grazie al suo un ampio campo visivo di circa 160°, permette di catturare maggiori aspetti chiave. Oltre a questa, il dispositivo si serve dei suoi sensori interni di movimento, che vanno a costituire l'unità IMU (Inertial Measurement Unit): questa unità utilizza accelerometro, giroscopio e magnetometro per tenere traccia di quanto il dispositivo accelera e in quale direzione si sposta. L'insieme dei dati forniti dalla Fisheye camera e dall'unità IMU permettono, quindi, di determinare quantitativamente lo spostamento e l'orientamento del dispositivo in uno spazio 3D con un'accuratezza maggiore rispetto alle tecniche standard; questi dati, infine, vengono elaborati in tempo reale fino a 100 volte al secondo.

Una delle applicazioni più interessanti di questa tecnologia è la possibilità di tenere traccia degli spostamenti all'interno di spazi chiusi (indoor navigation) e quindi di assistere l'utente all'interno di determinati percorsi, il tutto senza l'utilizzo del segnale GPS, mantenendo comunque un'ottima accuratezza. Combinando la capacità di tracciare il movimento e rilevare la rotazione in maniera molto accurata, inoltre, permette al dispositivo di essere usato come una camera virtuale all'interno di un ambiente 3D come ad esempio un videogame: la piattaforma Tango, infatti, fornisce un SDK per il famoso motore grafico di videogiochi Unity3D, oltre alla possibilità di integrarsi con il motore grafico OpenGL ES.

Motion Tracking permette la tracciabilità dello spostamento anche in ambienti aperti: ciononostante, dopo lunghe distanze e periodi di tempo di utilizzo, l'accumulo di piccoli errori può condurre ad uno scostamento ("drift") rilevante della traiettoria reale, portando a calcoli errati della posizione assoluta.

2.3 Area Learning

Con la tecnologia Area Learning attiva, il dispositivo ha la capacità di riconoscere un ambiente precedentemente visitato. Ciò è possibile grazie all'uso della Fisheye Camera che, come per Motion Tracking, ha il compito di rilevare gli aspetti chiave dell'ambiente circostante, chiamati *landmarks*, e di salvarli nella memoria del dispositivo. Nello specifico il device memorizza al suo interno la posizione dei landmarks nello spazio tramite una loro descrizione matematica in un file chiamato *ADF* (Area Description File): quando Area Learning viene attivata, lo smartphone legge questo file dalla sua memoria e confronta i landmarks trovati nell'ambiente circonstante nel quale esso si trova con quelli precedentemente salvati nel file ADF e, se questi dati coincidono, il dispositivo riconosce lo spazio come già visitato, senza utilizzare servizi esterni.

La figura 3 mette in luce un esempio di generazione di landmarks in una sala coferenza, dove ciascuno di essi è rappresentato da un punto giallo [2]. La figura 4 mostra invece il riconoscimento nello spazio tridimensionale degli stessi landmarks da parte del dispositivo Tango: per ogni punto vi è rappresentato graficamente una linea gialla che dal device finisce nella posizione esatta del landmark [2].

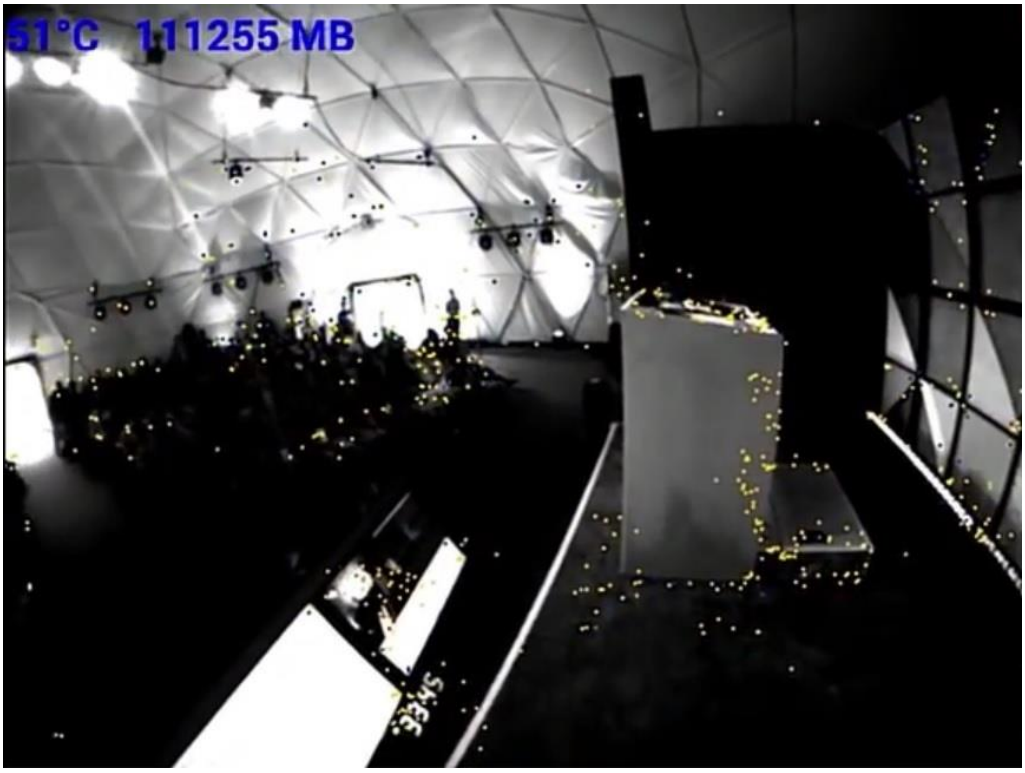

Figura 3 – Landmarks catturati in un generico ambiente

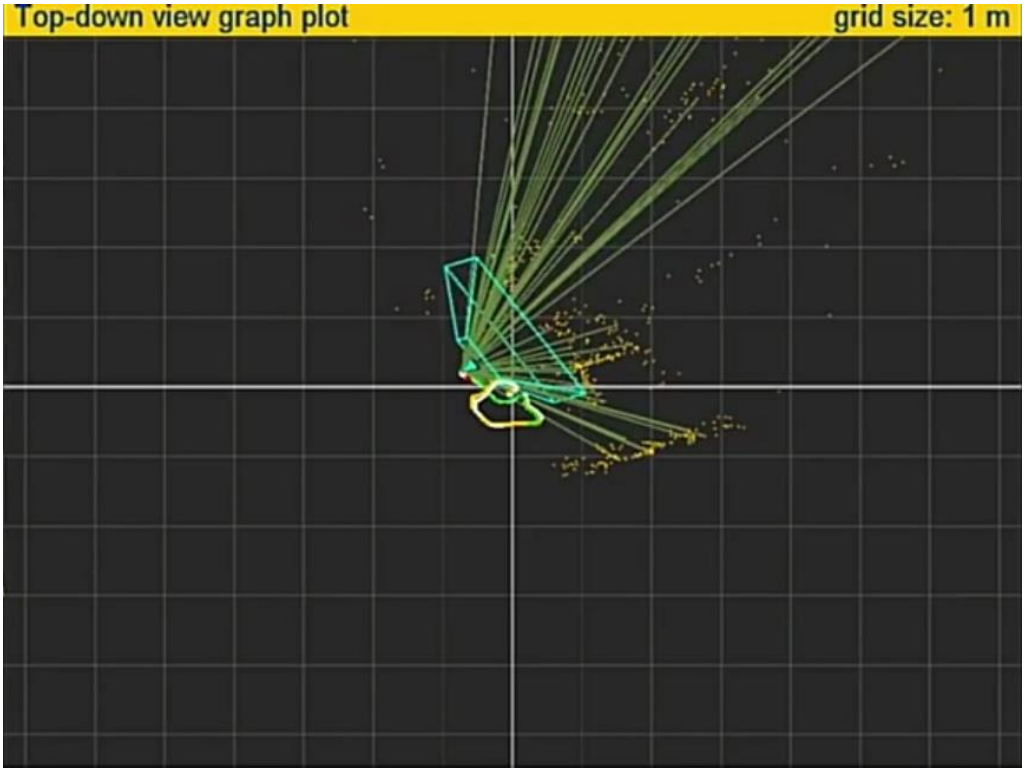

Figura 4 – Riconoscimento nello spazio dei landmarks

La memorizzazione di un'area comporta la possibilità di migliorare i dati forniti da Motion Tracking implementando due aspetti chiave:

- Rendere più accurata la traiettoria, correggendo gli scostamenti dalla posizione assoluta (*drift correction*, chiamato anche *loop closure*).

- Orientarsi e posizionarsi all'interno dell'area appena riconosciuta (*localization*).

2.3.1 Drift Correction (o Loop Closure)

La capacità di riconoscere ambienti precedentemente visitati permette di utilizzare i dati relativi ai landmarks dell'area per correggere gli errori nella stima della posizione, orientamento e movimento dello smartphone: quando il dispositivo vede un luogo che comprende di aver già visto in una precedente sessione, si rende conto di essersi mosso in un "loop" e aggiusta così il suo percorso per poter essere più consistente con le sue precedenti osservazioni. Queste correzioni permettono di correggere la posizione del device e la traiettoria in tempo reale all'interno dell'applicazione. La figura 5 mostra graficamente un esempio di drift correction.

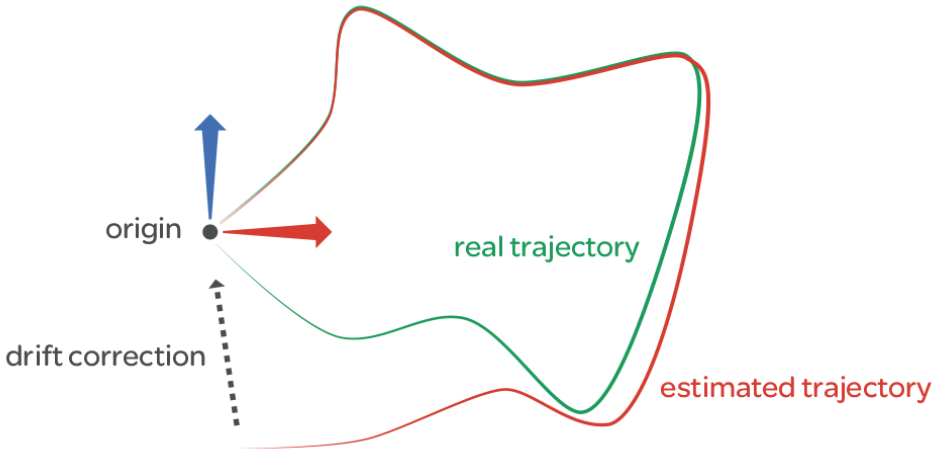

Figura 5 – Esempio grafico di «drift correction»

Quando si cammina all'interno di un luogo, si vengono a delineare simultaneamente due differenti traiettorie: il percorso che si sta fisicamente tracciando (la "real trajectory", in verde) e quello che il dispositivo ha stimato sino a quel momento (la "estimated trajectory", in rosso). Si può notare come la linea rossa con il tempo si sia scostata da quella verde di una certa quantità non trascurabile. Quando il dispositivo ritorna nei pressi dell'origine (il punto dal quale ha iniziato a muoversi) e si rende conto di esserci stato prima, corregge l'errore e aggiusta così la traiettoria stimata per farla tornare ad essere il più possibile simile a quella reale.

Questa proprietà di Tango, quindi, è fondamentale per tenere sempre traccia degli spostamenti compiuti mantenendo l'errore di stima molto basso, potendo così calcolare nel tempo la traiettoria percorsa in modo affidabile.

2.3.2 Localization

Questo aspetto di Tango viene sfruttato seguendo due passi: il caricamento di un file ADF precedentemente salvato e lo spostarsi nell'area i cui landmarks sono stati salvati in quel file. A questo punto il dispositivo provvede a "localizzarsi" in questo ambiente: una volta cioè che lo smartphone riconosce dove si trova, istantaneamente individua la sua posizione in relazione all'origine salvata nel file ADF, ovvero il punto da dove è iniziata la sessione di Area Learning all'interno dell'area salvata. Senza la localizzazione, infatti, il punto di partenza dell'esplorazione di un ambiente viene perso ogniqualvolta si termina la sessione.

Grazie alle funzioni viste finora Tango permette, oltre ad assistere l'utente nella percezione ambientale, che costituisce l'obiettivo di questa Tesi, di costruire esperienze di realtà aumentata interessanti: la possibilità di localizzarsi, infatti, porta a creare ambienti virtuali dove i vari oggetti presenti mantengono esattamente la stessa posizione di quando erano stati lasciati l'ultima volta che quell'area era stata visitata, permettendo di creare un'interazione tangibile col mondo virtuale. Inoltre, la capacità di memorizzare gli aspetti chiave di un ambiente, permette di poter condividere queste informazioni con un insieme generico di utenti: il file ADF, infatti, può essere distribuito su più dispositivi; gli utenti, a questo punto, lo caricano e si localizzano all'interno di questo ambiente potendo interagire sia con esso che con altre persone presenti in quel luogo, beneficiando pure della tecnologia di Motion Tracking (l'uso combinato delle due tecniche è noto come *SLAM* – Simultaneous Localization and Mapping). Esempi interessanti potrebbero essere la navigazione all'interno di un supermercato virtuale, dove l'ADF è reso pubblico dall'azienda che è a capo dell'esercizio commerciale; oppure si può pensare ad esperienze di gioco in multiplayer all'interno di scenari virtuali condivisi.

Parlando di aspetti negativi, si può evidenziare il fatto che la riuscita del riconoscimento di un ambiente dipende dalle sue caratteristiche visive: nei casi peggiori, se si è presenti in un'area che comprende spazi pressoché identici o privi di qualche aspetto chiave, la localizzazione diventa complicata da raggiungere. In generale un ambiente può risultare diverso se viene osservato da diverse angolazioni, con condizioni di luce differenti o con oggetti e mobili che sono sistemati in una posizione differente dall'ultima volta che si è stati lì; la localizzazione, perciò, avrà maggiori possibilità di successo se le condizioni attuali dell'ambiente sono simili a quelle esistenti al momento della creazione del file ADF. Dato che gli ambienti difficilmente nel tempo rimangono gli stessi ma anzi tendono comprensibilmente a cambiare in qualche loro aspetto, Tango permette di poter salvare nello stesso file ADF più sessioni di Area Learning di uno stesso ambiente,

potendo in questo modo inserire all'interno del file aspetti visivi diversi dello stesso spazio, aumentando la probabilità di una corretta localizzazione.

2.4 Depth Perception

Questa tecnica consente al dispositivo di percepire la profondità spaziale e quindi la distanza dagli oggetti, le loro dimensioni e di rilevare superfici, generando dunque i Point Cloud. Tutto questo non rappresenta una novità assoluta a livello tecnologico in quanto nel mercato odierno esistono generalmente tre differenti tecniche capaci di generare la nuvola di punti:

1) *Structured-Light System*

   Viene proiettata una luce nello spettro dell'infrarosso sugli oggetti, seguendo determinati pattern codificati consistente in un insieme di punti [3]. In questo modo il pattern verrà in un certo senso "deformato" a causa della superficie degli oggetti e questo cambiamento viene acquisito da una camera, posta accanto all'emettitore, per permettere di trovare le coordinate tridimensionali dei punti: ciascuno di essi varierà di dimensione in maniera proporzionale alla sua distanza dall'emettitore di luce.

2) *Time-of-flight System*

   È considerata una sottoclasse della tecnica *LIDAR* (Laser Imaging Detection and Ranging): un singolo fascio di luce ad infrarosso alla volta è inviato dall'emettitore verso l'ambiente [4]: la camera, posta accanto all'emettitore, cattura il fascio riflesso da qualsiasi oggetto presente nell'area e, calcolando il tempo trascorso dall'emissione del fascio alla successiva cattura, il sistema può stimare la distanza tra la camera e l'oggetto colpito e quindi la posizione di questo punto nello spazio tridimensionale.

3) *Stereo System*

La scena viene catturata attraverso l'uso di due fotocamere collocate l'una accanto all'altra [4]: l'area è quindi ripresa da due angolazioni leggermente differenti e, tramite un sistema di triangolazione sulle due immagini acquisite, è possibile calcolare la profondità dell'ambiente e quindi degli oggetti che ne fanno parte. Un sistema analogo è usato dagli occhi umani per poter percepire la tridimensionalità dello spazio attorno.

Per quanto riguarda specificatamente Tango, lo smartphone Lenovo Phab 2 Pro è progettato usando la tecnica numero due: l'uso dell'emettitore ad infrarosso, infatti, consente di ottenere delle ottime prestazioni in spazi interni, con scarsa condizione di luminosità e con superfici prive di trame.

Il risultato quindi è la generazione di un insieme di punti ciascuno con delle coordinate spaziali tridimensionali ben precise. Grazie a questa nuvola di punti è possibile riconoscere la profondità dell'ambiente e quindi la distanza degli oggetti dall'osservatore e la loro posizione all'interno dello spazio, caratteristiche chiave nel fornire una percezione ambientale simile a quella che si avrebbe se si fosse fisicamente presenti in esso.

Inoltre questa tecnologia permette, in generale, di realizzare applicazioni di realtà aumentata innovative rispetto al passato: il riconoscimento di superfici ed oggetti, la loro dimensione e distanza dall'utente permette di poter interagire con oggetti virtuali nel momento in cui ci si approccia ad essi, come se realmente fossero sulla scena reale. Inoltre, questi oggetti non solo rimangono nel tempo nella loro posizione originaria senza disallinearsi grazie ad Area Learning, ma con la tecnologia Depth Perception vengono posizionati in accordo con le caratteristiche fisiche degli spazi reali. Un esempio pratico di quanto detto poc'anzi è visibile nelle figure 6 e 7.

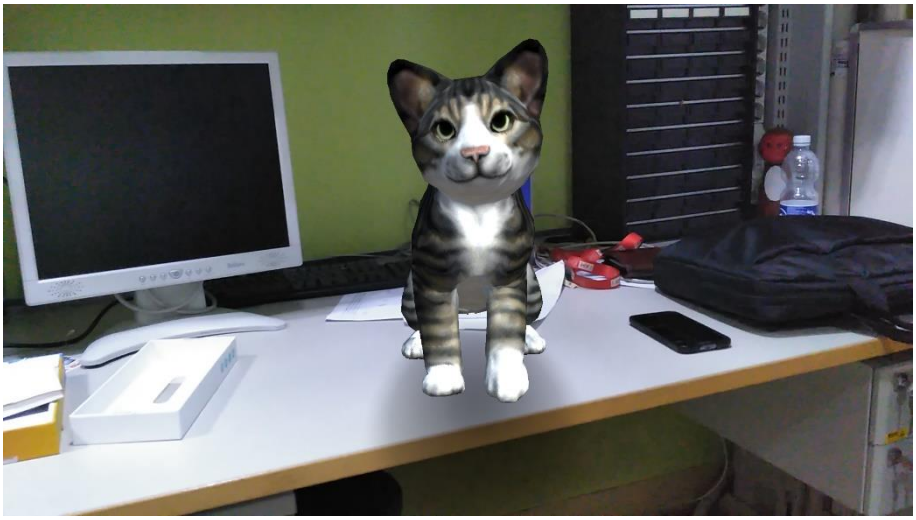

Figura 6 – Collocazione coerente nello spazio reale di un elemento virtuale

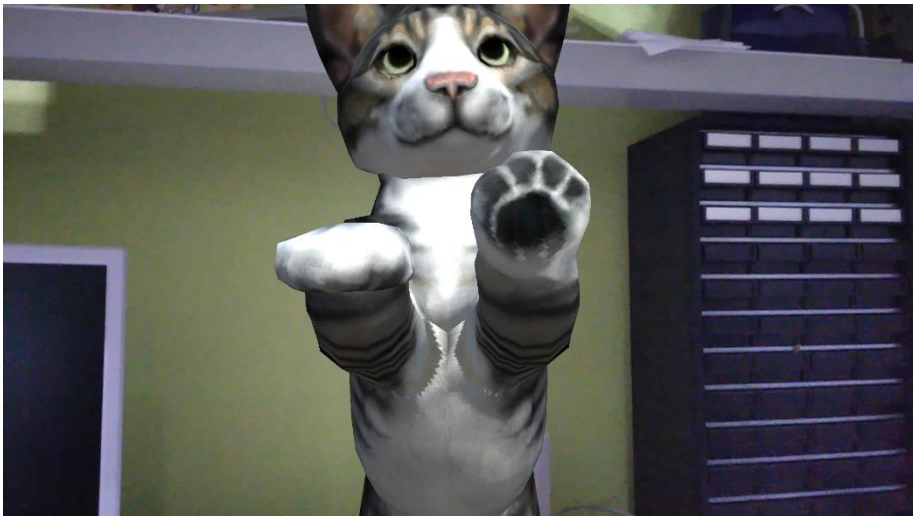

Figura 7 – Interazione tra elemento virtuale e dispositivo Tango in base alla distanza tra loro

Questi esempi sono stati creati sfruttando una delle applicazioni Tango presenti di fabbrica nello smartphone che permettono all'acquirente di familiarizzare con queste nuove tecnologie e di mostrarne le potenzialità.

La figura 6 mette in luce come un generico elemento virtuale (in questo caso un gatto) è collocato sulla superfice piana di un tavolo in maniera realistica: l'animale è come se poggiasse realmente sulla superficie, perché la nuvola di punti ha riconosciuto perfettamente il piano d'appoggio.

La figura 7 mostra come il riconoscimento della distanza tra oggetti, sia reali che virtuali, permette un'interazione naturale con essi: in questo caso il gatto sembra che poggi le sue zampe anteriori sullo schermo dello smartphone, quando raggiunge una certa distanza da esso.

Oltre a queste funzionalità, la nuvola di punti può essere usata in combinazione con la fotocamera tradizionale dello smartphone, operante nello spettro della luce visibile, in modo da assegnare a ciascun punto della nuvola un colore specifico, derivante dall'immagine a colori catturata. Questo permette di generare modelli 3D di oggetti o di un'intera stanza: con questo risultato è possibile ricostruire tridimensionalmente l'ambiente visitato e di poterlo perciò osservare sotto vari punti di vista, diversi da quelli effettivamente occupati durante l'acquisizione. Un risultato concreto di questa tecnica è possibile vederlo nelle figure 8 e 9: esso è stato generato usando una delle migliori applicazioni presenti nello store di Android che sfrutta i sensori Tango a questo scopo, Matterport Scenes[3].

[3] https://matterport.com/matterport-scenes/

Figura 8 – Modello 3D generato dal dispositivo Tango

Figura 9 – Modello 3D generato dal dispositivo Tango, altra prospettiva

Dalle immagini ci si può rendere conto che qualitativamente il risultato ottenuto è discreto, considerando il fatto che il tutto viene generato da un semplice smartphone ed in pochi minuti; la resa finale, infatti, presenta diversi difetti, come la non completa riuscita di mappare gli oggetti in tutte le loro sfaccettature e superfici particolarmente riflettenti, oltre alla difficoltà di rappresentare singoli oggetti da diverse angolazioni senza evitare delle distorsioni evidenti. Per ora si possono considerare queste soluzioni ancora in una prima fase di sviluppo, ed è comprensibile dato che sono funzioni che vengono per la prima volta a trovarsi in uno smartphone; c'è da aspettarsi in futuro dei miglioramenti interessanti. Naturalmente per avere risultati qualitativamente ottimali bisogna rivolgersi a soluzioni professionali ad un costo decisamente superiore di uno smartphone attualmente di fascia alta, tuttavia l'esplorazione spaziale in un secondo momento dall'acquisizione dei dati è possibile e può fornire nuove ed interessanti prospettive.

Come le altre due tecnologie di Tango che si sono analizzate, anche Depth Perception possiede dei limiti: i sensori infatti sono ottimizzati per lavorare bene in spazi ed ambienti chiusi, dove è ottima l'individuazione di superfici, oggetti particolarmente grandi come pareti, tavoli, porte; il sensore ad infrarosso, infatti, permette di rilevare oggetti ad una distanza in media compresa tra 0,5 e 4 metri. Detto ciò, questa tecnologia non è ideale per analizzare oggetti ad una distanza ravvicinata né per riconoscere dei gesti (gesture detection) e quindi non è adatta nell'utilizzo in situazioni ad alta dinamicità. Inoltre ci sono casi dove l'accuratezza dei dati ottenuti viene meno: la radiazione infrarossa, infatti, è meno affidabile se usata verso un'area con una forte sorgente di luce e su superfici molto scure le quali non riflettono la luce, bensì l'assorbono; tutto questo comporta una minore acquisizione di punti.

Terminata questa descrizione generale di tutti gli aspetti che caratterizzano la tecnologia Tango, il prossimo capitolo è dedicato all'analisi dell'applicazione Android sviluppata con tale piattaforma, chiamata "HelloTango".

CAPITOLO 3

SVILUPPO DI PROJECT TANGO IN ANDROID

3.1 L'aspetto software: inizializzare Tango

L'applicazione Android permette di utilizzare in modo concreto le funzionalità principali della piattaforma Tango: per questa ragione l'app sfrutta contemporaneamente tutti i sensori per poter gestire fin da subito le tre tecnologie chiave. Una fonte importante che ha permesso questo sviluppo sono i vari esempi di codice che Google fornisce agli sviluppatori nell'apposito sito web[4].

Inizialmente, è necessario spiegare come l'implementazione di Tango sia legata al concetto di "Service" in Android: un Service è un componente di un'applicazione che esegue delle operazioni in background (tipicamente a lunga durata) non fornendo, quindi, nessuna interfaccia grafica. Esistono due tipi di Services, *started* e *bounded*: il primo è inizializzato da un componente dell'applicazione (come ad esempio un'Activity) e può eseguire le sue operazioni anche se lo stesso componente che l'ha creato viene in un secondo momento terminato; quindi tipicamente questo service compie le sue operazioni e poi ha in gestione la sua corretta terminazione. Il secondo, invece, può essere legato ("bound", appunto) a più componenti ed interagire con loro tramite un'interfaccia client-server, inviando richieste, ottenendo risposte, eseguendo operazioni di Inter-Process Communication (IPC); la differenza sostanziale è che il suo ciclo di vita è strettamente legato ai componenti cui è connesso: solamente quando tutti questi si disconnettono dal bounded service, esso termina.

Premesso ciò, l'architettura di sviluppo di Tango consiste concretamente nella creazione di un bounded service, tipicamente dall'Activity principale, il quale si lega ad essa attraverso delle interfacce di callbacks che forniscono i dati che il sistema Tango ottiene dai suoi sensori, potendo così essere elaborati dai vari componenti dell'applicazione. Questa architettura software è visibile in figura 10.

[4] https://github.com/googlesamples/tango-examples-java

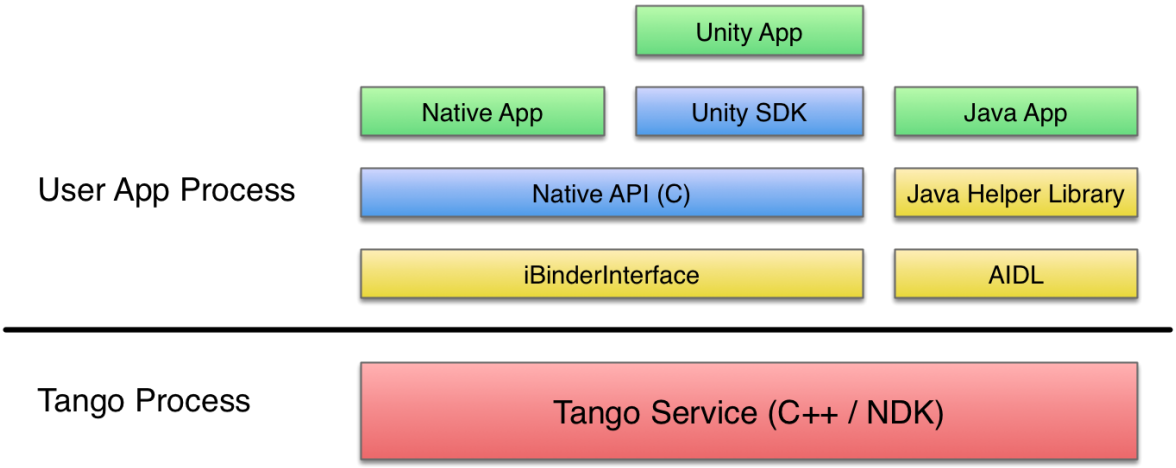

Figura 10 – Architettura software della piattaforma Tango

TangoService è il bounded service menzionato prima. La differenza sostanziale rispetto a quello tradizionale in Android è che quello in Tango esegue le sue operazioni in un thread secondario: le chiamate di callbacks per gestire i dati non sono quindi gestiti dal thread principale.

Google ha fornito agli sviluppatori varie API per poter creare applicazioni Tango utilizzando diversi linguaggi di programmazioni e strumenti: è possibile, per esempio, sviluppare tramite un'interfaccia nativa, usando le API del linguaggio C e Android NDK (Native Development Kit), oppure interfacciarsi con un SDK del noto motore grafico Unity per creare giochi e applicazioni che richiedono una visualizzazione 3D. Infine è possibile sviluppare applicazioni tramite il linguaggio Java, sfruttando le API di Google, le quali si interfacciano al service Tango tramite AIDL (Android Interface Definition Language).

Per questa applicazione si sono studiate ed utilizzate le API Java della piattaforma Tango, seguendo quindi un approccio standard; di seguito vengono evidenziate le fasi principali di inizializzazione di Tango nell'applicazione. Come primo aspetto si evidenziano due oggetti Java fondamentali, appartenenti alle classi `Tango` e `TangoConfig`:

```
private Tango tango;
private TangoConfig tangoConfig;
```

`Tango` è la classe che permette di interfacciarsi con il TangoService e di configurarlo secondo le esigenze, oltre a permettere l'iscrizione alle callbacks che forniscono i dati dei sensori.

La classe `TangoConfig` permette la configurazione dei parametri da passare poi al service; questi consistono, ad esempio, di abilitare o meno il tracciamento dei movimenti così come i sensori di profondità e l'Area Learning, oltre ad altri aspetti.

Le due istanze di queste classi vengono impiegate nel codice seguente.

```
tango = new Tango(MainActivity.this, new Runnable() {

    @Override
    public void run() {
        synchronized (MainActivity.this) {
            try {
                TangoSupport.initialize(tango);
                tangoConfig = setupTangoConfig(tango);
                tango.connect(tangoConfig);
                startupTango();
                isConnected = true;
                setDisplayRotation();

            } catch (TangoOutOfDateException e) {
                e.printStackTrace();
            } catch (TangoErrorException e) {
                e.printStackTrace();
            } catch (TangoInvalidException e) {
                e.printStackTrace();
            }
        }
    }
});
```

Innanzitutto si può osservare come sia necessario passare come parametro al costruttore della classe `Tango` un oggetto di tipo `Runnable`: in questo modo si crea il thread secondario che gestirà la comunicazione tra l'Activity principale e il TangoService.

Il metodo *`initialize`* consente di inizializzare `TangoSupport`, ovvero la libreria che contiene una serie di API Java di supporto alla manipolazione dei dati forniti da Tango; i suoi metodi sono molto importanti per lo sviluppo e verranno in parte citati in seguito.

L'oggetto `tangoConfig` incorpora la configuazione scelta e viene passato nel metodo `connect` che consente di avviare il TangoService e di legarlo, quindi, all'Activity, senza chiamare altri metodi come nei services standard che Android offre. Da questo momento

potranno essere letti i dati di Tango, previa iscrizione alle callbacks che viene effettuata nel metodo `startupTango()`.

La procedura standard di come avviene la configurazione è descritta in questa porzione di codice.

```
private TangoConfig setupTangoConfig(Tango tango) {

    TangoConfig config = tango.getConfig
                              (TangoConfig.CONFIG_TYPE_DEFAULT);

    config.putBoolean(TangoConfig.KEY_BOOLEAN_MOTIONTRACKING, true);
    config.putBoolean(TangoConfig.KEY_BOOLEAN_COLORCAMERA, true);
    config.putBoolean(TangoConfig.KEY_BOOLEAN_DEPTH, true);

     [...]

    config.putBoolean(TangoConfig.KEY_BOOLEAN_LEARNINGMODE, true);


    return config;
}
```

Questo estratto mostra come inizialmente si crea l'oggetto della classe `TangoConfig` passandogli la configurazione di default del sistema, la quale prevede che solo Motion Tracking sia attiva. Una volta creato questo oggetto si possono inserire i parametri di configurazione desiderati, abilitandoli tramite il tipo di dato booleano `true`. Nell'applicazione che si considera, come accennato, vengono abilitate anche le funzionalità di Area Learning e Depth Perception, come si può vedere dal codice; inoltre viene abilitato l'uso della fotocamera posteriore principale (chiamata *color camera*, operante quindi nello spettro visibile) per poter visualizzare l'ambiente circostante sul display dello smartphone.

La gestione delle callbacks è riassunta in questa porzione di codice.

```
private void startupTango() {

    [...]

    tango.connectListener(framePairs, new Tango.TangoUpdateCallback() {
```

```
        @Override
        public void onPoseAvailable(final TangoPoseData pose) {

            [...]

        }


        @Override
        public void onPointCloudAvailable(TangoPointCloudData
                                                pointCloud){

            [...]

        }
    }
}
```

Sostanzialmente basta assegnare all'oggetto `Tango` un listener tramite il metodo `connectListener`; esso ha il compito di implementare i metodi della classe astratta `TangoUpdateCallback` i quali vengono eseguiti ogniqualvolta sono resi disponibili i dati di interesse. Il metodo `onPoseAvailable` permette di ottenere i dati dei sensori di movimento sotto forma di un oggetto della classe `TangoPoseData`; infine `onPointCloudAvailable` fornisce i dati della nuvola di punti che vengono gestiti dall'oggetto della classe `TangoPointCloudData`. Oltre a questa procedura basata sulle callbacks, è possibile ottenere le informazioni di Motion Tracking ogniqualvolta lo si decida espressamente (approccio "polling-based").

Per quanto riguarda Area Learning, la sua gestione, diversa da quella vista per Motion Tracking e Depth Perception, passa attraverso tre metodi Java fondamentali, mostrati nei seguenti tre casi.

```
1. ArrayList<String> fullUuidList = tango.listAreaDescriptions();

2. config.putString(TangoConfig.KEY_STRING_AREADESCRIPTION,
   fullUuidList.get(fullUuidList.size() - 1));

3. String uuid = tango.saveAreaDescription();
```

Il metodo `listAreaDescriptions()` della classe `Tango` consente di reperire tutti i file ADF presenti nel dispositivo, fornendo la lista degli identificativi univoci (UUID) di ciascun ADF.

Una volta ottenuti, un loro possibile impiego è quello di far leggere al sistema Tango uno di questi ADF, in modo da consentire una possibile localizzazione: questo deve essere eseguito in fase di configurazione del TangoService tramite il metodo `putString` della classe `TangoConfig`; perciò, ogniqualvolta si voglia cambiare ADF, bisogna disconnettersi e riconnettersi al service. Nel codice dell'app HelloTango, viene letto il file ADF più recente salvato, ovvero l'ultimo UUID inserito nella lista, come riporta il codice al punto 2.

Infine l'ultimo caso descrive come è possibile salvare una sessione di Area Learning, ovvero generare un file ADF: occorre semplicemente invocare il metodo `saveAreaDescription()` della classe `Tango`, il quale genera l'identificativo del file; questa operazione può comportare tempi molto lunghi di esecuzione, per questo viene eseguita in un thread secondario.

3.2 Sviluppo di Motion Tracking

Dopo aver introdotto anche con degli esempi di codice la prima fase di funzionamento dell'applicazione, ovvero l'inizializzazione di Tango, si analizzano ora gli aspetti tecnici e l'implementazione nell'app delle tre funzionalità principali della piattaforma Tango, a cominciare da Motion Tracking.

Innanzitutto, nella figura 11 si mostra l'interfaccia dell'applicazione, che consente di eseguire tutte le operazioni.

Figura 11 – Interfaccia dell'applicazione HelloTango

Gran parte della schermata è occupata dalle immagini che il sensore della color camera cattura continuamente. Partendo dall'alto, si possono visualizzare i dati di Motion Tracking e quelli di Depth Perception.

Per quanto riguarda i primi, si può verificare come i dati sul tracciamento dello spostamento siano due: si ha la posizione del dispositivo rispetto all'origine di un sistema di riferimento e la sua rotazione, od orientamento, rispetto allo stesso sistema, entrambi misurati in metri. In Tango questi dati insieme vengono chiamati *Pose* del dispositivo e, come già accennato, vengono generati fino a cento volte al secondo.

In questo modo, il sistema riesce a percepire ogni movimento del dispositivo nei cosiddetti "sei gradi di libertà" (figura 12): questo è un modo per chiamare ogni movimento che un corpo rigido è capace di effettuare nello spazio tridimensionale, ovvero andare avanti/indietro (forward/back), su/giù (up/down), destra/sinistra (right/left) ed effettuare le tre possibili rotazioni quali imbardata (yaw), beccheggio (pitch) e rollio (roll).

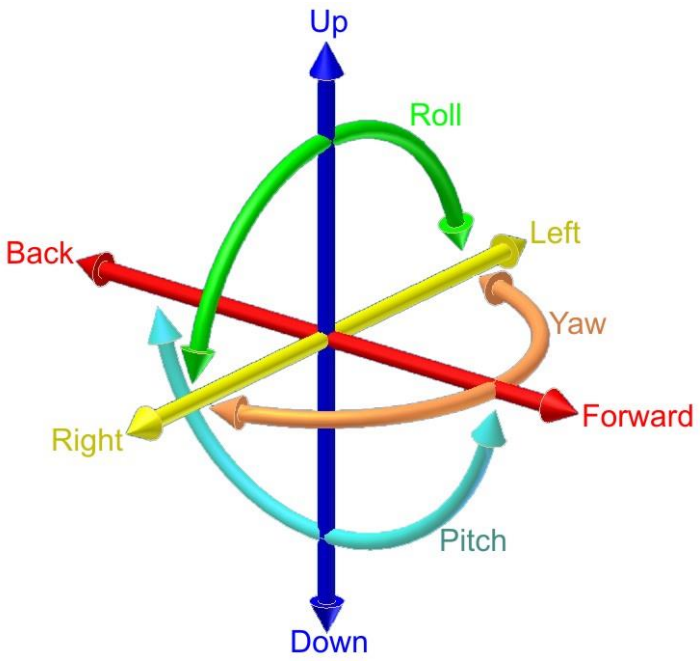

Figura 12 – I sei gradi di libertà rappresentati graficamente

Il sistema di riferimento segue le regole dello spazio Euclideo (figura 13): la posizione del dispositivo, o traslazione, viene descritta tramite le tre coordinate cartesiane [X, Y, Z], visualizzate sullo schermo dell'app in questo ordine, mentre per la rotazione viene utilizzato in Tango un oggetto matematico chiamato quaternione. Questo strumento è stato introdotto nel 1843 dal celebre matematico William Rowan Hamilton come estensione dei numeri complessi, ed uno dei suoi campi applicativi più diffusi è proprio la modellizzazione della rotazione dei corpi. Un quaternione permette di descrivere una rotazione tramite i concetti di asse di rotazione ed angolo di rotazione (figura 14) [5].

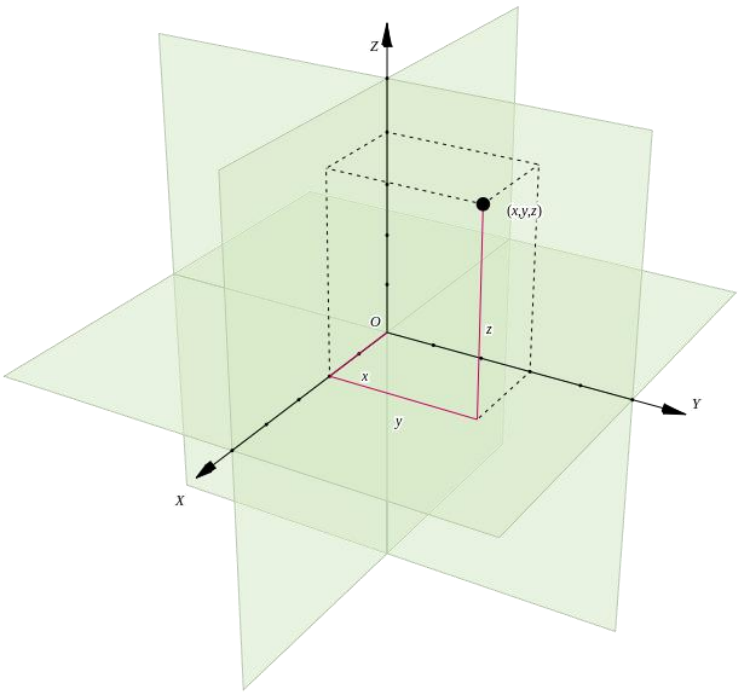

Figura 13 – Spazio Euclideo tridimensionale

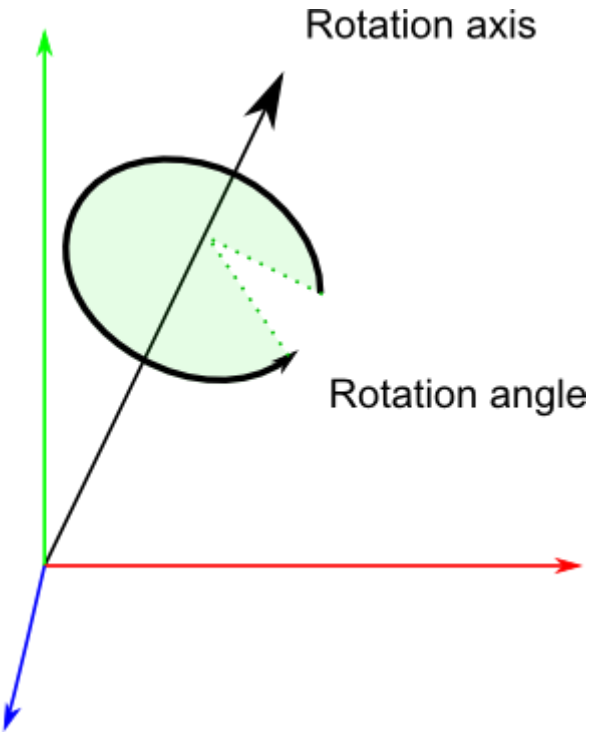

Figura 14 – Rotazione di un corpo nello spazio Euclideo

Il primo descrive l'asse attorno al quale viene compiuta la rotazione mentre il secondo l'angolo che la rotazione forma su questo asse. Un quaternione è quindi formato da un set di quattro numeri [X, Y, Z, W] che vengono calcolati in questo modo (l'angolo di rotazione è espresso in radianti):

X = RotationAxis.x * sin (RotationAngle / 2)

Y = RotationAxis.y * sin (RotationAngle / 2)

Z = RotationAxis.z * sin (RotationAngle / 2)

W = cos (RotationAngle / 2)

Questi quattro valori sono nell'ordine quelli che vengono visualizzate sul display dello smartphone.

Il sistema di coordinate che è usato da Tango per calcolare il Pose segue la convenzione della cosiddetta "mano destra" ed è raffigurato in figura 15.

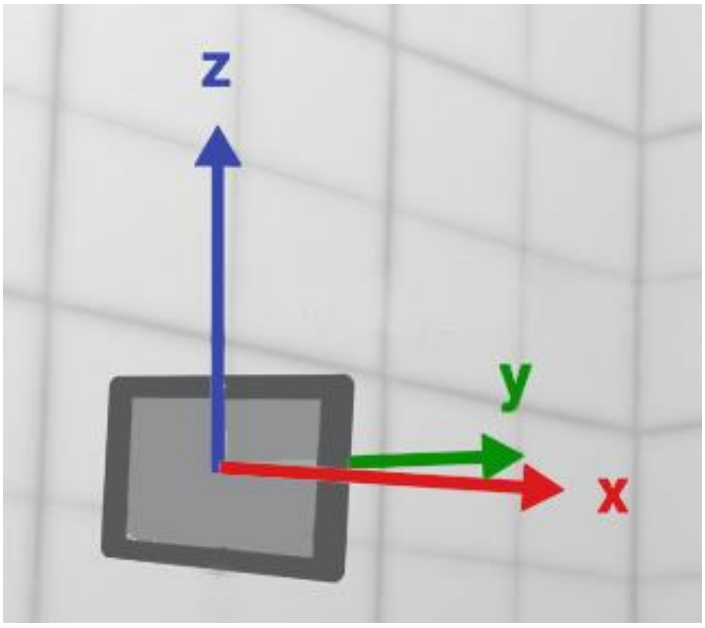

Figura 15 – Sistema di coordinate usato da Motion Tracking

Esso è allineato con il centro del dispositivo: si può osservare come l'asse X dello spostamento è orizzontale ed ha valori positivi verso destra; l'asse Y corrisponde alla profondità (avanti e indietro) ed ha valori positivi in avanti. Infine l'asse Z è verticale con valori positivi verso l'alto.

Fondamentale in tutto ciò che si è analizzato finora è la definizione dell'origine del sistema di coordinate; questo permette di introdurre dei concetti cardini del funzionamento di Tango: *base frame of reference* e *target frame of reference*. Il primo rappresenta il sistema di riferimento "base" ovvero l'origine dal quale vengono fatte le misurazioni dei sensori, ed è ovviamente fisso nello spazio 3D; il secondo rappresenta il sistema di riferimento "target" ovvero l'oggetto verso cui vengono eseguite le misurazioni ed è invece mobile nello spazio.

Per poter ottenere il Pose del dispositivo nel tempo, è necessario prima di tutto decidere cosa sia il base frame e cosa il target frame: in generale Tango permette di scegliere varie combinazioni a seconda di specifiche esigenze, come si vedrà in seguito parlando di Area Learning. Per Motion Tracking, la scelta tipica ricade sul definire il primo come il punto dove è stato inizializzato il TangoService e col secondo il dispositivo stesso: in questo modo il Pose che viene calcolato è quello relativo allo spostamento nello spazio del device a partire dal punto in cui è stato avviato Motion Tracking. Tutto questo può per esempio essere effettuato durante la configurazione del service Tango, prima che possa iniziare il suo lavoro, come è stato fatto per HelloTango e il codice seguente ne mostra la realizzazione.

```
private void startupTango() {

    final ArrayList<TangoCoordinateFramePair> framePairs =
            new ArrayList<TangoCoordinateFramePair>();


    framePairs.add(new TangoCoordinateFramePair(
            TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE,
            TangoPoseData.COORDINATE_FRAME_DEVICE));



    [...]


    tango.connectListener(framePairs, new Tango.TangoUpdateCallback() {

       @Override
       public void onPoseAvailable(final TangoPoseData pose) {
```

```
            if(pose.baseFrame ==
                 TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE &&
               pose.targetFrame == TangoPoseData.COORDINATE_FRAME_DEVICE &&
               pose.statusCode == TangoPoseData.POSE_VALID){


                float translation[] = pose.getTranslationAsFloats();
                float orientation[] = pose.getRotationAsFloats();

                final StringBuilder stringBuilder = new StringBuilder();



                stringBuilder.append("Position: " +
                    translation[0] + ", " + translation[1] + ", "
                    + translation[2] + "\n\n");


                stringBuilder.append("Orientation: " +
                    orientation[0] + ", " + orientation[1] + ", " +
                    orientation[2] + ", " + orientation[3]);


                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        poseTextView.setText(stringBuilder.toString());

                    }
                });


                [...]


            }

            [...]

        }

    }

}
```

Si può notare come il base frame e target frame siano specificati attraverso delle parole chiave contenute nella classe `TangoPoseData`: il base frame che caratterizza l'origine a partire dall'inizializzazione del service è selezionato tramite `COORDINATE_FRAME_START_OF_SERVICE`, mentre `COORDINATE_FRAME_DEVICE` indica come target il dispositivo. Una volta che questa coppia è specificata, viene verificato nella callback `onPoseAvailable` se le informazioni del Pose che sono pervenute sono relative a quella specifica coppia piuttosto che ad altre. Infine si conclude facendo visualizzare questi dati sullo schermo, come si è visto in figura 11.

Il codice presenta un'altra peculiarità importante: esso controlla se i dati che vengono forniti siano validi o meno. L'oggetto `TangoPoseData`, infatti, contiene al suo interno uno stato che se letto permette di capire la stima fatta dal sistema sul Pose fornito. Gli stati che il Pose può assumere sono quattro:

1. TANGO_POSE_INITIALIZING
2. TANGO_POSE_VALID
3. TANGO_POSE_INVALID
4. TANGO_POSE_UNKNOWN

Nel primo caso, il sistema Motion Tracking è in fase di inizializzazione o sta tentando di recuperare il tracciamento a seguito di uno stato invalido; in questo stato i dati non sono ancora disponibili.

Il secondo è il caso migliore: il sistema fornisce dati che considera pienamente affidabili e che quindi possono essere usati con sicurezza.

Il terzo è il caso peggiore, ovvero il sistema Tango ha per qualche ragione incontrato dei problemi e dunque la stima è verosimilmente sbagliata.

Infine l'ultimo caso informa che il sistema è in uno stato sconosciuto.

In questo modo si viene a creare un vero e proprio ciclo di vita del Pose, rappresentato graficamente in figura 16.

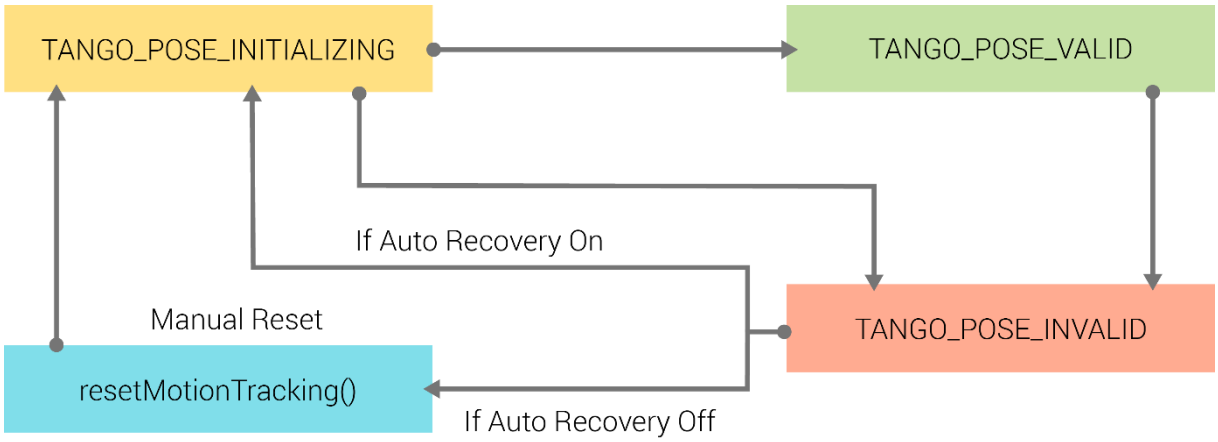

Figura 16 – Ciclo di vita dello stato del Pose

Inizialmente, durante l'avvio del framework Tango, lo stato è in fase di inizializzazione e i dati non sono ancora disponibili. Una volta che il service è avviato, essi sono disponibili e dichiarati come validi. Se il sistema incontra una qualche difficoltà, la stima entra nello stato invalido e a questo punto esistono sono due scenari, dipendenti dalla configurazione scelta all'inizio: se è stato abilitato la funzione di recupero automatico (auto recovery) il sistema resetta immediatamente il sistema Motion Tracking e si ritorna nello stato di inizializzazione; viceversa, si permane sempre nello stato invalido, senza ricevere dati, fintantoché non si invoca il metodo `Tango.resetMotionTracking()`, il quale re inizializza il sistema.

In questa applicazione i dati di Motion Tracking sono stati sfruttati per poter far visualizzare in tempo reale la traiettoria che il dispositivo descrive spostandosi nello spazio. Si possono apprezzare la qualità dei risultati raggiunti negli esempi di utilizzo riportati in figura 17 e 18.

Figura 17 – Traiettoria generata da Motion Tracking (esempio 1)

Figura 18 – Traiettoria generata da Motion Tracking (esempio 2)

Le due figure rendono l'idea di quanto sia preciso il sistema di tracciamento di Tango. In particolare, nella figura 17 l'utente si è spostato dal ciglio del marciapiede andando per poco dritto e poi sterzando a destra giungendo al punto d'arrivo attraversando il tombino posto al centro della strada; l'applicazione ha mostrato proprio questo percorso nelle sue peculiarità: in questo modo si può risalire il tragitto inverso e ritornare al punto di partenza facilmente.

La figura 18 mostra altre particolarità: questo esempio mette in evidenza la sensibilità dei sensori Tango nel rilevare spostamenti anche minimi: in particolare è ben visibile il fatto che l'utente abbia camminato tra il punto di partenza e di arrivo camminando inizialmente dritto, poi sterzando a destra e poco dopo a sinistra, descrivendo una specie di "S". Motion Tracking ha riportato queste informazioni nell'applicazione molto accuratamente.

Nella figura 19 si può apprezzare, invece, la qualità di tracciamento in un ambiente indoor.

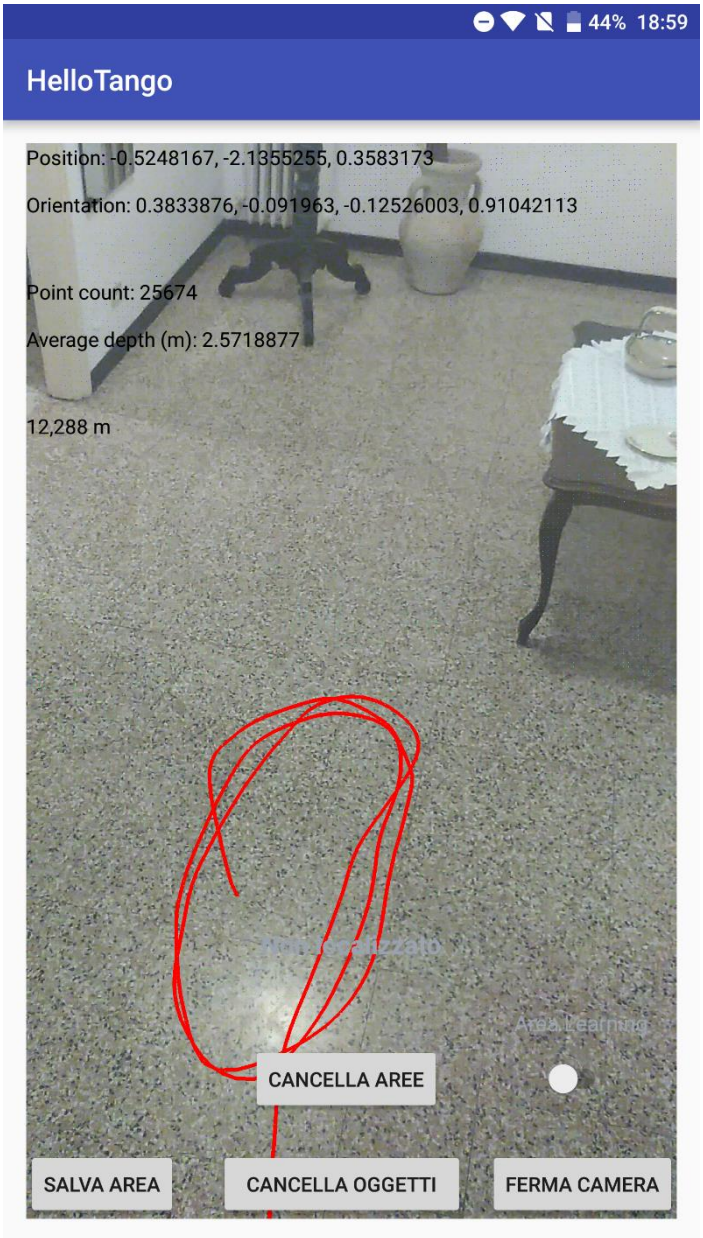

Figura 19 – Traiettoria generata da Motion Tracking (esempio 3)

In questo caso all'interno di una stanza si è voluto percorrere più volte una traiettoria a forma circolare per poi allontanarsi. Il disegno tracciato dall'applicazione sul pavimento mostra come Motion Tracking abbia rilevato molto bene questo particolare spostamento, in un ambiente totalmente chiuso dove un normale smartphone con GPS attivo avrebbe inesorabilmente fallito. Il sistema quindi si può apprezzare essere molto preciso, con una grande accuratezza di dati sia in ambienti esterni che interni.

Un'altra funzionalità presente nell'applicazione HelloTango, che può facilmente essere sviluppata a partire da Motion Tracking, è il calcolo della distanza percorsa dal momento che si è iniziato a stimare il movimento (essa è visualizzabile al di sotto dei dati di Depth Perception). La precisione dei sensori fa sì che si possano rappresentare spostamenti anche di pochi millimetri, ma dall'interfaccia in figura 11 si può evincere come la sensibilità in assoluto sia ancora maggiore: le coordinate di spostamento [X, Y, Z] infatti, così come quelle per la rotazione [X, Y, Z, W], sono rappresentati tramite numeri in virgola mobile a singola precisione (in Java il tipo di dato usato è il `float`) ed il risultato che Tango fornisce è una precisione anche decine, centinaia di volte un millimetro. Si può infatti vedere come i sensori riportino per la coordinata X dello spostamento il valore 0,12484266, per la Y 0,18771929 ed infine per la Z 0,17390938 (tutti espressi in metri); questi numeri dimostrano come la sensibilità sia negli ordini dei micrometri ed inoltre la posizione attuale relativa all'origine, cioè dove Tango è stato inizializzato, è di poche decine di centimetri distante (essendo entrambe questi numeri prossimi allo zero). Ciononostante si è percorso poco più di due metri e mezzo (2,745 m) dall'origine per poi tornare, come detto, quasi esattamente al punto di partenza.

3.3 Sviluppo di Depth Perception

Dopo aver analizzato in maniera concreta la tecnologia Motion Tacking, si passa ora ad effettuare una simile analisi per quanto riguarda Depth Perception. Dalla figura 11 i dati che possono essere letti continuamente sono il numero di punti che sono stati catturati dal sensore di profondità e la distanza media dei punti dallo smartphone. L'aspetto visivo di questa nuvola di punti è possibile osservarla nelle immagini seguenti (figura 20 e 21). Tramite l'applicazione è possibile cambiare istantaneamente la visualizzazione sul display dalla fotocamera principale (figura 20) alla sola nuvola di punti tramite il pulsante "Ferma Camera". Una volta che si è nella modalità di visualizzazione dei Point Cloud, lo stesso pulsante riabilita la visualizzazione della fotocamera principale.

Figura 20 – Prima immagine campione per visualizzare i point cloud (vista fotocamera)

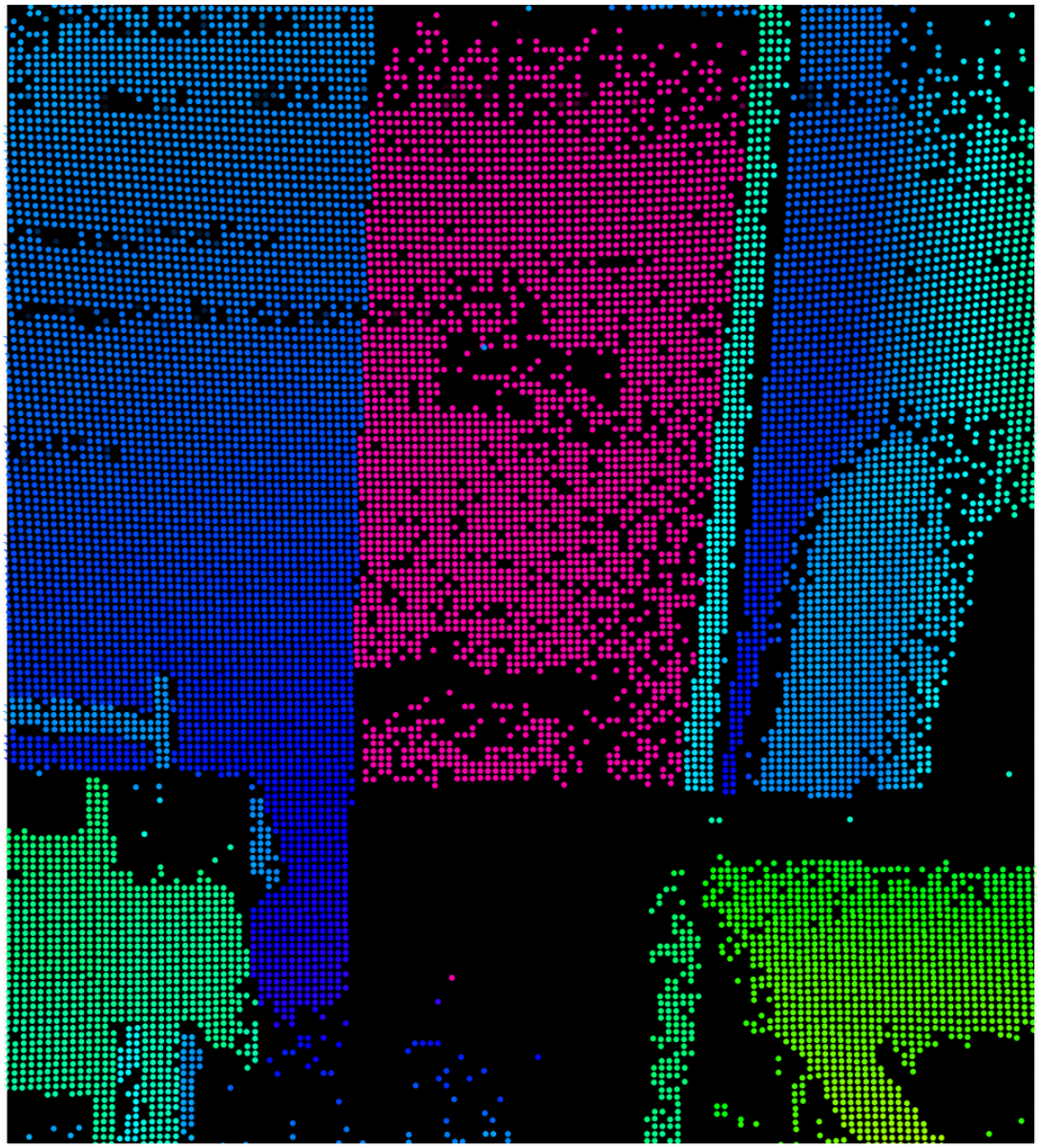

Figura 21 – Prima immagine campione per visualizzare i point cloud (vista nuvola di punti)

La figura 21 mostra, come si era già accennato, che la nuvola di punti è stata rappresentata tramite una gamma di colori, a seconda della distanza dalla sorgente di luce ad infrarosso; in generale si sono scelti colori "caldi" per gli oggetti più vicini, mentre per quelli più lontani i colori sono "freddi". Nell'esempio i punti sono tendenti al verde nella media distanza, mentre la parete bianca accanto alla porta presenta colori blu e infine la parte più lontana, cioè il muro al di là della porta distante circa cinque metri e mezzo, ha come colore il violetto; nella parte in basso a destra, invece, i colori tendono leggermente all'arancione. Con questa grafica ci si può rendere conto di quanto il sistema sia molto efficace nel rilevare la distanza dall'ambiente che lo circonda e di riuscire, quindi, a percepire la sua profondità. In diversi punti, comunque, il sistema Tango non riesce in modo perfetto a stimare la distanza degli oggetti: come si può vedere dalla figura 21, parte del tavolo e del pavimento non presentano affatto punti e quindi la densità finale non raggiunge il massimo consentito; questo è abbastanza comune nella maggior parte dei casi, dove a causa di determinate prospettive e materiali, può capitare di avere dei piccoli spazi vuoti nella nuvola di punti.

Le due immagini che seguono (figura 22 e 23) presentano sempre uno spazio interno con degli oggetti più vicini alla sorgente. In questo caso si vede come la parte più vicina al dispositivo, la maglia raccolta sopra la sedia, abbia quasi totalmente punti di colore giallo scuro/arancione. Anche qua per le zone mediamente più lontane, come il letto, i punti sono sul verde (verde acqua per il pavimento, essendo più lontano), mentre la zona della porta presenta tutti punti blu; essendo questo spazio più piccolo, non vengono evidenziati colori più "freddi" come il viola nella figura 21. Sono presenti anche in questo esempio piccole zone con assenza di punti, come il pavimento nella sua parte più lontana e il poggia braccio che, essendo completamente nero, assorbe la luce infrarossa, ma globalmente i sensori riescono a mappare la quasi totalità dell'ambiente.

La massima distanza percepibile che si è sperimentato è di circa sei metri; questo dato è ottimo se viene pensato per un uso in ambienti chiusi; spingersi più in là, secondo quanto spiegato da Google, avrebbe richiesto un maggior consumo di potenza per il sensore ad infrarosso e maggior processamento dei dati, riducendo in tal modo l'autonomia dello smartphone. Le successive immagini (figura 24 e 25) dimostrano proprio che in uno spazio aperto la quantità dei punti calcolati è molto scarsa. La figura 24 mostra come siano stati calcolati solamente 3774 punti, sostanzialmente tutti localizzati nella parte bassa dell'immagine, dove sono presenti dei paletti e parte della strada, mentre la zona centrale è sostanzialmente vuota (figura 25); mediamente una cattura indoor fornisce un numero di punti di circa 35000.

Figura 22 – Seconda immagine campione per visualizzare i point cloud (vista fotocamera)

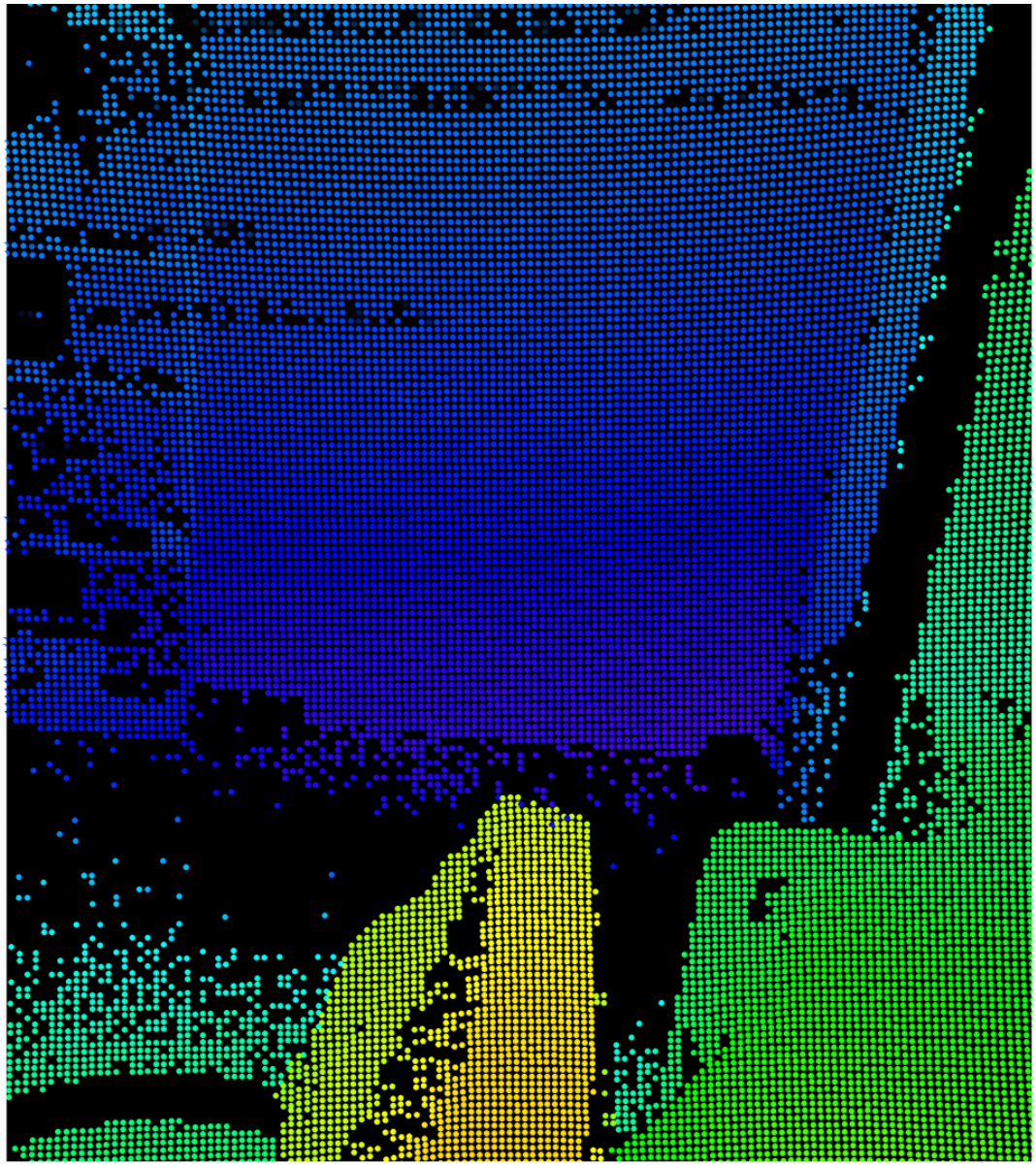

Figura 23 – Seconda immagine campione per visualizzare i point cloud (vista nuvola di punti)

Figura 24 – Terza immagine campione per visualizzare i point cloud (vista fotocamera)

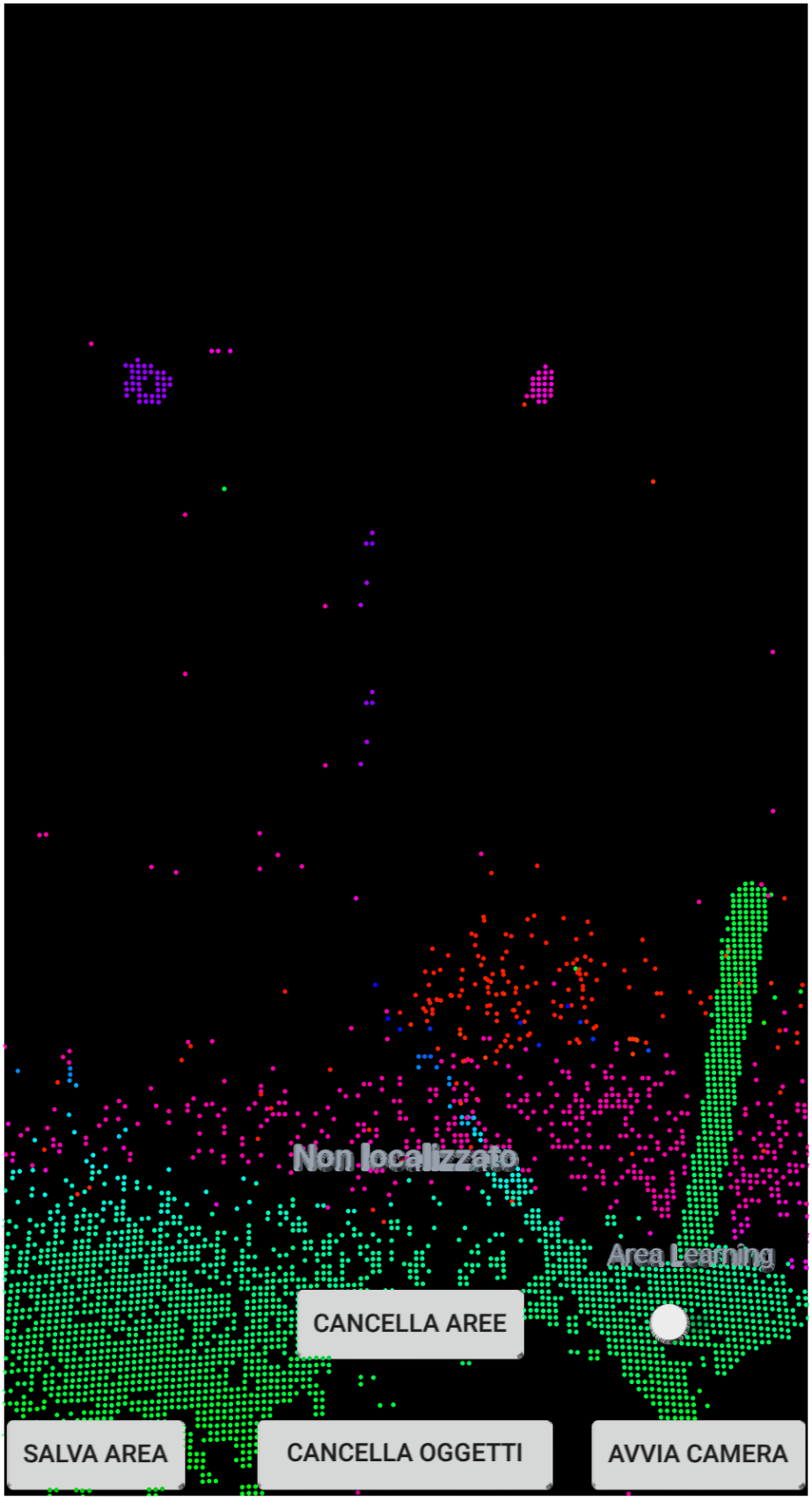

Figura 25 – Terza immagine campione per visualizzare i point cloud (vista nuvola di punti)

3.3.1 La libreria grafica Rajawali

Tutti gli esempi visti finora e quelli che verranno mostrati per analizzare la funzionalità di Area Learning, necessitano di disegnare delle primitive grafiche sovrapponendosi alle immagini della fotocamera; in questo modo si viene a creare la cosiddetta "realtà aumentata", dove oggetti del mondo reale e del mondo virtuale si uniscono insieme per generare un'unica esperienza più ricca di contenuti. Finora le primitive grafiche che si sono potute osservare sono un insieme di linee rette per descrivere la traiettoria ed un insieme di punti per creare i Point Cloud. Per poter realizzare tutto questo nell'applicazione HelloTango si è utilizzata una libreria grafica open source basata su OpenGL ES, chiamata Rajawali[5]. Questo framework grafico permette di utilizzare le primitive e le istruzioni presenti nelle specifiche OpenGL sfruttando direttamente il codice Java: in questo modo si facilita la vita allo sviluppatore, cui vengono fornite delle API di più facile utilizzo, senza usare direttamente il codice di OpenGL.

La figura 26 presenta un diagramma UML (Unified Modeling Language) che descrive, attraverso classi ed interfacce, il modo in cui Rajawali astrae l'utilizzo di OpenGL. La parte in alto della figura mostra l'implementazione standard di OpenGL in Android: il primo elemento caratteristico è la classe `GLSurfaceView`; essa è l'estensione della classe base `View` di Android, ed è la superficie di disegno di tutti gli oggetti grafici, sfruttando la GPU presente nel device. Tutto questo è possibile delegando i comandi di disegno ad un render grafico, il quale viene creato implementando l'interfaccia `GLSurfaceView.Renderer`: questa è la parte più complessa, perché vi è presente tutta la logica di disegno; tutto il lavoro del render avviene in un thread secondario.

[5] https://github.com/Rajawali/Rajawali

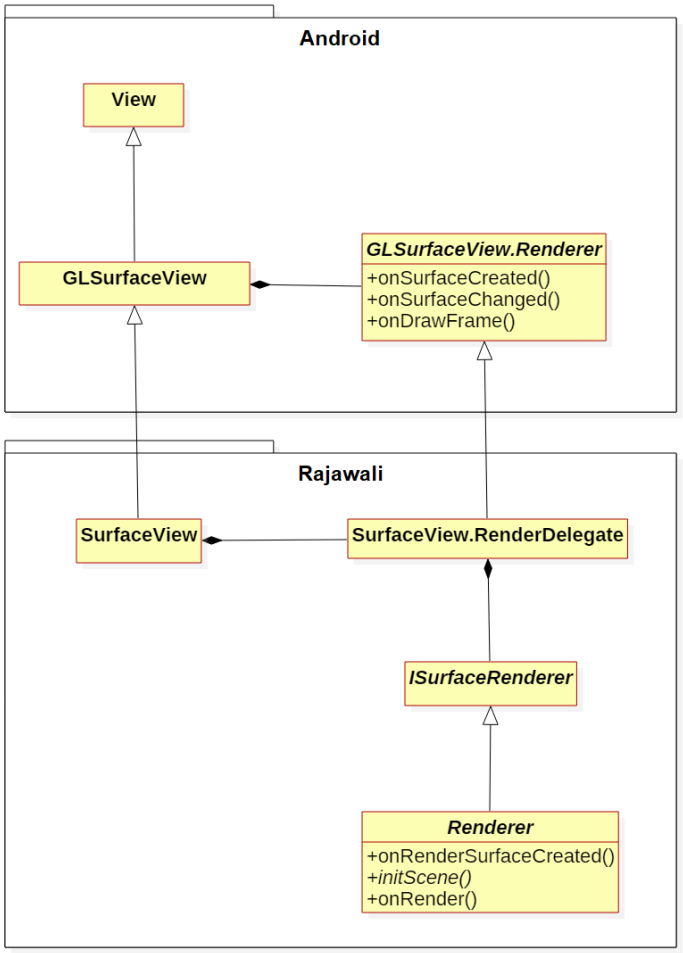

Figura 26 – Relazione tra Rajawali ed OpenGL mediante classi ed interfacce

La parte in basso presenta le componenti principali del framework Rajawali. Sostanzialmente sono simili a quelle presenti in OpenGL, ovvero una `SurfaceView` che estende la `GLSurfaceView` classica di Android ed il render che si occupa del disegno grafico. L'implementazione di quest'ultimo tuttavia è più facile dato che i comandi OpenGL sono implicitamente invocati utilizzando il codice Java, e consiste di tre metodi. Dei tre, il metodo `initScene()` è l'unico ad essere astratto (quindi la sua implementazione è d'obbligo), e rappresenta l'inizializzazione della scena virtuale, all'interno della quale tutti gli oggetti vengono creati. Si mostra ora parte dell'implementazione di questo metodo, per meglio spiegare il funzionamento di Rajawali con l'applicazione HelloTango.

```
@Override
protected void initScene() {


    if (backgroundQuad == null) {
        backgroundQuad = new ScreenQuad();
        backgroundQuad.getGeometry().setTextureCoords(textureCoords0);
    }


    Material tangoCameraMaterial = new Material();
    tangoCameraMaterial.setColorInfluence(0);

    tangoCameraTexture = new StreamingTexture("camera",
        (StreamingTexture.ISurfaceListener) null);

    try {
        tangoCameraMaterial.addTexture(tangoCameraTexture);
        backgroundQuad.setMaterial(tangoCameraMaterial);

    } catch (ATexture.TextureException e) {
        Log.d(TAG, "Exception creating texture for RGB camera contents",
            e);
      }

    getCurrentScene().addChildAt(backgroundQuad, 0);


    trajectory = new Trajectory(Color.RED, 5);
    trajectory.setPosition(0, -3, 0);
    getCurrentScene().addChild(trajectory);

    mPointCloud = new PointCloud(MAX_NUMBER_OF_POINTS, 4);
    getCurrentScene().addChild(mPointCloud);


    getCurrentCamera().setNearPlane(CAMERA_NEAR);
    getCurrentCamera().setFarPlane(CAMERA_FAR);

    getCurrentCamera().setFieldOfView(80);


    [...]

}
```

In Rajawali tutti gli elementi che vengono disegnati derivano dalla classe base `Object3D`. Dentro di essa sono presenti tutti i metodi e gli attributi che permettono di disegnare qualunque oggetto virtuale di qualunque forma tridimensionale. Per fare alcuni esempi, al suo interno è presente un'istanza della classe `Material`, che definisce delle caratteristiche visuali dell'oggetto, come texture e luce ambientale. Un'altra classe fondamentale presente in quella base è `Geometry3D`, dove vengono memorizzati tutti i

dati necessari per disegnare un oggetto, quali l'insieme dei suoi vertici, le coordinate delle texture, i colori, le normali.

Nello sviluppo di HelloTango si può vedere come il render viene inizializzato creando degli oggetti, derivanti da `Object3D`, che rappresentano la traiettoria e la nuvola di punti, rispettivamente `Trajectory` e `PointCloud`. Con poche righe di codice si assegnano la loro posizione iniziale ed infine vengono inseriti all'interno della scena virtuale, e quindi disegnati da OpenGL; con altrettanti semplici passaggi si imposta la camera di visualizzazione all'interno della scena. L'efficacia di Rajawali si evince anche dagli esigui passaggi richiesti per renderizzare le immagini della color camera all'interno della SurfaceView, tramite il metodo `StreamingTexture`.

Una volta che questi oggetti grafici sono inseriti nella scena, verranno di volta in volta aggiornate la posizione e l'orientamento nello spazio tramite i dati di Tango che vengono forniti al render.

Concludendo questo approfondimento riguardante gli strumenti utilizzati per permettere il disegno degli oggetti virtuali all'interno dell'app, si porta ora in evidenza il fatto che i dati provenienti da Tango utilizzano dei sistemi di coordinate differenti da quelle usate da OpenGL (e quindi Rajawali), descritte graficamente nella figura 27. Per quanto riguarda il Pose, è stato già mostrato quale sistema di riferimento viene adottato, mentre per quanto riguarda la nuvola di punti, di contro, viene adottato un sistema di riferimento diverso: in questo caso ciascun punto possiede sempre i tre valori, espressi in metri e di tipo `float`, delle coordinate spaziali [X, Y, Z] ma ora la Y è l'asse verticale positivo verso il basso, e la Z è l'asse perpendicolare al piano della depth camera (profondità), positivo verso il suo asse ottico. L'asse X continua ad essere orizzontale e positivo verso destra.

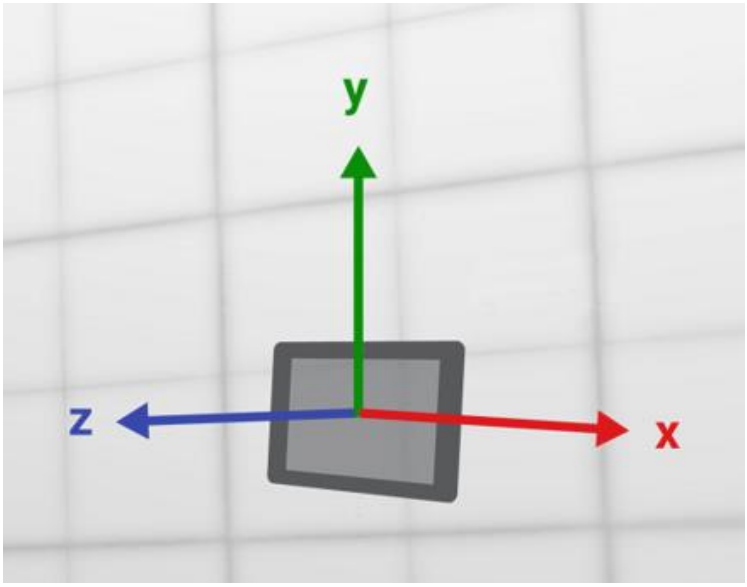

Figura 27 – Sistema di coordinate utilizzato in OpenGL/Rajawali)

Tutto questo porta a dover effettuare delle operazioni di conversione sui dati che i sensori Tango forniscono per poter essere utilizzati da OpenGL nelle operazioni di disegno. Le API del framework Tango permettono in maniera agevole di effettuare questo tipo di operazioni, tramite la già citata classe `TangoSupport`. Essendo questo un punto fondamentale dello sviluppo dell'applicazione, si mostra tramite delle porzioni di codice come vengono passati i dati relativi ai PointCloud una volta letti.

```
 TangoSupport.TangoMatrixTransformData transform =
        TangoSupport.getMatrixTransformAtTime(pointCloud.timestamp,
                TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE,
                TangoPoseData.COORDINATE_FRAME_CAMERA_DEPTH,
                TangoSupport.TANGO_SUPPORT_ENGINE_OPENGL,
                TangoSupport.TANGO_SUPPORT_ENGINE_TANGO,
                TangoSupport.ROTATION_IGNORED);
if (transform.statusCode == TangoPoseData.POSE_VALID)
    renderer.updatePointCloud(pointCloud, transform.matrix);
```

```
public void updatePointCloud(TangoPointCloudData pointCloudData, float[]
openGlTdepth) {

    mPointCloud.updateCloud(pointCloudData.numPoints,
pointCloudData.points);
```

```
    Matrix4 openGlTdepthMatrix = new Matrix4(openGlTdepth);


    mPointCloud.setPosition(openGlTdepthMatrix.getTranslation());
    mPointCloud.setOrientation(new
Quaternion().fromMatrix(openGlTdepthMatrix));

}
```

La prima parte di codice presenta il metodo `getMatrixTransformAtTime` della classe `TangoSupport`, presente nell'Activity principale: tramite questa invocazione, viene creata una matrice di trasformazione, specificando il sistema di riferimento base e quello target. In questo caso il sistema base è il punto di avvio del service Tango, mentre il sistema target non è quello del dispositivo ma della camera di profondità, proprio perché utilizza un sistema di coordinate differenti. Infine si specifica per ciascuno il sistema di coordinate: per il riferimento base si sceglie quello di OpenGL mentre per il target quello nativo di Tango; questa differenza di scelta ha come risultato la creazione di una matrice che permette di passare dal sistema di coordinate Tango a quello di OpenGL. Essa viene poi usata dal metodo `updatePointCloud`, invocato nel render grafico, assieme ai nuovi dati della nuvola di punti: in questa seconda parte del codice vengono innanzitutto aggiornati i dati dei Point Cloud da visualizzare sullo schermo, quali i vertici e i colori, invocando `updateCloud` il quale accetta il numero di punti della nuvola e l'oggetto `FloatBuffer` contenente tutte le coordinate di ciascun punto. Una volta aggiornate queste informazioni, per essere posizionati correttamente all'interno della scena, si aggiornano la posizione e l'orientamento dell'oggetto `PointCloud` sfruttando la matrice di trasformazione calcolata prima.

3.4 Sviluppo di Area Learning

Si analizza ora l'implementazione di Area Learning nell'applicazione HelloTango. L'interfaccia dell'app gestisce la funzionalità in questo modo: innanzitutto nella parte destra è presente un pulsante a forma di switch che permette di abilitare o disabilitare Area Learning in qualunque momento. Una volta abilitata, basta muoversi nell'ambiente che subito il dispositivo in automatico inizia a trovare tutti i landmarks possibili; quando si decide di terminare la sessione, basta toccare il pulsante "Salva Area" e in questo modo viene generato il file ADF che verrà salvato nella memoria del dispositivo. Per cancellare tutti i file di descrizione ambientale memorizzati, si preme il pulsante "Cancella Aree".

Quando un file ADF è presente, è possibile provare a localizzarsi: l'applicazione carica, in automatico all'avvio, il file ADF più recente, se disponibile. Una volta caricato, l'app informerà l'utente se la localizzazione è stata raggiunta tramite un'etichetta posta in basso dell'interfaccia, con la frase "Localizzato"; altrimenti, se il dispositivo non riesce ancora o si è disorientato per qualche ragione, la scritta apparirà come "Non localizzato". Come primo esempio si mostra in figura 28 un tentativo di localizzazione.

Figura 28 – Esempio di localizzazione riuscita in uno spazio aperto

Questo caso mostra come il dispositivo sia riuscito a localizzarsi in un ambiente all'aperto. Si è potuto osservare comunque che il sistema Tango impiega molto meno tempo a localizzarsi in ambienti chiusi (tipicamente pochi secondi) che in quelli aperti (media si va dai dieci ai trenta secondi); questo molto probabilmente è dato dal fatto che in un ambiente ristretto il sistema riesce a catturare molti più landmarks.

Un'applicazione importante di Area Learning, precedentemente introdotta, è quella di poter ricostruire la scena virtuale creata nell'ambiente una volta che si torna in quell'area e ci si localizza. L'app HelloTango dimostra chiaramente questa importante funzionalità, andando ad inserire nello spazio fisico degli oggetti 3D a forma di cubo: questi fanno parte di alcune primitive grafiche di default presenti nella libreria Rajawali, mentre l'inserimento di questi oggetti nella scena è stata resa possibile semplicemente toccando sullo schermo dello smartphone il punto di collocamento desiderato, il tutto sfruttando la Depth Percetion per calcolare il piano d'appoggio della superficie. Tramite il pulsante "Cancella Oggetti", invece, si eliminano dalla scena tutti gli oggetti finora inseriti, in maniera tale da ricominciare da capo con nuovi esperimenti. Questo modo di procedere consente anche, attraverso gli esempi che seguiranno, di avere una visione diretta di quanto questa tecnologia sia efficace nel riconoscere uno spazio già visitato e quindi nel percepire l'ambiente circostante.

Questi esempi mostrano alcune prove effettuate all'interno di una stanza, nella quale vi sono degli elementi chiave come dei quadri, un tavolo, un appendiabiti e un'anfora come porta ombrelli. La prima prova è visibile in figura 29: inizialmente si è attivato Area Learning per salvare i dettagli dello spazio, poi si è provveduto a localizzare il dispositivo all'interno di questo ambiente, operazione che ha impiegato pochi secondi. Una volta riconosciuta l'area si sono collocati due cubi: il primo, contrassegnato dal nome "Cubo 1" nelle sue sei facce, è stato posto sulla sinistra nei pressi di un uscio, mentre il secondo, con etichetta "Cubo 2", è stato piazzato sulla destra accanto ad una porta.

Una volta creata questa scena virtuale, l'ambiente è stato cambiato togliendo alcuni elementi che lo caratterizzavano e si è provato di nuovo la localizzazione; il risultato è mostrato in figura 30. Si osserva come il dispositivo Tango sia riuscito a localizzarsi nonostante l'ambiente avesse perso alcuni suoi dettagli descrittivi: avendo riconosciuto l'area dove era stato prima, gli oggetti virtuali sono stati automaticamente piazzati esattamente negli stessi punti.

Infine la figura 31 mostra lo spazio questa volta spogliato di tutti i suoi elementi caratteristici, risultando in una stanza semi vuota. In questo caso le differenze rispetto al caso iniziale sono molte ed infatti il dispositivo non è riuscito a localizzarsi; questo comporta che gli oggetti virtuali non si ricollocano più nella posizione originaria, ma in un'altra sbagliata.

Figura 29 – Area Learning in un ambiente indoor con dettagli al completo

Figura 30 – Area Learning in un ambiente indoor con alcuni dettagli mancanti

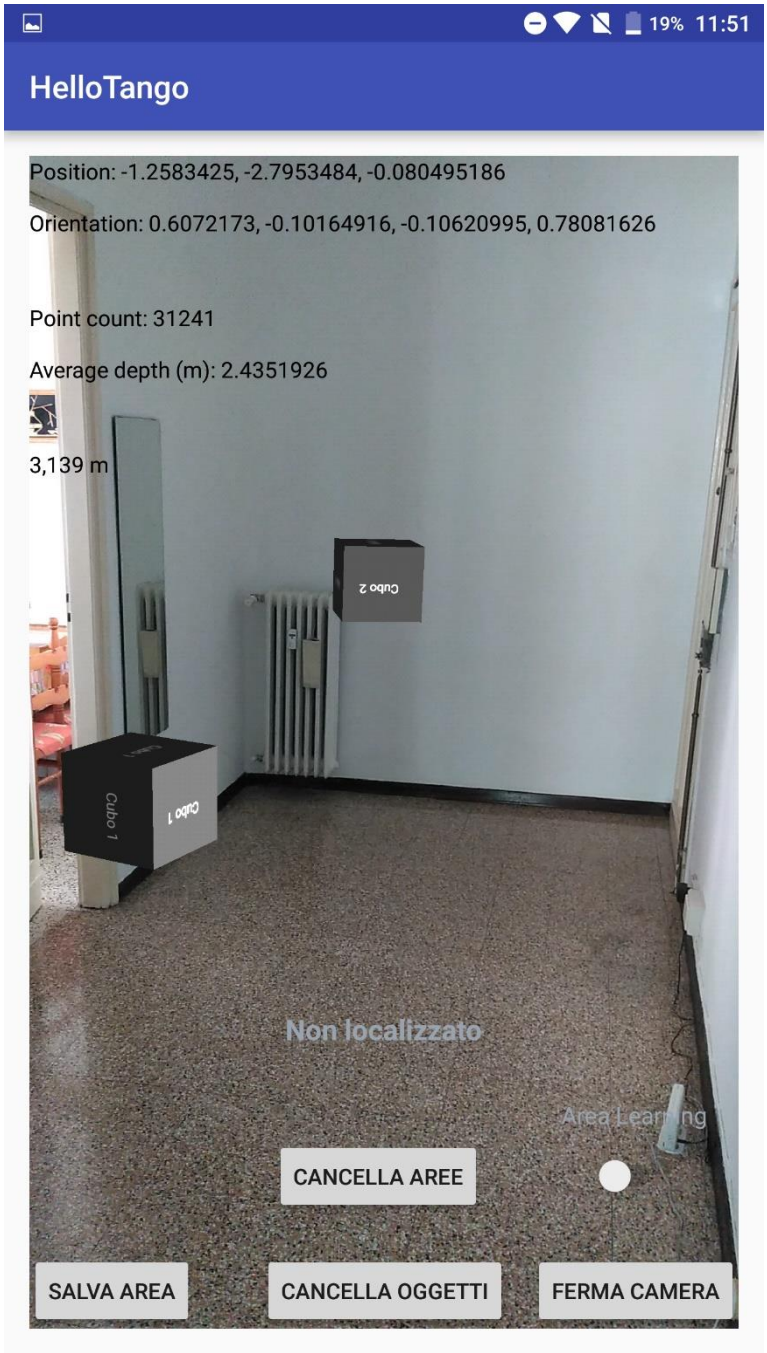

Figura 31 – Area Learning in un ambiente indoor con tutti i dettagli mancanti

L'aspetto tecnico che permette ad Area Learning di ricostruire una scena virtuale è il salvataggio nel file ADF della posizione esatta da dove è iniziata la sessione di apprendimento. Una volta che l'utente torna in un ambiente già visitato e il file ADF relativo ad esso è stato caricato, se il sistema si localizza, da quel momento in avanti si possono ottenere dati relativi al Pose del dispositivo *anche* rispetto all'origine salvata all'interno del file di descrizione dell'area, oltre a quella corrispondente all'inizializzazione del service Tango vista finora. È proprio grazie a questo nuovo sistema di coordinate che gli oggetti virtuali si ricollocano esattamente negli stessi posti di prima. Per descrivere meglio questo aspetto si riporta un esempio di localizzazione effettuata all'aperto.

La figura 32 mostra la traiettoria descritta per arrivare al punto dove il dispositivo da lì a poco si sarà localizzato.

La figura 33 descrive la presenza di due traiettorie nella scena: quella vista fino ad' ora è di colore rosso ed è la traiettoria descritta con il sistema di coordinate che prevede come origine il punto di inizializzazione del service Tango. La traiettoria blu, invece, è rappresentata sulla scena solo a seguito di una localizzazione: essa infatti è disegnata con il sistema di coordinate che prevede come origine quella salvata nel file ADF. La presenza in simultanea delle due traiettorie permette dunque di cogliere la differenza dei due sistemi di coordinate usati per tracciarle: dato che, una volta che si è localizzati, il punto di vista all'interno della scena (la camera virtuale in OpenGL) cambia in relazione al nuovo sistema di coordinate (questo permette di disegnare correttamente la traiettoria blu), comporta un disegno della traiettoria rossa non più coerente con l'ambiente; infatti si può osservare come la parte concava descritta da essa sia ora rivolta dall'altra parte della strada, a differenza di quanto si può osservare nella figura 32.

Si può notare ancora più marcatamente questa differenza tramite la figura 34: la traiettoria blu mostra parte del percorso (visibile anche dalla figura 33) effettuato dal punto di avvenuta localizzazione alla posizione, spostata di qualche metro, dove la traiettoria rossa iniziava a disegnarsi (inizio del service Tango, e quindi di Motion Tracking). Si può chiaramente osservare come la linea rossa sia in realtà disegnata sul cigno opposto della strada, dove il dispositivo non si è mai diretto. I dati di Tango confermano tutto questo: infatti il Pose che l'applicazione mostra sullo schermo è sempre riferito al sistema di coordinate avente come origine il punto di avvio del service, e mostra di essere spostati di circa quattro metri a sinistra ed un metro indietro rispetto al punto esatto dell'origine.

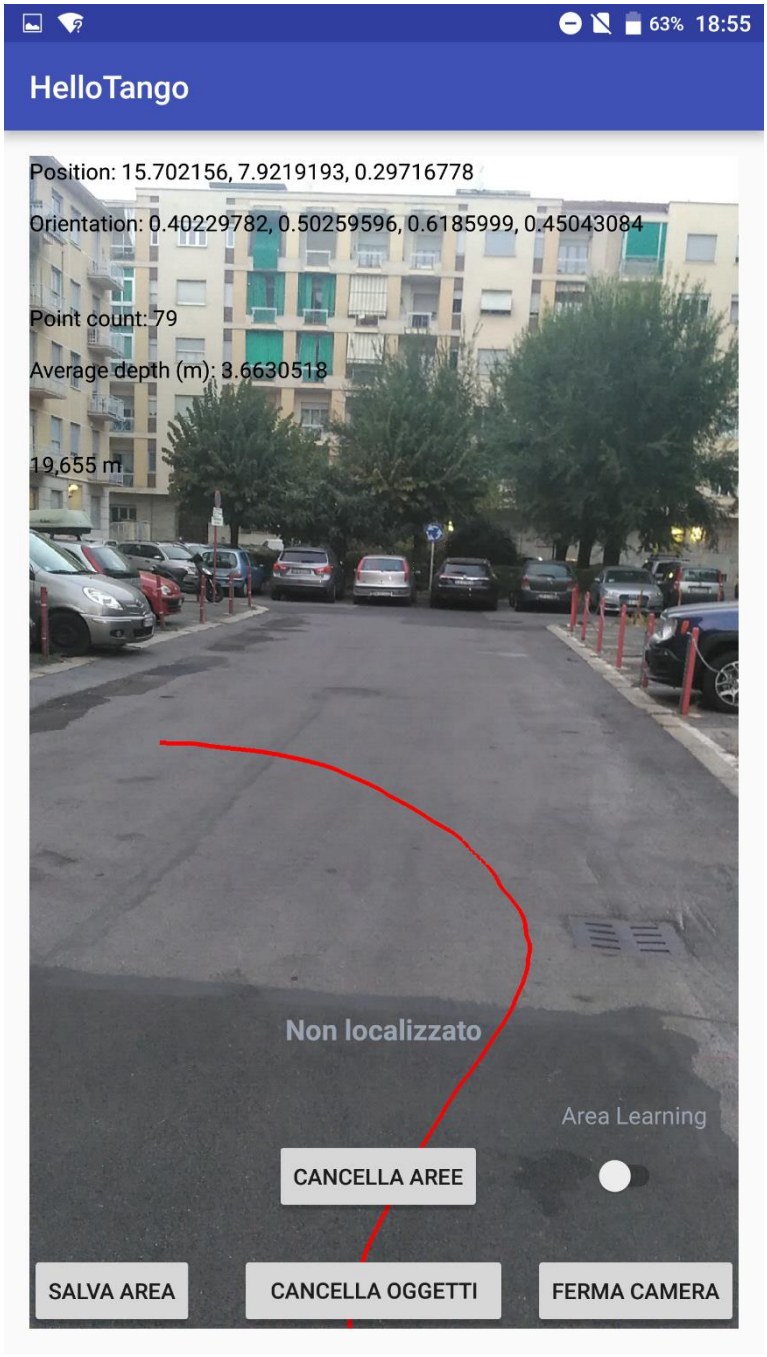

Figura 32 – Traiettoria tracciata prima di una successiva localizzazione all'aperto

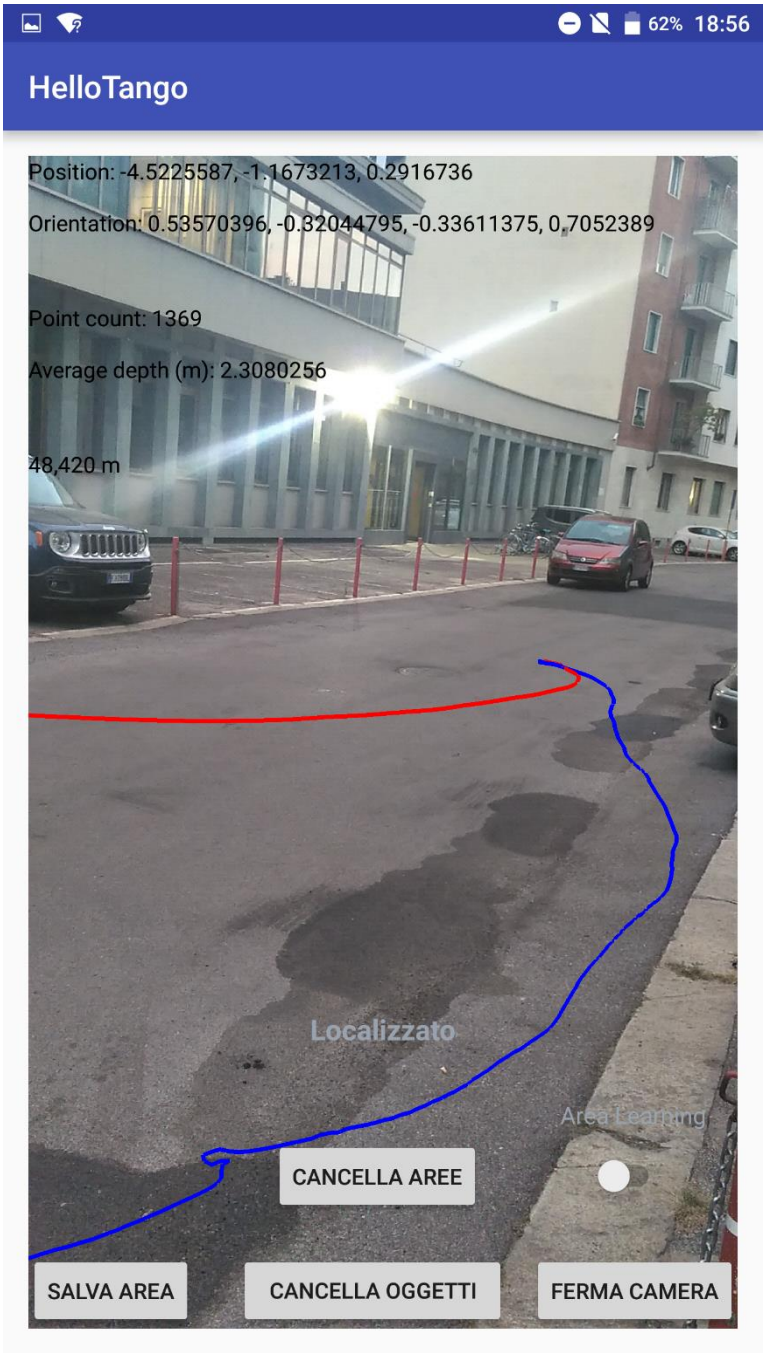

Figura 33 – Percorso della traiettoria rossa cambiata a seguito di una localizzazione

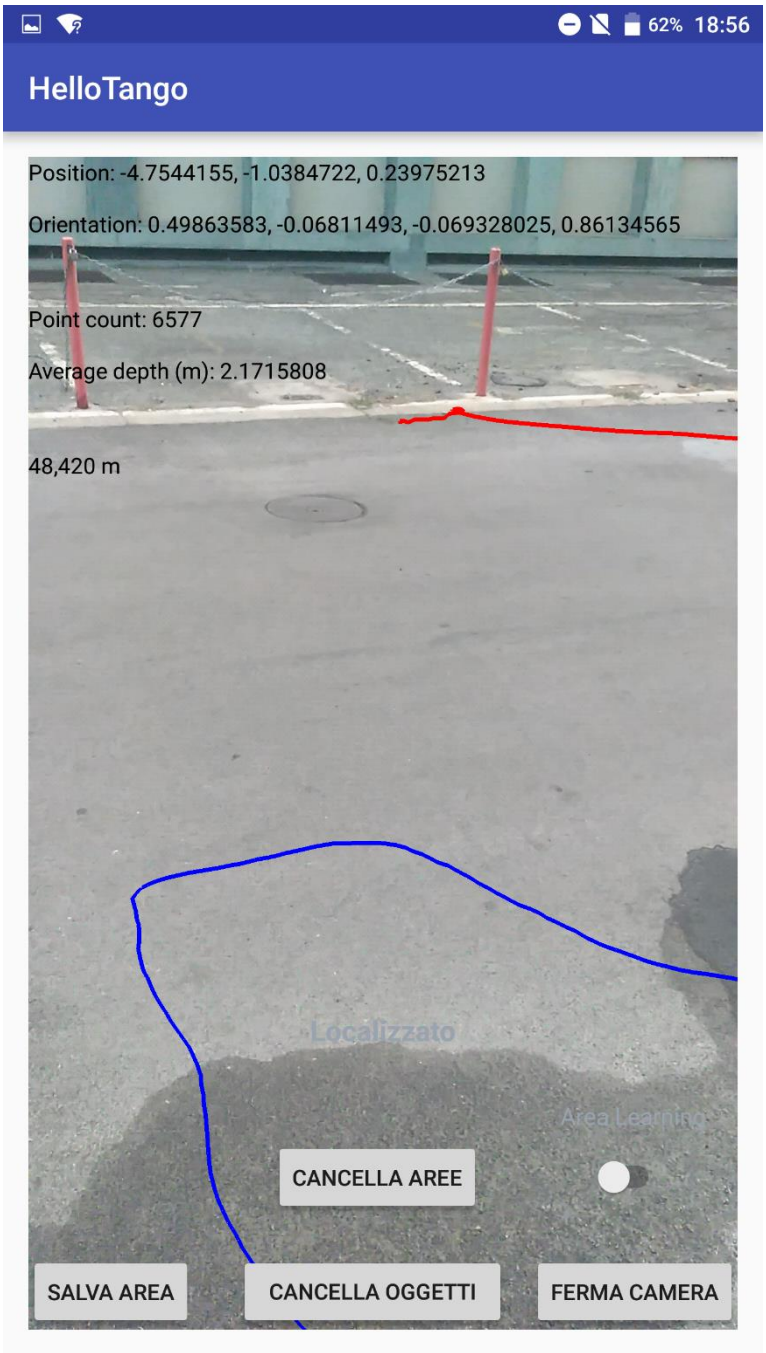

Figura 34 – Percorso della traiettoria rossa cambiata a seguito di una localizzazione, altra prospettiva dal punto di inizio di Motion Tracking

Tutto ciò è stato possibile implementarlo all'interno dell'applicazione HelloTango tramite i concetti di *base frame of reference* e *target frame of reference*, già introdotti in precedenza per parlare del concetto di Pose: la diversa scelta del sistema di coordinate, infatti, presuppone una determinata scelta di questi due componenti. Nello specifico bisogna diversificare due casi: il primo riguarda la fase di localizzazione e di il secondo quella seguente l'eventuale esito positivo.

La fase di localizzazione prevede di assegnare come base frame e target frame rispettivamente la coppia `COORDINATE_FRAME_AREA_DESCRIPTION` e `COORDINATE_FRAME_START_OF_SERVICE`: la seconda è già stata spiegata essere il punto d'origine dell'avvio del TangoService, mentre la prima rappresenta l'origine contenuta nel file ADF caricato. Se il sistema Tango fornisce dei dati di Pose validi con questa coppia di frame significa che il dispositivo si è localizzato. Si può vedere questo procedimento nel codice che segue.

```
private void startupTango() {

    final ArrayList<TangoCoordinateFramePair> framePairs =
            new ArrayList<TangoCoordinateFramePair>();

    [...]

    framePairs.add(new TangoCoordinateFramePair(
            TangoPoseData.COORDINATE_FRAME_AREA_DESCRIPTION,
            TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE));



    tango.connectListener (framePairs, new Tango.TangoUpdateCallback() {

       @Override
       public void onPoseAvailable (final TangoPoseData pose) {

       [...]

           if(pose.baseFrame ==
                  TangoPoseData.COORDINATE_FRAME_AREA_DESCRIPTION &&
                pose.targetFrame ==
                  TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE){


               if(pose.statusCode == TangoPoseData.POSE_VALID){

                   islocalized = true;

                   runOnUiThread(new Runnable() {
                       @Override
```

```
                        public void run() {
                        localizedTextView.setText("Localizzato");
                        }
                    });

                }

                else {

                    islocalized = false;


                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            localizedTextView.setText("Non localizzato");
                        }
                    });
                }
            }

        }

        [...]

    }

}
```

Esso mostra proprio quanto detto poc'anzi, con l'assegnazione di base frame e target frame nel metodo che svolge la funzione di listener alle chiamate di callbacks invocate da Tango: se il Pose è valido si imposta l'etichetta nella schermata dell'applicazione come "Localizzato" altrimenti sarà impostata come "Non localizzato" fino ad una futura localizzazione.

Per quanto riguarda la fase successiva alla localizzazione, il dato booleano `islocalized` è utilizzato per segnalare l'esito positivo di riconoscimento ambientale, segno che da quel momento in avanti si potranno richiedere i dati di Pose con riferimento all'origine nel file ADF. Si analizza ora il codice che differenzia proprio questo fatto nel momento in cui si decide di disegnare la traiettoria percorsa.

```
if (!islocalized || (tangoConfig.getString
        (TangoConfig.KEY_STRING_AREADESCRIPTION).equals(""))){


    TangoPoseData lastFramePose = TangoSupport.getPoseAtTime(0,
            TangoPoseData.COORDINATE_FRAME_START_OF_SERVICE,
            TangoPoseData.COORDINATE_FRAME_DEVICE,
            TangoSupport.TANGO_SUPPORT_ENGINE_OPENGL,
```

```
            TangoSupport.TANGO_SUPPORT_ENGINE_OPENGL,
            displayRotation);


    if(lastFramePose.statusCode == TangoPoseData.POSE_VALID){

        renderer.updateCameraPose(lastFramePose);
        renderer.updateTrajectory(lastFramePose);

       }
}
else {


    TangoPoseData lastFramePose1 = TangoSupport.getPoseAtTime(0,
            TangoPoseData.COORDINATE_FRAME_AREA_DESCRIPTION,
            TangoPoseData.COORDINATE_FRAME_DEVICE,
            TangoSupport.TANGO_SUPPORT_ENGINE_OPENGL,
            TangoSupport.TANGO_SUPPORT_ENGINE_OPENGL,
            displayRotation);


    if(lastFramePose1.statusCode == TangoPoseData.POSE_VALID) {

        renderer.updateTrajectoryLocalized(lastFramePose1);
        renderer.updateCameraPose(lastFramePose1);
    }
}
```

La prima istruzione controlla se il dispositivo è nello stato localizzato: in questo modo, se la verifica è negativa, viene richiesto espressamente (approccio polling-based) il Pose più recente (al tempo zero) con base frame il punto d'avvio del service Tango e con le opzioni di conversione da sistema di coordinate Tango a quelle di OpenGL, già viste nel caso della nuvola di punti. Infine se il Pose è valido si chiamano i metodi del render grafico che aggiornano i dati per disegnare la traiettoria, che quindi sarà rossa, e posizionare correttamente il punto di vista (camera di OpenGL) nella scena virtuale. Da notare che questo codice viene eseguito anche se la variabile `islocalized` è vera, questo perché è possibile si verifichi il caso in cui sia iniziata una nuova sessione di Learning, quindi la funzionalità di Area Learning è stata attivata ma non è presente nessun ADF: se succede questo, il sistema Tango si comporta come se una localizzazione fosse riuscita, in quanto la coppia base e target frame, rispettivamente "Area Description" e "Start of Service", fornisce dati validi. È stato quindi aggiunto un ulteriore controllo sul fatto che un file ADF sia stato caricato o meno: se non è presente, allora la localizzazione non è stata davvero effettuata, ma frutto dell'inizio di una nuova sessione, quindi si considera il caso come non localizzato.

Se viceversa la localizzazione è riuscita, il codice è simmetrico al caso precedente, ma ora si richiede il Pose aggiornato con il sistema di riferimento che ha come origine quella salvata all'interno del file ADF caricato e, una volta validati i dati, si chiamano i metodi del render per disegnare la traiettoria blu e posizionare correttamente la camera.

Terminata questa parte di approfondimento ed implementazione della tecnologia Tango all'interno di un'applicazione Android, si passa ora a trattare la seconda parte di questo lavoro di Tesi, che ha riguardato lo studio e lo sviluppo di una soluzione tecnologica basata sullo streaming real-time da una sorgente ad un gruppo generico di destinatari. Come accennato, la sorgente sarà sia il dispositivo Tango che una classica webcam gestita da una scheda elettronica appositamente programmata, il Raspberry Pi. Inizialmente si tratteranno i dettagli implementativi di quest'ultimo caso, ed infine si andrà a descrivere l'uso di Tango in questo scenario.

CAPITOLO 4

LIVE STREAMING CON RASPBERRY PI

4.1 Architettura generale

Questo capitolo affronta lo sviluppo dello streaming dati eseguito in tempo reale usando come sorgente una tradizionale webcam, nello specifico la PSEye (Playstation Eye, figura 35), sviluppata dalla Sony nel 2007 come periferica per la nota console di gioco Playstation 3.

Figura 35 – Webcam "Playstation Eye" utilizzata per il progetto di streaming

La fotocamera possiede una risoluzione di 640*480 pixel ed è capace di catturare video con una frequenta massima di 60 frame al secondo. Come detto, questa sorgente è stata gestita dalla scheda elettronica Raspberry Pi (figura 36). Essa fa parte della categoria cosiddetta dei "single-board computer" ovvero un vero e proprio calcolatore elettronico implementato su di un singolo circuito stampato, ed è sviluppato in Gran Bretagna dalla Raspberry Pi Foundation.

Con le dovute limitazioni dovute alla potenza di calcolo, il Raspberry Pi possiede infatti tutte le componenti principali che caratterizzano un computer tradizionale: CPU, GPU, periferiche di input/output.

Figura 36 – Scheda elettronica Raspberry Pi 3 model B

Il primo modello è stato rilasciato nel 2012 e fino ad oggi sono nate ben otto versioni. Il successo di vendita è dovuto sicuramente dall'economicità di questo prodotto, unito dalla sua estrema maneggevolezza (può benissimo risiedere nel palmo di una mano) e dalle sue più che discrete caratteristiche tecniche che permettono a sviluppatori ed appassionati di cimentarsi in svariati progetti, anche complessi.

Il modello usato per il lavoro di Tesi è l'ultimo immesso nel mercato in ordine cronologico, conosciuto con il nome di Raspberry Pi 3 model B; la sua scheda tecnica è la seguente, così come riportata dal sito web ufficiale della compagnia[6]:

- Soc Broadcom BCM2837
- CPU ARM Quad Core a 1,2 GHz, 64 bit
- GPU Broadcom VideoCore IV
- 1GB RAM
- LAN Wireless 802.11 b/g/n
- Bluetooth 4.1 a risparmio energetico (BLE)
- 40 pin GPIO
- 4 porte USB

[6] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

- Connettore audio stereo e video composito
- Uscita HDMI
- Connettore CSI per Raspberry Pi camera
- Connettore per display DSI
- Porta micro SD per sistema operativo e salvataggio dati
- Alimentatore micro USB 2,5 A
- Porta Ethernet 10/100 Mbit/s

Si può constatare come la scheda elettronica di per sé abbia tutte le più importanti componenti di un classico calcolatore elettronico, nonostante il prezzo sia molto competitivo, attualmente circa 40 €.

Il Raspberry è stato usato installando una distribuzione Linux derivante da Debian, chiamata Raspbian, versione più leggera appositamente studiata per funzionare su processori ARM. Questo sistema operativo possiede già nel suo kernel i driver funzionanti per diverse webcam, tra cui la PSEye utilizzata: in questo modo basta collegarla ad una delle quattro porte USB della scheda per essere subito riconosciuta.

Per interfacciarsi con la scheda e dunque poter installare ed implementare il software necessario per lo streaming si è provveduto a gestire il Raspberry da remoto tramite un comune PC da dove impartire i comandi mediante tastiera e mouse. Questa soluzione è molto comune in ambito informatico ed è chiamata Virtual Network Computing (VNC). In sostanza si tratta di installare un software sia sulla macchina da controllare che su quelle macchine che andranno poi a controllarla. Nel progetto di questa Tesi si è utilizzato il software gratuito messo a disposizione dalla compagnia RealVNC[7]: essa fornisce il programma VNC Connect, che funge da server e va installato nella macchina da controllare e VNC Viewer che è il software da installare sulle macchine clients, quelle cioè che impartiscono i comandi a distanza. La comunicazione tra queste due componenti avviene in modalità wireless protetta dalla crittografia end-to-end AES-CGM con chiave a 128 bit ed inoltre è richiesta una password dal server per autenticare i clients. Si è preferita questa soluzione di controllo remoto rispetto ad altre basate su emulazione di terminale (per esempio il client SSH PuTTY) in quanto permette ai dispositivi clients di gestire l'interfaccia grafica del sistema operativo e quindi di potersi muovere all'interno di esso come in un normale computer: questo ha permesso una migliore produttività e praticità quando si è dovuto implementare e modificare il framework GStreamer.

[7] https://www.realvnc.com

Si passa ora a descrivere l'architettura del progetto di streaming multimediale, rappresentata sinteticamente in figura 37.

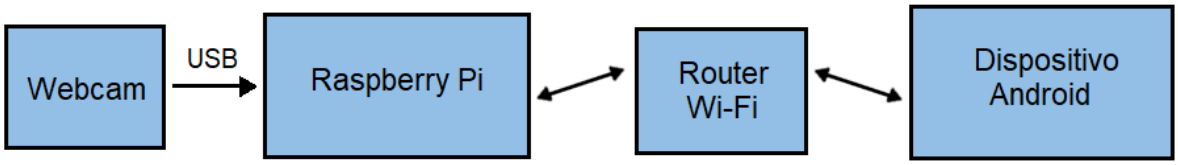

Figura 37 – Architettura utilizzata per lo streaming dati tramite Raspberry

Come detto, il Raspberry si interfaccia alla webcam tramite il collegamento USB. Il software apposito installato nella scheda elettronica cattura ed elabora ogni singolo fotogramma che la webcam produce e, tramite il modulo WI-FI integrato in essa, invia questi dati ad un router. Questo apparecchio, una volta ricevuti i dati, ha il compito di inviarli a tutti i destinatari connessi ad esso, sempre tramite comunicazione wireless. I dispositivi che ricevono questi dati sono dei comuni smartphone o tablet con a bordo il sistema operativo Android ed hanno installata un'applicazione appositamente sviluppata per poter rielaborare i dati ricevuti ed infine mostrare i vari fotogrammi sul loro display.

Come era stato anticipato, questa soluzione è stata pensata per essere utilizzata in una realtà ristretta, ovvero all'interno di una stessa rete locale dove sono presenti tutti i componenti sopra descritti. Il router, quindi, è un dispositivo non pensato per essere connesso alla rete globale Internet, ma è invece adoperato solo per svolgere la funzione di smistamento dei pacchetti dalla sorgente direttamente alle destinazioni, senza passare, come detto, tramite dei server esterni: questo permette di alleggerire il carico di lavoro e di avere delle prestazioni più reattive, indispensabili in un contesto real-time di cui questo approccio allo streaming fa parte.

Un aspetto fondamentale nello sviluppo è stato la scelta del protocollo di trasporto da utilizzare. In informatica, nell'ambito delle reti di calcolatori, questo aspetto permette di realizzare un canale logico di comunicazione, eventualmente affidabile, ed end-to-end, cioè direttamente tra sorgente e destinatari [6]. Esso si appoggia al sottostante protocollo di rete, il quale fornisce la funzionalità di indirizzamento ed instradamento

delle unità informative, denominate *pacchetti*, all'interno di una rete di comunicazione, scegliendo il percorso più appropriato tra mittente e destinatario. Nel corso del tempo, questi concetti hanno assunto una grande importanza per la creazione e lo sviluppo della futura rete Internet. In sostanza si sono affermati due differenti protocolli di trasporto tramite i quali la rete Internet, per esempio, si è potuta evolvere, denominati TCP ed UDP.

Il TCP (Transmission Control Protocol) [7] nacque negli anni '70 da un gruppo di ricercatori del Dipartimento di Difesa Statunitense. Esso è un protocollo di trasporto cosiddetto affidabile, in quanto i segmenti (così vengono chiamate le unità informative) sono garantiti giungere a destinazione e nell'esatto ordine in cui sono stati spediti; questa caratteristica è implementata tramite delle tecniche sofisticate di acknowledgment e ritrasmissione su timeout (se si manifestano eventuali perdite di pacchetti). Inoltre è un protocollo cosiddetto orientato alla connessione: per poter permettere lo scambio dati, si deve prima provvedere a stabilire una connessione tra mittente e destinatario, la quale rimane attiva anche in assenza di dati e viene terminata esplicitamente da una delle due parti, rilasciando così le risorse occupate. In questo modo lo scambio dati è logicamente rappresentato a livello applicativo da un flusso di byte bidirezionale, dove essi sono garantiti essere trasportati in ordine e consegnati a destinazione; in realtà questo flusso viene frazionato in blocchi che vanno a formare il cosiddetto *segmento TCP* e poi successivamente inviato.

Il protocollo UDP (User Datagram Protocol) [8] è nato nel 1980 grazie al lavoro dell'informatico statunitense David Patrick Reed. A differenza del TCP, è un protocollo "connectionless", ovvero non prevede nessuna creazione di connessione tra le due parti. Inoltre il servizio offerto da questo protocollo è inaffidabile, infatti non vengono attuate misure per assicurare che i dati arrivino a destinazione, né se siano in ordine. Sostanzialmente è un protocollo molto semplice, che occupa poche risorse ed è questo il suo punto di forza. I dati, a differenza del TCP, sono ben definiti all'interno di un'unità informativa, chiamata *datagramma*: in questo modo la comunicazione non si basa più sul concetto logico di flusso, ma è caratterizzata dall'invio di singoli datagrammi, ognuno con una dimensione massima di 65535 bytes; se esso non riuscisse a raggiungere la destinazione, l'intero carico informativo presente (payload) andrebbe perso, in quanto non sono previste ritrasmissioni.

I due protocolli di trasporto sono dunque estremamente differenti tra loro e per questo vengono tipicamente utilizzati in contesti diversi: TCP viene impiegato quando c'è necessità di trasmettere in modo affidabile tutti i dati, come ad esempio il trasferimento di un file, l'invio della posta elettronica. UDP, invece, viene utilizzato in quei contesti dove l'affidabilità non gioca un ruolo chiave nella trasmissione, come nel caso della telefonia via Internet (VOIP), nella trasmissione di contenuti multimediali. In un

contesto di streaming multimediale senza interruzioni, con la sorgente che trasmette in tempo reale, l'uso del protocollo TCP richiederebbe dei ritardi nel caso in cui le unità informative vadano perse, questo perché il sistema invierebbe di nuovo quel segmento, che nel frattempo è diventato inutile perché la trasmissione prosegue nel tempo, inviando datagrammi più recenti. Durante lo sviluppo del progetto è stato verificato concretamente questo aspetto, andando ad implementare una soluzione basata su TCP ed un'applicazione Android in grado di leggere e decodificare i dati nel formato JPEG, reperibile gratuitamente in rete[8]. Questa implementazione è risultata aspettatamente insoddisfacente, in quanto il flusso video visualizzato a destinazione risultava essere in ritardo già dopo poche decine di secondi.

Per questa ragione si è deciso di implementare una soluzione basata sul protocollo UDP: così facendo, la comunicazione non richiede nessuna ritrasmissione, la quale non rappresenta teoricamente un problema se la perdita di datagrammi avviene in maniera sporadica, proprio come nel caso di questo progetto dove il contesto di utilizzo è una singola rete locale, quindi un'infrastruttura avente un traffico dati contenuto e quindi una probabilità di perdita di dati bassa.

Nel panorama odierno caratterizzato dall'esistenza pervasiva di innumerevoli applicazioni e piattaforme di trasmissioni multimediali in Internet, sono presenti diversi protocolli applicativi che, appoggiandosi su quelli di trasporto, gestiscono questa specifica comunicazione. Uno dei più diffusi è sicuramente il protocollo RTP (Real Time Protocol) [9]. Esso è nato per gestire applicazioni real-time e si appoggia su UDP per consegnare i dati a destinazione. È un protocollo pensato per applicazioni distribuite sulla rete Internet e quindi molto complesse. In questo lavoro di Tesi si è implementata una soluzione più snella escludendo l'uso di questo protocollo applicativo, in quanto la situazione tipica di utilizzo è molto più semplice in termini di infrastruttura e numero di utenti rispetto all'intera rete Internet. Ciononostante il progetto di Tesi ha richiesto, come accennato nel capitolo introduttivo, la progettazione di un protocollo di comunicazione ad hoc per permettere la corretta fruizione dei contenuti multimediali, con alcune caratteristiche teoriche simili a quelle presenti nel protocollo RTP che verranno a breve descritte.

[8] https://bitbucket.org/coisme/simplemjpegview_gst/wiki/Home

4.2 La libreria multimediale GStreamer

Prima di passare alla descrizione di questo protocollo, si analizza ora il software utilizzato dalla scheda elettronica per permettere di creare il flusso di comunicazione; come anticipato, è stata utilizzato il framework *GStreamer*[9]. Questa piattaforma software permette la creazione di applicazioni multimediali di varia natura, come la semplice riproduzione in playback di filmati o file audio, la registrazione audio/video da sorgenti di varia natura, lo streaming audio/video anche real-time e molto altro. Essa supporta una grande varietà di formati file, protocolli e codec multimediali, il che lo rende uno strumento molto potente, oltre al fatto di essere open source, quindi modificabile secondo proprie necessità, e multipiattaforma.

La caratteristica fondamentale che rende questo framework così interessante è la sua modularità: esso, infatti, è basato sull'esistenza di svariati moduli, chiamati *plugins*, ciascuno dei quali apporta determinate funzionalità e può essere usato come una black box, cioè senza la necessità di dover conoscere la sua struttura interna per poterlo implementare in un progetto, ma solo le funzionalità che offre. All'interno del framework esiste una sommaria divisione dei plugins in tre macroscopici gruppi: good, bad, ugly:

- Il gruppo *good* contiene il set di plugins considerati di alta qualità, quindi in grado di garantire la massima affidabilità in termini di prestazioni, grazie anche alla certezza di essere passati sotto varie sessioni di testing e debugging.

- Il gruppo *bad* unisce insieme plugins che possono raggiungere una buona qualità di realizzazione ma sono deficitari in qualcosa, come una mancanza di una approfondita documentazione, revisione di codice o un utilizzo intenso.

- Il gruppo *ugly*, infine, racchiude quei plugins che sono etichettati essere di bassa qualità, e che quindi possono causare dei problemi di distribuzione.

[9] https://gstreamer.freedesktop.org/

Questi plugins possono naturalmente essere modificati per estendere le loro caratteristiche o possono esserne creati degli altri. Inoltre, è possibile caricarli dinamicamente durante l'esecuzione del codice, il che li rende delle vere e proprie librerie dinamiche.

Questa architettura modulare basata su plugins viene implementata concretamente in GStreamer tramite il concetto di *pipeline* [10]. Ogni progetto realizzato con questo framework presuppone, infatti, la realizzazione di almeno una pipeline, ovvero un insieme di elementi collegati fra loro uno dopo l'altro. Gli elementi sono formati da tre tipi: sorgente (*source*), filtro (*filter*) e pozzo (*sink*)

- L'elemento *sorgente* ha il compito di prelevare i dati da una sorgente esterna (quale ad esempio un generico file, una periferica), che saranno in seguito utilizzati all'interno della pipeline dagli altri elementi.

- L'elemento *filtro* si occupa della elaborazione dei dati passati dall'elemento precedente nella catena; accetta quindi dei dati in input e produce dei dati in output, utilizzabili dagli elementi successivi della pipeline. Possono quindi essere usati per comunicare con altri elementi ma non con l'esterno, come la sorgente.

- L'elemento *pozzo* è quello finale all'interno della pipeline ed ha il compito di fornire verso l'esterno i dati avuti dall'elemento precedente; a differenza della sorgente, non può produrre dati.

Ogni elemento esistente nel framework GStreamer è contenuto in un certo plugin. Un semplice esempio per capire meglio questa struttura è riportato in figura 38.

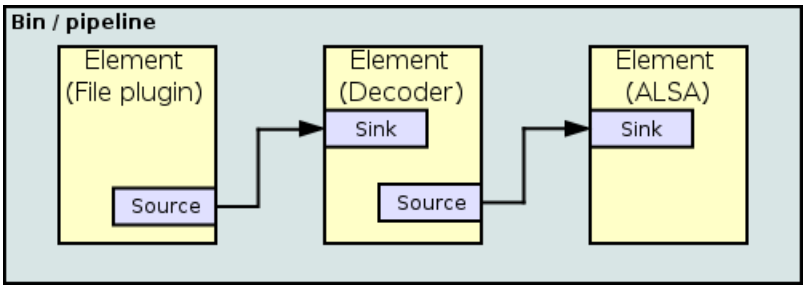

Figura 38 – Esempio grafico di una pipeline in GStreamer

Essa rappresenta la pipeline di un semplice lettore mp3. Come si può osservare, il primo elemento della catena, la sorgente, ha il compito di leggere il file audio presente nell'hard disk; esso è formato, dunque, da un solo "punto di uscita" ("Source" nella figura) chiamato in GStreamer con il termine di *pad*. In generale esistono due tipi di pad, cosiddetti *source pad* e *sink pad*, definiti dal punto di vista interno dell'elemento: il primo è il punto dove l'elemento genera i dati da spedire, mentre il secondo rappresenta il punto dove i dati vengono ricevuti in qualche modo. Nell'esempio che si sta descrivendo, esso è un source pad, infatti i dati che la sorgente acquisisce vengono, per mezzo di questo pad, trasferiti all'elemento successivo che è il filtro. Esso è caratterizzato infatti da due pad, uno di ingresso (sink pad) e l'altro di uscita (source pad); il suo scopo in questo contesto è la decodifica del formato mp3. Una volta compiuta questa operazione, i dati passano per mezzo del suo source pad verso l'elemento finale, il pozzo. Quest'ultimo, infine, ha il compito di inviare i dati ricevuti alla scheda audio per la successiva riproduzione.

La struttura di GStreamer, quindi, è molto flessibile, permettendo infatti l'utilizzo di una grande quantità di pugins già disponibili e di interfacciarli tra di loro in svariati modi per poter creare delle soluzioni multimediali di ogni genere. Questo framework, inoltre, risolve un problema classico che afferisce i plugins, ovvero l'incompatibilità: infatti è molto comune trovare plugins che sono compatibili solo con l'applicazione per la quale è stato sviluppato. GStreamer, invece, garantisce che un'applicazione che gestisce plugins tramite la sua struttura sarà compatibile con tutti gli altri plugins e applicazioni che a loro volta sono state sviluppate tramite questo framework multimediale.

4.2.1 La pipeline GStreamer sviluppata

Nel progetto di Tesi l'utilizzo di questo framework è stato fondamentale, permettendo di creare la sessione di streaming video in tempo reale. Tutto ciò è stato possibile progettando una specifica pipeline ed eseguendola da riga di comando tramite il terminale Linux. La sua struttura è visibile in figura 39.

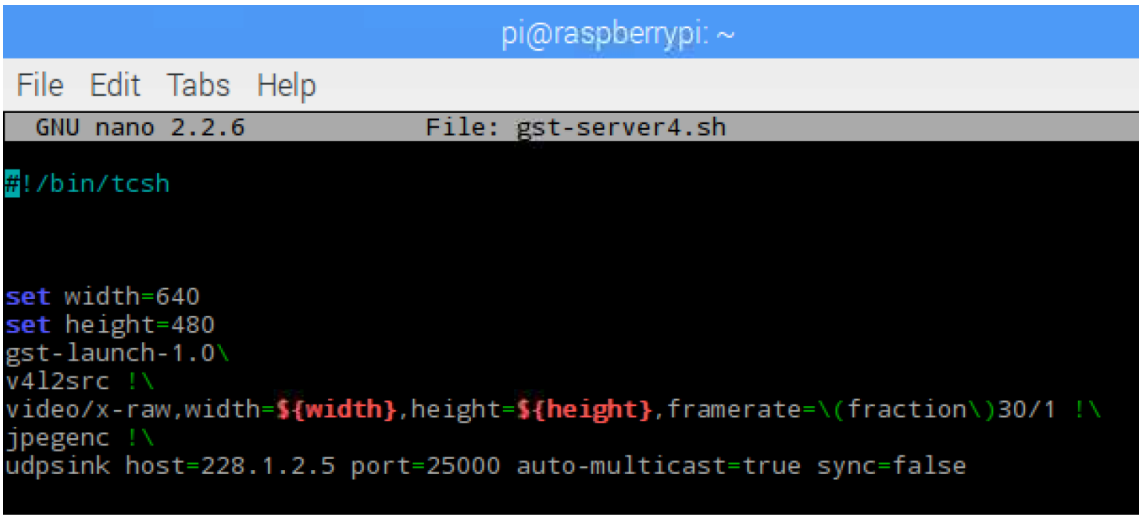

Figura 39 – Pipeline utilizzata in GStreamer per l'invio dei dati in streaming tramite Raspberry

La pipeline è stata racchiusa all'interno di uno script e successivamente eseguito. Si può notare come essa sia composta da vari elementi separati dal comando "!\" e che alcuni di essi siano costituiti da diverse proprietà.

Da questa struttura si possono descrivere i dettagli di realizzazione del progetto di streaming. Innanzitutto sono state create delle variabili che rappresentano la risoluzione delle immagini che sono acquisite dalla webcam, in questo caso quella massima di 640*480 pixel. La parte di codice successiva è il comando `gst-launch-1.0`: esso permette la costruzione e l'esecuzione di pipeline da linea di comando; il numero presente indica che si è utilizzata la versione 1.x di GStreamer, precisamente la 1.4.4, versione stabile del 2014 e presente come pacchetto Debian da scaricare direttamente.

Successivamente vi è il primo elemento della catena, ovvero la sorgente: `v4l2src`. Essa in GStreamer ha il compito di acquisire immagini da un qualunque device che supporta l'API Video4Linux2 (abbreviata V4L2) presente in Linux, ovvero un framework che permette in generale la cattura di immagini, l'acquisizione di filmati e la registrazione di audio da vari dispositivi compatibili, come webcam USB o schede TV. In questo caso l'elemento acquisisce i fotogrammi dalla webcam PSEye tramite connessione USB, la quale implementa in modo nativo i driver V4L2 e quindi pienamente funzionante fin dalla sua prima connessione alla scheda elettronica.

I dati che vengono catturati dalla webcam sono dei fotogrammi non compressi né codificati, ovvero dei dati grezzi ("raw" in inglese) così come il sensore li fornisce e questo viene compiuto tramite l'elemento "video/x-raw", il quale ha assegnato diverse proprietà come la risoluzione della cattura (640*480) e la sua frequenza: in questo caso è stato impostato il valore di 30, anche se la fotocamera ha la capacità di fornire fino a

60 frame al secondo, questo perché il framework non riesce a negoziare con la sorgente una frequenza più alta.

Questo elemento rappresenta il source pad della sorgente: i dati che esso rappresenta, infatti, sono quelli che vengono inviati all'elemento successivo, il filtro, chiamato `jpegenc`. Il suo nome sottintende che in questo progetto ciascun fotogramma viene codificato nel formato JPEG: in questo modo ogni immagine viene compressa, operazione indispensabile per poter aumentare la larghezza di banda dello streaming e quindi il numero di fotogrammi per secondo trasmissibili. Questo modo di procedere, in realtà, crea una basilare implementazione del noto codec video *Motion JPEG* (MJPEG).

Una volta compiuta questa operazione, i dati vengono passati all'ultimo elemento della pipeline, il pozzo, chiamato `udpsink`: anche qua dal nome è chiaro come questo componente della catena abbia il compito di trasmettere le immagini codificate in JPEG nella rete per mezzo di datagrammi UDP, appena sono rese disponibili dal filtro. Questo elemento ha specificato nelle sue proprietà l'indirizzo IP di destinazione e la porta logica ed in questo caso è stato scelto l'indirizzo "228.1.2.5".

Questa particolare scelta suggerisce che la trasmissione dei dati avviene per mezzo della tecnica *Multicast* [11]: questo modello di comunicazione prevede che la sorgente invii i pacchetti dati a più destinatari, senza però indirizzarli singolarmente, e quindi senza duplicare le unità informative per ogni singolo nodo di destinazione, ma trasmettendole invece una sola volta; sarà infatti compito dei dispositivi di rete (i router) duplicare i dati e consegnarli alle corrette destinazioni. Nel caso di comunicazione tra nodi mediante il protocollo di rete IP, come nel caso di questo progetto di streaming, si è storicamente stabilito che solo uno specifico range di indirizzi IP debbano essere usati per le comunicazioni in Multicast: esso va dall'indirizzo 224.0.0.0 all'indirizzo 239.255.255.255 costituendo la cosiddetta classe D degli indirizzi IP [12]; quello utilizzato nella pipeline appartiene proprio a questo intervallo. Per quanto riguarda la porta logica, è stata scelta la numero 25000, valore che non appartiene al range delle porte riservate ad applicazioni specifiche assegnate su base universale [13], perciò usabile senza vincoli.

Nello specifico, perciò, il singolo datagramma UDP verrà inviato tramite il modulo WI-FI del Raspberry al router. I dispositivi che vogliono prendere parte allo streaming dovranno prima di tutto informare l'apparato di rete, tramite degli appositi messaggi basati sul protocollo IGMP (Internet Group Management Protocol), la volontà di ricevere i dati che hanno come mittente l'indirizzo Multicast "228.1.2.5": essi andranno a costituire il cosiddetto "gruppo Multicast" per quello specifico indirizzo. Una volta ricevuto il datagramma, il router lo duplicherà tante volte quanti sono i nodi che hanno fatto richiesta e invierà questi pacchetti alle rispettive destinazioni, sulla porta logica 25000.

Gli utenti destinatari, infine, avranno la possibilità di visionare il contenuto che nel tempo viene trasmesso tramite un'app Android: questa svolgerà il compito di decodificare il formato JPEG dell'immagine e di far visualizzare, infine, il suo contenuto sul display del dispositivo.

L'uso della tecnica Multicast è fondamentale in questo progetto, in quanto evita alla sorgente di duplicare i pacchetti dati e di inviarli direttamente ad ogni destinatario, risparmiando risorse, come la banda trasmissiva, e tempi di elaborazione da parte della scheda elettronica; inoltre permette di ignorare l'esatto indirizzo IP dei destinatari, in quanto è necessario solo conoscere quello del gruppo Multicast. Questa caratteristica consente alla trasmissione in streaming di poter essere fruita da un numero indefinito di utenti, nonché dinamico, perché essi possono in qualunque momento iscriversi al gruppo Multicast o lasciarsi: la gestione di tutto questo è affidata al router in modo completamente trasparente.

4.3 Lo sviluppo software: modifica al framework GStreamer

Questa soluzione così descritta, in realtà, è risultata funzionare bene solo in determinate circostanze, cioè quando i dati codificati in JPEG non superavano la dimensione di 65507 bytes. Questa, infatti, rappresenta il limite massimo che il carico utile (payload) di ogni datagramma UDP deve avere, in quanto l'intestazione (header) di ogni datagramma è stata progettata essere di 8 bytes, mentre quella del protocollo IP, cui si unisce l'header di UDP per poter trasportare l'unità informativa, è normalmente fissata a 20 bytes. Di conseguenza, un payload superiore al limite sopra menzionato comporterebbe un datagramma ad avere una dimensione teorica maggiore di 65535 bytes, superando quindi il massimo consentito dal protocollo UDP e per questo non verrebbe trasmesso, ma scartato direttamente dal sistema operativo. Nella praticità questo ha comportato che la trasmissione streaming, in determinate situazioni, si bloccasse all'improvviso e poi riprendesse anche dopo svariati secondi. Questo problema, quindi, è legato al tentativo di trasportare immagini codificate della fotocamera troppo grandi per poter essere ospitati in singoli datagrammi UDP e ciò può naturalmente succedere in moltissimi casi, caratterizzati da certe condizioni ambientali o dalla presenza di specifici pattern che vengono inquadrati dalla webcam (figura 40).

Figura 40 – Esempio di pattern che causa l'impossibilità di inviare il singolo fotogramma tramite un solo datagramma UDP

Questo fatto comporta una grossa limitazione di utilizzo dello streaming dati con gli strumenti descritti finora: il contesto di utilizzo in questo progetto di Tesi, infatti, è incentrato sull'esplorazione ambientale, quindi la webcam si troverà verosimilmente sovente ad inquadrare un'ampia varietà di caratteristiche ambientali differenti tra loro, al contrario di un suo utilizzo statico, come per esempio in un'applicazione di videosorveglianza.

Per questo motivo, come si era accennato nell'introduzione al lavoro di Tesi, è stata pensata ed implementata una soluzione ad hoc per risolvere questo tipo di problema. La procedura generale è risultata nell'andare a distribuire i dati di ogni immagine, che eccedesse la dimensione massima del singolo pacchetto, in più datagrammi UDP. In questo modo, a seconda dei casi, un singolo fotogramma è inviato dalla sorgente alle singole destinazioni sotto forma di uno o più datagrammi UDP: tutto questo consente quindi la trasmissione di immagini di qualunque dimensione, evitando ogni blocco dello streaming.

Questo modo di procedere, però, non è implementato nativamente in GStreamer: il framework consente solo l'invio di dati tramite singoli pacchetti ed informa l'utente tramite dei messaggi di Warning nel terminale l'eventualità che essi siano stati scartati. Sfruttando una delle potenzialità di questa piattaforma software, quella cioè di essere open source, si è proceduto quindi a modificare il suo codice sorgente, scritto in linguaggio C. Nello specifico, la componente di interesse in questa procedura è stata l'elemento "multiudpsink" appartenente al gruppo dei plugins cosiddetti "goods". Il suo codice sorgente, in particolare, ingloba anche le funzionalità dell'elemento udpsink, usato nella pipeline, in quanto implementa il caso generale di invio di datagrammi UDP a più indirizzi IP.

Si passa ora a descrivere in maniera più dettagliata gli aspetti principali della soluzione. Prima di tutto, essendo questo un elemento finale, il sink, ha già compreso nel suo codice un puntatore al buffer contenente tutti i dati dell'immagine da inviare, passatogli dall'elemento encoder JPEG. Agendo quindi direttamente sui dati interessati, la prima istruzione compiuta è stata controllare se la dimensione del buffer eccedesse o meno la massima consentita; se essa rientra nel limite, il codice invia i dati direttamente.

Viceversa, inizia la fase che comporta la creazione di un numero di datagrammi tale da poter trasportare tutti i dati presenti nel buffer. Su ciascuno di essi è stato inserito un header aggiuntivo, con lo scopo di aggiungere le informazioni necessarie a destinazione per poter capire per ciascun datagramma ricevuto a quale immagine appartenesse e in che ordine. Nello specifico è stata implementata la soluzione visibile in figura 41.

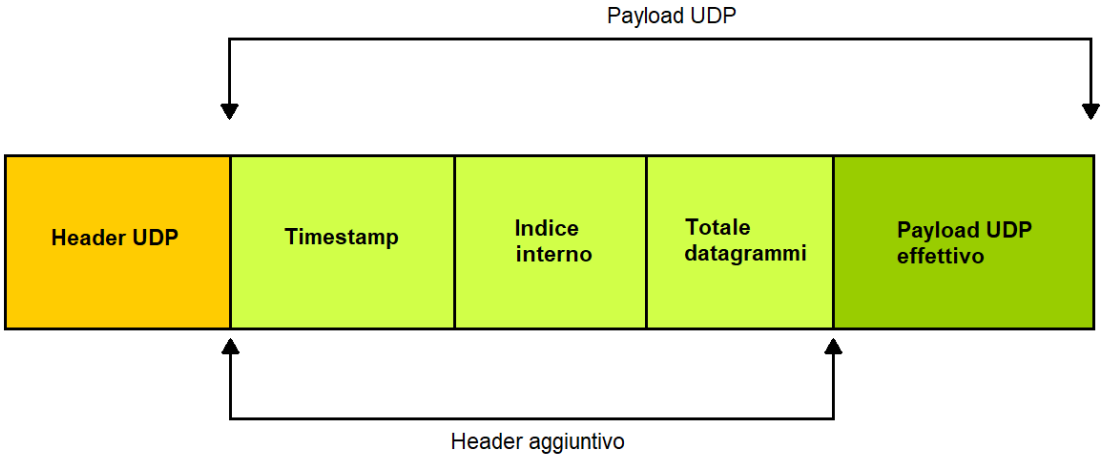

Figura 41 – Formato del datagramma UDP usato per lo streaming tramite Raspberry

La figura descrive il formato del datagramma UDP che è stato progettato ed utilizzato per il trasporto di ogni singolo fotogramma. Oltre all'header del protocollo UDP, necessario per trasportare i pacchetti correttamente a destinazione, sono state aggiunti tre nuovi dati all'interno del payload UDP.

1) *Timestamp*: un numero intero con segno a 64 bit, rappresentante un preciso istante di tempo, espresso in millisecondi. Il suo valore è lo stesso per tutti i datagrammi UDP che trasportano i dati della stessa immagine e viene aggiornato per ogni nuova immagine da inviare.

2) *Indice interno*: un numero intero con segno a 16 bit. Indica la posizione del datagramma all'interno della successione di pacchetti che fanno parte di una stessa immagine; è indispensabile per risolvere un eventuale problema di arrivo a destinazione di datagrammi con una sequenza diversa da quella di invio.

3) *Totale datagrammi*: anch'esso un interno con segno a 16 bit. Indica il numero totale di datagrammi che trasportano i dati di una determinata immagine. È necessario al destinatario per poter creare un buffer di dimensioni adatte ad ospitare tutti i datagrammi di una singola immagine.

In totale questo header occupa 12 bytes (96 bit) e dunque il carico utile che viene trasferito non sarà più di dimensione 65507 bytes come consente lo standard, ma sarà inferiore di 12 bytes (65495 bytes), che va a costituire, come riporta la figura, il Payload UDP effettivo. Ogni pacchetto UDP trasporta, dunque, 12 bytes in meno di informazione utile: una perdita molto bassa, inferiore dello 0,02% rispetto al payload totale e quindi accettabile in termini di risorse.

In questo modo per ogni immagine che viene generata dall'encoder JPEG, vengono creati datagrammi UDP con questo formato, anche nel caso in cui sia necessario uno solo di essi per trasportare tutta l'informazione, questo per avere una omogeneità negli algoritmi eseguiti a destinazione. Il datagramma UDP viene utilizzato in tutta la sua dimensione per poter usare un numero minimo di pacchetti per ciascuna immagine; naturalmente l'ultimo pacchetto potrà essere caratterizzato da un payload che non occupa tutta la dimensione possibile, dipendente dal fatto che la dimensione

dell'immagine sia multipla o meno della dimensione del payload effettivo trasmissibile (65495 bytes).

Per ogni immagine prodotta, quindi, vengono generati un numero variabile di datagrammi UDP con il formato in figura 41 e vengono inviati uno alla volta. A ricezione, gli utenti dispongono di un dispositivo Android ed eseguono un'app appositamente progettata per gestire questo specifico formato, stabilendo in questo modo un protocollo di comunicazione tra mittente e destinatari. Il nuovo formato dei datagrammi ricalca in parte, come anticipato, quello caratteristico del protocollo RTP: anche in esso, infatti, è presente un timestamp e un indice interno (chiamato "sequence number") in modo tale da sincronizzare i contenuti ricevuti nel tempo e numerare ogni pacchetto RTP per poter identificare perdite e ristabilire l'ordine corretto.

L'applicazione mobile consente semplicemente di visualizzare a tutto schermo la sequenza video che viene trasmessa. L'inizio della riproduzione avviene immediatamente dopo la sua apertura, questo perché è stata progettata per inscriversi all'avvio al gruppo Multicast di interesse così da mettersi subito in ascolto dei dati in arrivo.

Per ogni datagramma ricevuto, viene letto l'header aggiuntivo e in questo modo vengono prese le decisioni necessarie per gestire i casi possibili. Per prima cosa, viene sempre salvato il timestamp dell'ultima sequenza di datagrammi, in modo da confrontarlo con quello del datagramma appena ricevuto; logicamente possono succedere tre casi.

1) Timestamp salvato > timestamp letto

   Questa eventualità occorre quando arriva un pacchetto fuori sequenza, ed è più vecchio rispetto alla precedente sequenza di datagrammi: tutto ciò può teoricamente succedere in quanto, come detto, il protocollo UDP non è affidabile e perciò non dà garanzie che i pacchetti arrivino nella stessa sequenza di come sono stati spediti; per di più bisogna tenere conto anche del traffico presente sulla rete che può causare un ritardo di trasferimento. In questo caso si sceglie sempre di scartarlo, perché rappresenta un'informazione passata. L'applicazione, quindi, accetta sempre e solo immagini che siano le più vicine in termini di tempo a quelle che il trasmettitore sta elaborando ed inviando, in modo da garantire la latenza minima.

2) Timestamp salvato < timestamp letto

   Quando si verifica questa situazione, significa che è stato ricevuto il primo datagramma di una nuova sequenza di pacchetti e che quindi il trasmettore ha generato un nuovo fotogramma. In questo caso viene aggiornato il timestamp salvato con quello letto e viene poi controllato se il datagramma è unico o se ne devono ricevere degli altri di una stessa immagine; il primo caso avviene quando l'indice interno e il numero totale di datagrammi sono entrambi uguali ad uno. Se ciò non accade, il sistema salva il payload di questo pacchetto nel buffer appositamente creato e si rimette in ascolto dei successivi, aggiornando un contatore per tenere traccia della fine della sequenza di datagrammi dell'immagine; altrimenti si procede a decodificare i dati e a visualizzare il fotogramma sullo schermo del dispositivo.

3) Timestamp salvato = timestamp letto

   Quest'ultimo caso si manifesta quando si legge un datagramma che trasporta dati appartenenti alla stessa immagine di quella dei pacchetti che sono stati precedentemente analizzati. Si procede, quindi, a salvare le sue informazioni nel buffer e ad aggiornare il contatore. Se esso riporta di aver ricevuto tutti i pacchetti della stessa immagine (assumendo il valore zero), si procede a decodificare i dati salvati fino a quel momento e quindi a visualizzare il fotogramma.

Quanto appena descritto è in sintesi la procedura di ricezione sviluppata nell'applicazione Android. Un esempio grafico maggiormente intuitivo è rappresentato in figura 42. Essa rappresenta un caso di ricezione di un fotogramma, trasmesso su tre datagrammi UDP con un valore di timestamp generico T, l'ultimo dei quali presenta un payload che non occupa tutto lo spazio possibile (il caso più probabile). Si è ipotizzato anche che i pacchetti arrivino a destinazione fuori sequenza: grazie alla linea temporale, si mette in evidenza come il terzo pacchetto generato dalla sorgente arrivi prima del secondo, che quindi viene analizzato per ultimo; la sequenza di ricezione quindi è 1-3-2. Il primo pacchetto letto, dunque, ha il timestamp impostato ad un certo valore T, l'indice interno è 1 e il numero totale di datagrammi è 3. Da queste informazioni si procede dunque ad aggiornare il timestamp con il valore T e a creare il buffer che

conterrà tutti i dati dell'attuale immagine che si sta ricevendo. Questo buffer è rappresentato in figura, sotto alla catena dei tre pacchetti, ed è logicamente suddiviso in tre parti, perché la sua dimensione è impostata grazie al valore "totale datagrammi", 3 in questo caso: quindi si crea un buffer capace di contenere teoricamente tre datagrammi UDP con dimensione massima del payload effettivo (3*65495 bytes totali); in questo modo si è certi di poter salvare correttamente i dati dell'immagine dell'attuale serie di pacchetti, dato che a priori non è possibile conoscere la sua dimensione totale. È molto probabile, comunque, che non tutto il buffer venga riempito: nell'esempio il payload del terzo pacchetto, come accennato, non è di dimensioni massime e quindi esso sarà riempito solo parzialmente, la restante parte vuota è quella di colore grigio.

Una volta creato il buffer, si salvano i 65495 bytes del primo payload nei suoi primi 65495 bytes. Naturalmente l'ordine dei bytes è fondamentale e deve essere la stessa generata dall'encoder JPEG, altrimenti la decodifica porta a risultati errati. Infine si inizializza il contatore al valore 2, segno che bisogna ancora ricevere due datagrammi per completare la ricezione dell'intera immagine.

Successivamente si legge il prossimo datagramma ricevuto, che in realtà è il terzo ed ultimo della sequenza prodotta dalla sorgente. Nella sostanza non cambia nulla, perché i bytes del payload (inferiori al numero 65495, cioè il massimo) vengono salvati nell'ultima parte logica del buffer, ed il contatore, decrementato, informa che in realtà bisogna ricevere un ultimo pacchetto.

Infine, viene ricevuto il datagramma numero 2 della sequenza e il suo contenuto salvato nel buffer. A questo punto il contatore è azzerato, quindi tutti i dati memorizzati finora vengono decodificati tramite il decoder nativo in Android, il quale genera un'immagine Bitmap che verrà visualizzata tramite una Surface View. Una volta fatto, il sistema si rimette in ascolto di nuovi datagrammi, con un nuovo timestamp, e la procedura continua allo stesso modo.

Riassumendo, si è descritta in maniera dettagliata tutta la procedura eseguita dall'applicazione Android per connettersi alla sessione di streaming e visualizzarne i contenuti. La modifica al formato dei datagrammi UDP permette di superare il limite massimo di dimensione che ogni datagramma UDP deve rispettare, evitando quindi di essere scartato, e permette in questo modo di poter superare tutti i problemi connessi ad immagini JPEG più grandi del singolo payload UDP, eliminando ogni situazione di blocco dello streaming in tempo reale.

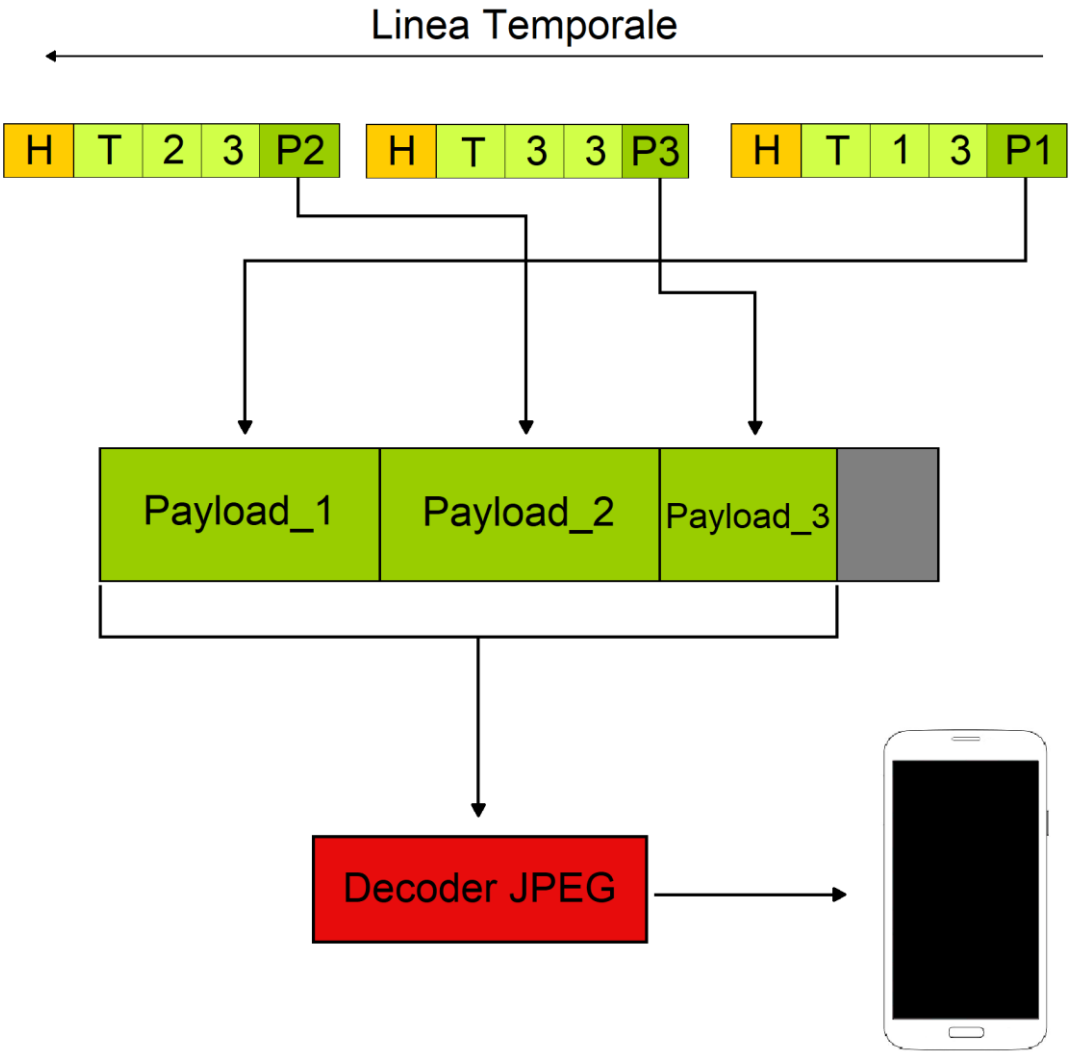

Figura 42 – Schema procedurale per la visualizzazione a destinazione di ciascun fotogramma inviato in streaming tramite Raspberry

4.4 Analisi delle prestazioni

Passando all'aspetto pratico, si riportano ora alcuni dati che descrivono le prestazioni di questa soluzione di streaming, in termini di latenza tra ciò che la webcam acquisisce e quello che viene visualizzato a destinazione e numero di fotogrammi per secondo trasmessi. Indubbiamente, in linea teorica, l'aumento del numero di datagrammi UDP per singola immagine comporta un ritardo maggiore nel visualizzarla rispetto alla gestione di un singolo pacchetto, perché il dispositivo di destinazione deve elaborare più dati per singolo fotogramma.

I dati sono stati generati usando due dispositivi connessi al gruppo Multicast ed entrambi hanno fornito le stesse prestazioni. Come era stato detto durante l'analisi del framework GStreamer, la webcam utilizzata permette in queste prove un frame rate massimo teorico di 30. Premesso questo, si riportano quattro casi dove, per ciascuno di essi, un'immagine viene trasmessa con un numero di datagrammi sempre maggiore, da uno a quattro; ciò è stato reso possibile modificando la qualità di codifica JPEG nella pipeline di GStreamer, portandola dal valore di default 85 a 95, su un massimo di 100. I risultati raccolti sono i seguenti.

1) Un solo datagramma trasmesso

   Si è potuto analizzare in media un frame rate pari a 24, ma è stato possibile raggiungere anche valori di 25-27 fps. Il ritardo calcolato è risultato essere molto basso, inferiore ai due decimi di secondo.

2) Due datagrammi trasmessi

   In questo caso inesorabilmente il frame rate è più basso rispetto al caso migliore di trasmettere un solo pacchetto: si è osservato in media un valore di 20 fps elaborati, con valori massimi intorno a 23. Per quanto riguarda la latenza, non si è analizzata grande differenza con il caso precedente, quindi il suo valore è sempre sotto i due decimi di secondo.

3) Tre datagrammi trasmessi

   Questo caso ha visto il frame rate scendere ancora ad un valore medio di 15, con valori massimi intorno a 18. La latenza si è osservata essere leggermente aumentata ed il suo valore quantitativo è di circa 30 decimi di secondo, ancora molto basso.

4) Quattro datagrammi trasmessi

   Quest'ultimo caso analizzato presenta il frame rate più basso, circa 11. Il ritardo rispetto alla sorgente è leggermente aumentato, circa 50 decimi di secondo.

Questi dati raccolti permettono di constatare due aspetti chiave: il primo riguarda, come ipotizzato, l'inevitabile calo di frame rate dovuto all'aumento dei datagrammi necessari per veicolare ogni singola immagine alle destinazioni. Si può riassumere dicendo che la fluidità della riproduzione video è bassa quando vengono impiegati quattro pacchetti per immagine, mentre migliora di poco per quanto riguarda il terzo caso. Nei primi due essa raggiunge buoni risultati.

L'altro aspetto riguarda una latenza complessiva sotto la soglia del secondo: questo è un risultato ottimo in quanto è importante che essa sia la più bassa possibile in applicazioni real-time, e da questo punto di vista rende fattibile l'approccio allo streaming dati proposto in questa Tesi.

C'è da dire, inoltre, che nella maggioranza dei casi che si sono analizzati è stato sufficiente la trasmissione massima di solo due datagrammi UDP, sia in ambienti chiusi che in quelli aperti, lasciando la qualità di codifica JPEG al valore di default di 85 su 100, il quale fornisce comunque buoni risultati. Questo fatto, quindi, rende la soluzione proposta realizzabile ed usabile in contesti reali, a patto di essere caratterizzati da una limitata dinamicità.

CAPITOLO 5

LIVE STREAMING CON PROJECT TANGO

5.1 Architettura generale

In quest'ultimo capitolo di Tesi si affronta la realizzazione di una soluzione di streaming in tempo reale impiegando come sorgente video la camera di profondità presente nel dispositivo Tango. In questo modo lo smartphone è in grado di trasferire la nuvola di punti ai vari destinatari, rappresentata graficamente con lo stesso codice colore visto nel capitolo 3, permettendo quindi di creare una condivisione della percezione ambientale che il dispositivo è in grado di offrire.

Lo schema progettuale di questa soluzione è mostrato in figura 43.

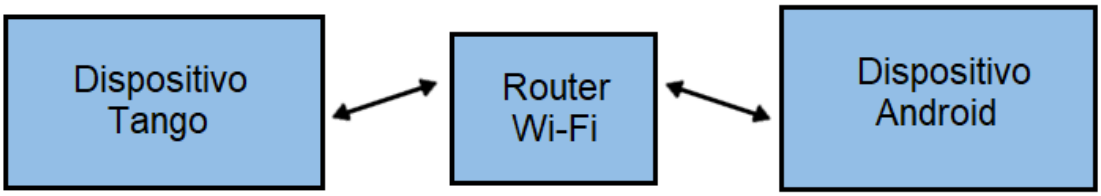

Figura 43 – Architettura utilizzata per lo streaming dati tramite la piattaforma Tango

L'idea è simile a quanto già visto nel capitolo precedente. Anche qua, quindi, il contesto di utilizzo è limitato ad una stessa rete locale, all'interno della quale le informazioni vengono distribuite. In questo caso la scheda elettronica insieme alla camera USB sono sostituiti dal dispositivo Tango, che integra così tutte le tecnologie necessarie per funzionare da sorgente dello streaming.

I dati della nuvola di punti generati dal device vengono trasportati in modalità wireless ai dispositivi destinatari per mezzo di un router, anche qua non connesso alla rete Internet ma solo usato come mezzo per trasportare le informazioni dal mittente ai vari destinatari.

Per implementare questo progetto, sia il dispositivo sorgente che quelli di destinazione eseguono due distinte applicazioni Android appositamente create: la prima, eseguita dalla sorgente, cattura la nuvola di punti tramite le API Java di Tango e, mediante il modulo Wi-Fi integrato, spedisce i dati al router. Un esempio di schermata di questa applicazione è osservabile in figura 44.

Figura 44 – Interfaccia dell'applicazione sorgente dello streaming dati

L'interfaccia impiegata è molto semplice, data la bassa richiesta di interazione con essa. Sull'intero schermo è visualizzato tutto ciò che la color camera riesce a catturare, questo per permettere al display di svolgere la funzione di "finestra" nell'ambiente e capire perciò cosa è inquadrato in quel momento dalla camera di profondità (utile soprattutto per la fase di sviluppo dell'app), e quindi comprendere quale porzione di nuvola di punti viene generata ed inviata. Nella parte alta sono presenti le informazioni riguardanti il Pose del dispositivo e il numero di punti della nuvola, con la loro distanza media dal device, calcolati istante per istante, esattamente come visto per l'app HelloTango; queste informazioni, come sarà descritto a breve, sono indispensabili per permettere la corretta visualizzazione dell'insieme di punti a destinazione. L'interfaccia termina con l'unica concreta interazione con l'utente, il pulsante "Inizia Streaming": esso permette di cominciare a trasmettere i Point Cloud, così come di interrompere tutto il processo se ripigiato una seconda volta (la scritta cambia in "Ferma Streaming"); in questo modo, esso svolge una funzione simile ad un interruttore On/Off, funzionante in ogni istante.

L'applicazione che riceve questi dati è del tutto simile come comportamento a quanto visto nel capitolo precedente: essa consente semplicemente di visualizzare la nuvola di punti su tutto lo schermo del dispositivo Android, sia esso uno smartphone che un tablet. La novità è nella presenza del render grafico della libreria Rajawali, usato per la visualizzazione dei Point Cloud tramite il codice colore.

Anche in questo caso, naturalmente, i dati di interesse sono trasportati a destinazione mediante il protocollo UDP, in modo da garantire un basso ritardo nella riproduzione, aspetto fondamentale in un contesto real-time. La sorgente, in questo caso, fa uso delle API dello standard Java per creare i datagrammi UDP, ogniqualvolta i dati vengono resi disponibili dal sistema Tango, e poterli spedire, il tutto integrato nell'applicazione senza usare software di terze parti.

Similmente a quanto già visto, l'indirizzo di destinazione è di tipo Multicast, ed è stata scelta una porta logica il cui uso non appartiene a nessuna applicazione standard.

Si passa ora ad analizzare con più dettaglio l'oggetto di interesse di quest'infrastruttura di streaming: la nuvola di punti. Questo è certamente ciò che rende differente questa soluzione di streaming dall'altra in termini di sviluppo tecnico; essa viene generata tramite l'interfaccia di callback vista nel capitolo 3.

L'applicazione sorgente, quindi, riceve la nuvola di punti ogniqualvolta è disponibile un nuovo set di dati dai sensori Tango, e viene salvata nell'apposito oggetto `TangoPointCloudData`. All'interno di esso, Tango memorizza l'insieme dei punti in un buffer apposito contenente dati di tipo `float`, gestito dall'oggetto `FloatBuffer`.

I dati memorizzati rappresentano, ogni quattro numeri `float`, le coordinate di ciascun punto nello spazio; la tabella in figura 45 fornisce un esempio schematico di tutto questo.

X	Y	Z	C
-0.24201293	0.21115293	0.9276736	1.0
-0.23685575	0.21045394	0.9254909	1.0
-0.2320274	0.21002676	0.9244713	1.0
-0.22650944	0.20894745	0.9205482	1.0
-0.22156155	0.20836258	0.91877025	1.0
-0.21689276	0.2080225	0.91804224	1.0
-0.21242148	0.20786221	0.9180803	1.0
-0.21242148	0.20786221	0.9180803	1.0
-0.20713946	0.20688593	0.9144851	1.0
-0.202706	0.2067341	0.9145061	1.0
-0.19678622	0.20502752	0.90761936	1.0
-0.19345886	0.20600656	0.91259557	1.0
-0.18821289	0.20494094	0.908491	1.0

Figura 45 – Rappresentazione tabellare di una porzione di nuvola di punti

Essa è stata costruita partendo da dati reali catturati dal sistema Tango e resi leggibili, in questa figura, tramite una rappresentazione tabellare; ovviamente nel buffer essi vengono salvati consecutivamente, partendo da sinistra a destra e dall'alto verso il basso della tabella.

Si può notare che le coordinate spaziali di ogni punto sono tre, nel formato [X ,Y, Z] seguendo la logica di uno specifico sistema di riferimento, descritto nel capitolo 3. Inoltre per ogni punto viene salvato anche un altro numero, il valore di confidenza C. Esso fornisce una stima di affidabilità dei valori delle coordinate che il sistema Tango genera per ogni punto, ed i suoi valori sono compresi tra 0 ed 1: il valore 1 indica che i dati generati sono totalmente affidabili, mentre il valore 0 suggerisce l'opposto. Da questa porzione di nuvola di punti in figura si può generalizzare, dicendo che questo valore in quasi tutti i casi possiede il valore 1 e rappresenta quindi, come verrà analizzato in seguito, un'informazione inutile da trasportare a destinazione.

5.2 Lo sviluppo software

Tutto quanto analizzato finora porta alla conclusione che la quantità di dati che la sorgente invia per ogni Point Cloud generato sia elevata: ogni punto, infatti, è caratterizzato da quattro numeri da 32 bit ciascuno, occupando quindi 16 byte in totale. Dato che, mediamente, il sistema Tango genera in un ambiente indoor una quantità di punti di circa 35000, ciascuno quindi da 16 byte, ecco che si può già intuire come l'insieme totale dei dati da inviare ai dispositivi di destinazione sia molto superiore rispetto alla dimensione massima del payload ospitabile in un datagramma UDP (65507 byte).

Come conseguenza di questo fatto, sono state sviluppate tecniche simili a quanto visto nel caso precedente di streaming: per ogni nuvola di punti generata, dunque, vengono creati un numero di datagrammi compreso tra uno e la quantità necessaria per trasportare tutti i dati a destinazione. Il datagramma UDP presenta un formato simile a quanto già visto nel capitolo 4 ma con alcune differenze, come mostra la figura 46.

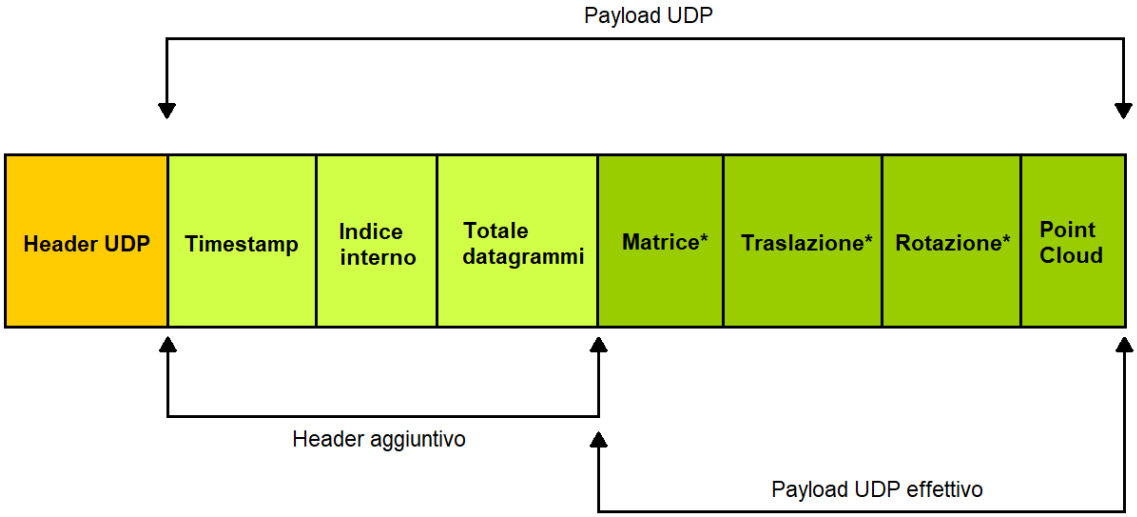

Figura 46 – Formato del datagramma UDP usato per trasportare in streaming la nuvola di punti

Come è possibile osservare, anche in questo caso il payload UDP, ovvero la quantità da 65507 byte che rappresenta l'informazione utile, è suddiviso in un header agguntivo e nel contenuto informativo vero e proprio.

L'header ulteriore ha esattamente la stessa struttura impiegata nel progetto precedente: è presente un timestamp, ovvero un numero intero con segno da 64 bit che rappresenta un preciso instante di tempo, quello di inizio di ogni serie di pacchetti per ciascuna nuvola di punti; un indice interno, numero intero con segno da 16 bit indicante la posizione del datagramma nella successione di pacchetti ed infine il totale datagrammi, anch'esso un numero intero con segno da 16 bit che specifica il numero totale di datagrammi che è stato necessario creare per il singolo Point Cloud.

Oltre a questi dati, utili per ricostruire a destinazione l'informazione originaria di partenza, lo stesso payload effettivo è suddiviso in quattro parti.

1) *Matrice*: indentifica la matrice di trasformazione, calcolata usando l'API Java del sistema Tango grazie alla classe `TangoSupport`. Essa permette, come era stato analizzato nel capitolo 3, di convertire il sistema di coordinate Tango in quello di OpenGL per poter rappresentare grafica la nuvola di punti. Essa è formata da 16 numeri in formato `float`, per un totale di 64 bytes.

2) *Traslazione*: sono i dati che rappresentano il vettore traslazione nel Pose del dispositivo Tango: in totale sono tre numeri `float`, ovvero le coordinate [X, Y, Z] e quindi 12 bytes di informazione.

3) *Rotazione*: è il quaternione, ovvero i dati che descrivono l'orientamento del dispositivo Tango nello spazio, sempre estrapolati dal Pose ottenuto grazie a Motion Tracking. È caratterizzato, come già descritto, da quattro numeri `float` nel formato [X, Y, Z, W], per un totale di 16 byte di dati.

4) *Point Cloud*: è la porzione di nuvola di punti trasportata dal singolo datagramma UDP.

In questa soluzione, dunque, il payload effettivo trasporta informazioni in più rispetto al dato reale di interesse, cioè la nuvola di punti. I primi tre dati, tuttavia, sono graficamente rappresentati con un asterisco: questo sta ad indicare che in realtà questi non sono presenti sempre in ogni datagramma UDP ed infatti vengono trasportati soltanto dal primo datagramma generato. I restanti pacchetti, dunque, avranno per l'esattezza 92 byte in più per trasmettere la restante parte della nuvola di punti.

Questa porzione di informazione in più, trasportata solamente una volta per ogni nuvola di punti generata dal sistema Tango, è necessaria per la visualizzazione dei Point Cloud sui dispositivi di destinazione: essi, come accennato, utilizzato il framework Rajawali per poter eseguire il render grafico, e quindi da una parte la matrice di trasformazione è essenziale per posizionare correttamente l'oggetto grafico della nuvola nella scena

OpenGL; dall'altro, è allo stesso tempo indispensabile che la camera virtuale all'interno di essa, cioè il punto di vista rispetto al quale la scena OpenGL viene osservata, sia posizionata nel modo corretto, così da poter fornire la giusta visuale all'ambiente virtuale: tutto questo è possibile utilizzando il Pose (traslazione e rotazione) fornito da Motion Tracking.

In una singola nuvola di punti, l'utilizzo dell'header aggiuntivo (12 byte) e dei tre dati trasportati una sola volta (92 byte) rappresentano, di nuovo, un overhead rispetto al carico utile assolutamente trascurabile: nel primo datagramma della catena, infatti, viene utilizzato poco meno dello 0,16% del payoad, mentre nei restanti pacchetti meno dello 0,02%.

Di contro, la presenza del valore di confidenza per ogni punto della nuvola genererebbe una grande quantità di informazione da trasportare, sostanzialmente inutile al fine di visualizzare i punti: infatti, prendendo ad esempio il valore 35000 che rappresenta verosimilmente la quantità media di punti in un ambiente indoor, il valore di confidenza comprende esattamente il 25% del totale dei dati generati, e sono necessari ben tre datagrammi UDP per poter trasmettere solo questo insieme di valori. In conseguenza di questo fatto, è stata implementata una procedura nella sorgente che, data la nuvola di punti originaria, ne genera un'altra identica con il valore di confidenza eliminato per ogni punto. In questo modo si è potuto così risparmiare notevolmente sulla quantità totale di dati da inviare, potendo diminuire il ritardo nella riproduzione della nuvola di punti a destinazione, nonché evitare di far calare ulteriormente il suo il frame rate.

Si passa ora ad analizzare tecnicamente come avviene la trasmissione dei dati. Sostanzialmente il procedimento è analogo a quanto visto nel capitolo precedente: per ogni datagramma inviato con il formato descritto in figura 46, infatti, viene letto il contestuale header aggiuntivo e viene deciso, tramite il timestamp, se salvare o meno il pacchetto; se la decisione è positiva, esso viene memorizzato all'interno del buffer ospitante tutti i dati dell'attuale nuvola di punti, prestando attenzione all'ordine in cui i bytes vengono salvati in esso. Un esempio grafico di questa procedura è presente in figura 47.

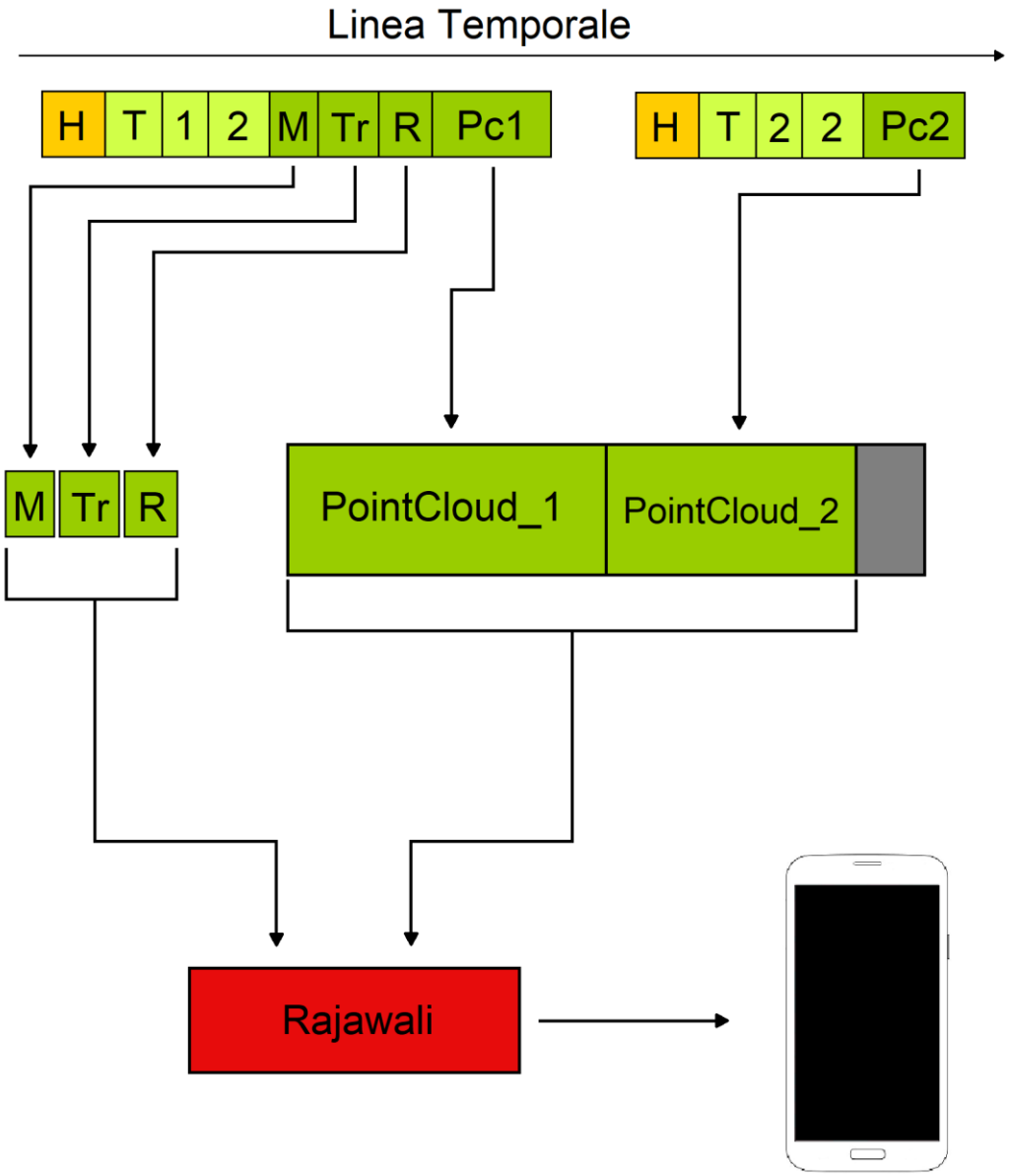

Figura 47 – Schema procedurale per la visualizzazione a destinazione della nuvola di punti inviata in streaming dal dispositivo Tango

L'esempio mostra la ricezione dei datagrammi UDP e le successive procedure che l'applicazione Android esegue per visualizzare il contenuto finale. Come si può osservare, la sorgente ha generato due datagrammi, numero inferiore rispetto ai casi reali di utilizzo generale, ma necessario per rappresentare con chiarezza il grafico. Il dispositivo Tango, nello specifico, legge i dati dai suoi sensori e successivamente invia queste informazioni tramite i due datagrammi: queste due operazioni sono effettuate su thread secondari diversi e per questo si è dovuto implementare un meccanismo di sincronizzazione per accedere ed utilizzare in maniera coerente i dati condivisi, evitando conflitti.

In questo caso, per rappresentare con più chiarezza il grafico, si è scelto di ricevere i pacchetti nell'esatto ordine di invio. Il primo datagramma ricevuto, quindi, avrà nel suo header aggiuntivo un certo timestamp T, l'indice interno di valore 1 e il totale datagrammi uguali al valore 2. Esso è anche l'unico che contiene le informazioni aggiuntive nel payload effettivo: una certa matrice di trasformazione M, un vettore di traslazione Tr ed un quaternione per la rotazione R, ed infine la prima parte della nuvola di punti Pc1; quest'ultima comprendente per ciascun punto, come in generale per tutti i pacchetti, le tre coordinate spaziali [X, Y, Z] con il valore di confidenza assente.

Una volta ricevuto questo datagramma, i tre dati utili per disegnare graficamente i punti vengono salvati in apposite variabili. Grazie al valore indicante il totale dei datagrammi per il suddetto insieme di punti, viene creato il buffer che ospiterà la nuvola di punti totale, diviso logicamente in due parti uguali, ciascuna di dimensione pari al massimo payload trasportabile (65507 byte). In questo modo ci si assicura una dimensione totale sufficiente per ospitare tutti i dati della nuvola. Infine, i dati della prima parte dei Point Cloud vengono salvati in questa struttura a partire dalla posizione 0 del buffer, essendo i primi.

Il secondo datagramma ricevuto, come detto, contiene la seconda parte della nuvola nel suo payload, il quale quindi non comprende i tre dati per la rappresentazione grafica. Una volta letto l'header aggiuntivo, si salva quest'ultima parte della nuvola nel buffer, a partire dalla posizione successiva dell'ultimo byte memorizzato in precedenza, riguardante la prima parte di Point Cloud. Anche in questo caso si è ipotizzato che verosimilmente non tutto lo spazio utile per il payload venga utilizzato e dunque il buffer non sarà riempito completamente; la parte eccedente è anche qua raffigurata in grigio.

Una volta ricevuto l'ultimo datagramma, con un opportuno sistema di sincronizzazione, si va ad assegnare ad una variabile condivisa il valore che permette al render grafico di capire che tutti i nuovi dati sono stati resi disponibili per essere visualizzati. A questo punto, tramite i suoi metodi, essi vengono passati al render, permettendo così di aggiornare i vertici ed i colori della nuvola di punti, nonché di impostare correttamente il punto di vista della scena in OpenGL ed infine di disegnare i Point Cloud sullo

schermo, con lo stesso codice colore che si è potuto osservare nel capitolo 3. Il procedimento poi ricomincia con una nuova sequenza di pacchetti nel medesimo modo. Tutto questo processo permette quindi di avere uno streaming senza interruzioni, qualunque sia la dimensione della nuvola di punti che deve essere trasmessa.

5.3 Analisi delle prestazioni

Per quanto riguarda le prestazioni, bisogna innanzitutto premettere come la frequenza di generazione della nuvola di punti da parte del sistema Tango sia molto più bassa di quella di una comune fotocamera/webcam nel generare i suoi dati. Se quest'ultima, nel caso della soluzione di streaming precedente, produce fino a 30 fotogrammi al secondo, la camera di profondità di Tango riesce a generare da 6 a 7 Point Cloud nello stesso lasso di tempo; questo fatto è causato dalla complessità di calcolo necessaria per generare ciascuna nuvola di punti e le relative coordinate per ognuno di essi.

Da questo dato di partenza, si passa ora ad esaminare le prestazioni quantitative di questa tecnica di streaming, sempre tenendo in considerazione la latenza ed il frame rate video riprodotto a destinazione. Si sono analizzati tre casi, caratterizzati da un numero di punti della nuvola decrescente e lo streaming ha riguardato due dispositivi Android connessi, i quali hanno fornito le stesse prestazioni; la quantità di punti espressa rappresenta un valore medio.

1) 35000 punti

   Questo è il caso più comune che si può avere durante l'esplorazione di un ambiente indoor. I dati raccolti hanno permesso di stimare che una quantità pari a 7 datagrammi UDP è necessaria per trasportare questi dati. Inoltre il frame rate calcolato è di circa 5 fps. Infine il ritardo tra la generazione della nuvola di punti dalla sorgente e la sua riproduzione a destinazione è di circa 60 decimi di secondo.

2) 20000 punti

   In questo caso il numero di datagrammi scende a 4. Il frame rate calcolato è intorno a 5,5 fps. Il ritardo si abbassa fino a 40 decimi di secondo.

3) 10000 punti

   In quest'ultimo esempio, il numero di datagrammi necessari scende a 2, mentre il frame rate è mediamente 5,7 fps; il ritardo è il più basso degli altri casi, sotto i 30 decimi di secondo.

Nella maggioranza dei casi, quindi, ogni nuvola di punti è trasmessa mediante in 6-7 datagrammi UDP, un valore triplo rispetto allo streaming della webcam visto nel precedente capitolo (dove in media era pari a 2); di contro, però, la frequenza con cui vengono generati i dati dal dispositivo Tango è circa 4-5 volte inferiore rispetto a quanto analizzato nel caso di utilizzo della webcam. Questa grande differenza permette quindi di mantenere un frame rate a destinazione di poco inferiore rispetto a quello prodotto dal sistema Tango: nel caso peggiore sono 5 frame al secondo riprodotti dai dispositivi di destinazione, anziché 6 o 7, quindi la frequenza di render dei dati è molto fedele rispetto alla sorgente; pur essendo un dato oggettivamente basso, esso consente comunque di gestire situazioni caratterizzate da una bassa dinamicità.

Era inoltre prevedibile come al diminuire della quantità dei punti, diminuisse il ritardo di riproduzione così come il numero di datagrammi necessari alla trasmissione, mentre aumentasse, anche se di una quantità minima, il frame rate. Nella maggioranza dei casi si ha un ritardo massimo di 60 decimi di secondo, un valore comunque basso per poter garantire una buona reattività dello streaming. Con una quantità intorno ai 10000 punti, invece, si ottiene un valore del ritardo paragonabile a quello mediamente calcolato nel precedente capitolo (circa 20 decimi).

CONCLUSIONI

Si giunge ora alla conclusione di questa Tesi. In questo lavoro si è posta l'attenzione sullo studio di tecnologie e lo sviluppo di applicazioni mobili incentrate sull'analisi e la condivisione in remoto delle caratteristiche ambientali di luoghi fisici, con l'obiettivo di migliorarne la percezione ambientale, nelle situazioni in cui questa viene ad essere limitata per svariate ragioni.

Un primo studio fondamentale del lavoro di Tesi è stato l'analisi dello smartphone Lenovo Phab2 Pro, con a bordo la tecnologia di computer vision Tango. Attraverso questo percorso di sviluppo software è stata creata un'applicazione mobile in grado di sfruttare le sue funzionalità principali: si sono quindi implementate in contesti reali le funzionalità di Motion Tracking, Area Learning e Depth Perception.

La prima tecnologia ha permesso di seguire con estrema precisione i movimenti compiuti dallo smartphone nell'ambiente fisico, sia esso indoor che outdoor, con la medesima qualità di risultati e senza l'aiuto di segnali esterni come il GPS. In questo modo è stato possibile rappresentare graficamente la traiettoria percorsa ed avere così salvato lo spostamento dell'utente in un dato luogo, potendo così capire i suoi spostamenti nello spazio, oltre a fornire dati ulteriori come per esempio la distanza percorsa.

La seconda tecnologia ha potuto realizzare il riconoscimento di ambienti precedentemente visitati in un tempo molto breve; la riuscita della localizzazione è stata compiuta sia in luoghi indoor che outdoor, con quest'ultimo caso che ha richiesto un tempo maggiore. Casi di fallimento si sono verificati quando l'ambiente circostante aveva cambiato in maniera profonda il suo aspetto, un risultato prevedibile ma che può venire mitigato dalla possibilità di salvare più volte lo stesso ambiente sotto vari aspetti.

La terza ed ultima tecnologia ha permesso di rappresentare graficamente la profondità spaziale di un ambiente mediante la generazione di una nuvola di punti: in essa, infatti, ciascun punto è stato disegnato con un codice colore in base alla distanza dal dispositivo Tango. La nuvola di punti è stata utilizzata anche per generare un modello tridimensionalmente dello spazio visitato; in tutti questi casi, si sono sempre presentate piccole porzioni dello spazio che non venivano riconosciute dai sensori e dunque caratterizzate dall'assenza di punti: questo è causato dalle caratteristiche fisiche del sensore ad infrarosso, ma non rappresenta un grosso problema perché la quasi totalità dell'ambiente viene ad essere percepita dal sistema Tango.

Si può definire, quindi, la tecnologia Tango essere concretamente utilizzabile in contesti reali grazie all'accuratezza dei suoi risultati, i quali contribuiscono a migliorare la percezione di ambienti fisici, il tutto con un costo massimo di 499 €, oggettivamente basso.

Nell'ultima parte del lavoro di Tesi si è affrontato lo studio di tecnologie che permettessero una condivisione dei dati percepiti dall'ambiente circostante verso un insieme generico di utenti. Questo aspetto ha coinvolto lo sviluppo di due distinte soluzioni di streaming dati eseguito in tempo reale tra una sorgente video e gli utenti destinatari.

I risultati ottenuti dalle due soluzioni tecnologiche sono stati positivi dal punto di vista della qualità dello streaming: infatti si è potuto misurare una latenza tra la cattura da parte della sorgente dei dati e la successiva riproduzione a destinazione sempre sotto il secondo, nella maggioranza dei casi di utilizzo. Per quanto riguarda la quantità di fotogrammi riprodotti al secondo nei dispositivi destinatari bisogna distinguere le due soluzioni: in quella caratterizzata dall'utilizzo di una tradizionale webcam, nei casi medi di utilizzo reali, si riesce a raggiungere un valore di frame rate medio di 22 (considerando che il numero di datagrammi trasmessi mediamente varia da uno a due), raggiungendo diverse volte anche valori di 25, 27 fotogrammi al secondo, sui 30 teorici catturabili dalla sorgente; tutto questo permette di utilizzare questo sistema in casi di bassa dinamicità, come ad esempio la guida a distanza di veicoli non particolarmente veloci (<1m/s). Riguardo l'altro caso, dove viene utilizzata come sorgente lo smartphone Lenovo per generare e distribuire la nuvola di punti, il frame rate si mantiene poco sotto quello della sorgente, circa 5fps, un valore oggettivamente basso, frutto della complessità di calcolo da parte del sistema Tango per generare la nuvola di punti e le relative coordinate spaziali; nonostante ciò, questa grandezza è sufficiente per governare situazioni a bassa dinamicità.

Dal punto di vista della percezione umana, l'utilizzo del codice colore nella nuvola di punti permette in maniera immediata di riconoscere la profondità degli spazi e le distanze maggiori tra gli oggetti; di contro, nei casi in cui elementi dell'ambiente si trovano ad una lontananza simile dal dispositivo la rappresentazione dei colori tende ad appiattire le distanze poiché la colorazione dello spazio risulterà simile; per lo stesso motivo, anche l'osservazione da punti di vista differenti di spazi ed oggetti risulta troppo approssimata affidandosi solo alla visualizzazione dei Point Cloud.

Alla luce di tali risultati, l'approccio dello streaming dati proposto in questa Tesi dimostra una sostanziale fattibilità, a condizione di operare in contesti compatibili con le condizioni evidenziate: c'è infatti da tenere in considerazione la natura consumer degli strumenti utilizzati, per loro natura limitati dal punto di vista della potenza di

calcolo, considerando i costi contenuti sia della scheda elettronica che del dispositivo Tango, rispettivamente di 40 € e 499 €.

L'approccio a soluzioni consumer, dunque, impone inevitabilmente alcuni limiti. Potendo ipotizzare di adottare soluzioni professionali con maggiore potenza di calcolo, un possibile sviluppo futuro consisterebbe nel proporre un sistema di streaming dati con un maggior numero di frame al secondo trasmessi, permettendo in questo modo di gestire situazioni ad alta dinamicità, anche nel caso in cui sia distribuita la nuvola di punti. Tutto questo, naturalmente, necessita anche di una maggiore larghezza di banda: contando anche su questo aspetto, una seconda soluzione interessante da implementare potrebbe essere la trasmissione in remoto e in diretta della ricostruzione tridimensionale dell'ambiente tramite i Point Cloud, potendo in questo modo interagire con lo spazio e gli oggetti in esso presenti anche sotto varie prospettive, arricchendo l'esperienza percettiva.

BIBLIOGRAFIA

[1] Gran parte della documentazione riguardante il framework Tango e descritta in questa Tesi deriva dal sito web ufficiale di Project Tango, raggiungibile a questo link: <https://developers.google.com/tango/developer-overview>.

[2] Le due figure sono state catturate dal video di presentazione di Area Learning "*Introducing Project Tango Area Learning - Google I/O 2016*", reperibile a questo link: <https://www.youtube.com/watch?time_continue=655&v=NTZZCtmR3OY>.

[3] R.J. Valkenburg, A.M. Mc Ivor, "*Accurate 3D measurement using a Structured Light System*", Image and Vision Computing, p. 2, 1997.

[4] Larry Li, "*Time-of-Flight Camera – An Introduction*", Technical White Paper, January 2014 - Revised May 2014.

[5] Capitolo 17 della guida online ufficiale di OpenGL, dedicato alla rappresentazione della rotazione: <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/>.

[6] Achille Pattavina, "*Reti di telecomunicazione*", McGraw-Hill, p. 82, 2007.

[7] Postel, J., "*Transmission Control Protocol*", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[8] Postel, J., "*User Datagram Protocol*", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[9] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "*RTP: A Transport Protocol for Real-Time Applications*", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[10] Arnaud Loonstra, "*Videostreaming with Gstreamer*", Leiden University, 2014, <http://mediatechnology.leiden.edu/images/uploads/docs/wt2014_gstreamer.pdf>.

[11] Deering, S., "*Host extensions for IP multicasting*", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <https://www.rfc-editor.org/info/rfc1112>.

[12] Cotton, M., Vegoda, L., and D. Meyer, "*IANA Guidelines for IPv4 Multicast Address Assignments*", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <https://www.rfc-editor.org/info/rfc5771>.

[13] Reynolds, J. and J. Postel, "*Assigned Numbers*", RFC 1700, DOI 10.17487/RFC1700, October 1994, <https://www.rfc-editor.org/info/rfc1700>.