Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica

Ultimo Tango a Mountain View

Ricostruzione 3D mediante dispositivi mobili Tango

Tesi di laurea triennale

Relatore
Prof. Gilberto Filè

Laureando
Tommaso Padovan

Anno Accademico 2015-2016

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, dal laureando Tommaso Padovan presso l'azienda Vic srl.
L'obiettivo principale era esplorare la possibilità di sfruttare gli innovativi dispositivi *Tango* di *Google* per produrre scansioni tridimensionali degli oggetti inquadrati.
In primo luogo era richiesta la progettazione e lo sviluppo di una applicazione *mobile Android* dotata di una minimale interfaccia grafica in grado di scannerizzare un oggetto, esportarlo in un formato portabile ed effettuare su quest'ultimo ottimizzazioni ed il calcolo del volume.
In secondo luogo era richiesto di massimizzare il riuso delle soluzioni *OpenSource* presenti sul mercato.

"Strict modeling of the real world leads to a system that reflects today's realities but not necessarily tomorrow's"

— Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

VIC è stata fondata da *Alessio Bisutti* che, dopo aver sviluppato una lunga esperienza nel campo ispettivo, ha deciso di costituire una società in grado di offrire ai propri clienti un servizio professionale, chiaro ed affidabile, appoggiandosi sulle nuove tecnologie.
Si occupa di controlli per grandi ordini, sia di materie prime che di semi-lavorati, di cui individua e riporta eventuali danni, carenze nella spedizione e non conformità con quanto ordinato.
VIC iniziò a Venezia 7 anni fa come piccola società di ispezione locale. Fin dall'inizio, l'obiettivo principale di *VIC* è stata la riduzione del tempo tra ispezione e reporting al cliente. Ora l'obiettivo è raggiunto, perché *VIC* sta fornendo ai suoi clienti tutti i risultati e le informazioni importanti in tempo reale, senza alcun ritardo, grazie agli investimenti fatti nel campo della tecnologia e delle applicazioni mobili.

1.2 L'idea

Mansioni come determinare la corretta forma, peso, quantità e dimensioni degli oggetti da ispezionare sono tra le più importanti per i controlli effettuati dall'azienda.
Gli ispettori possono scattare molte fotografie, prendere appunti e sfruttare la loro esperienza per fornire stime accurate; si è manifestata però la necessità di affiancare queste ultime a dei dati quanto più possibile oggettivi e rapidi da ottenere.
Da qui nasce l'idea di fornire agli ispettori uno strumento informatico in grado di effettuare queste stime. Grazie alla ricostruzione computerizzata resa disponibile dai *Tango device* sarà possibile non solo visualizzare su uno schermo il modello 3D del soggetto della ispezione, ma anche ottenere ulteriori vantaggi quali:

- Avere una stima del volume e quindi del peso della materia prima.
- Confrontare l'oggetto con un modello ideale, permettendo così un rapido controllo eventuali di danni o deformazioni.

Con il prototipo realizzato durante lo *stage* sono rese disponibili solamente le funzionalità di ricostruzione dell'oggetto e calcolo (approssimato) del volume.
Alcune operazioni sarebbero troppo pesanti per le potenzialità del tablet, quindi è stato realizzato un *backend server* per tutte le elaborazioni delle ricostruzioni, mentre la ripresa delle stesse è affidata all'applicativo per tablet.

1.3 Il Prodotto - lato dispositivo

L'applicazione prodotta risponde, in maniera minimale, alle esigenze citate nel punto precedente.
La sua realizzazione presenta molti punti critici e rischi piuttosto difficili da prevedere. Per questo sono stati realizzati molti prototipi, al fine di escludere vie non percorribili e trovare una soluzione soddisfacente.
Lo scopo principale dell'applicazione lato tablet è quello di rilevare ed elaborare un corretto *Point Cloud* dell'oggetto che si vuole ispezionare.
Un *Point Cloud* non è altro che una descrizione algebrica di un oggetto tridimensionale ottenuta tramite un insieme, il più possibile fitto, di punti che lo compongono. I dispositivi Tango infatti, grazie al sensore di profondità, cercano di rilevare le triplette di coordinate del maggior numero di punti possibile. Sfruttando questi dati è possibile posizionare dei vettori nello spazio in maniera da fornire all'utente una rappresentazione comprensibile dell'oggetto; come quella in figura 1.1.

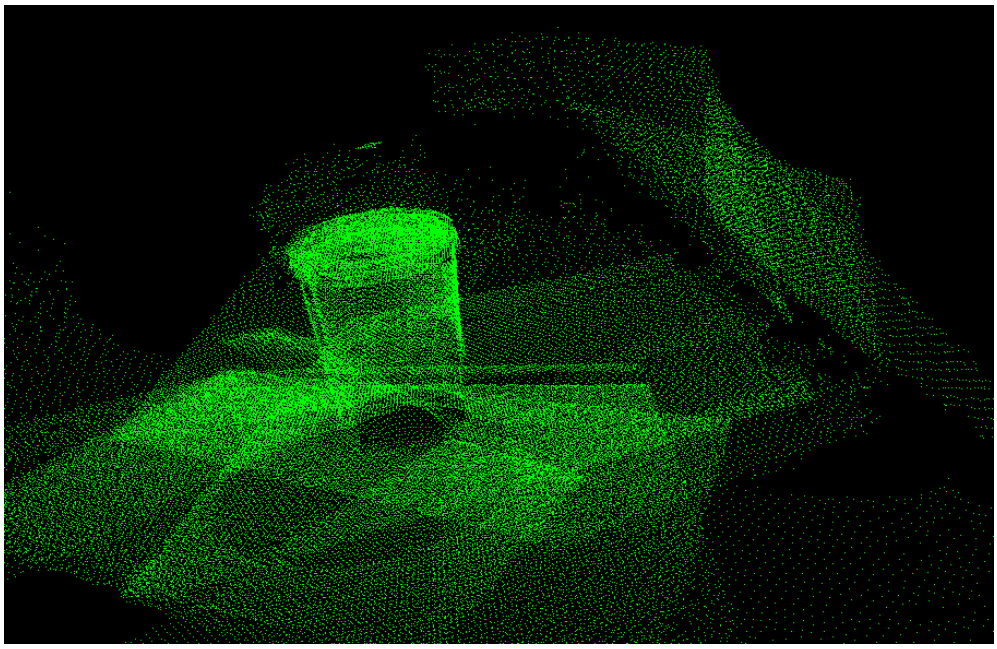

figura 1.1: Point Cloud di un bidone Conico

1.3.1 Primo prototipo

Il primo prototipo realizzato risponde all'esigenza di catturare e salvare in formato leggibile da un *render* grafico i dati forniti dal sensore di profondità. Nella sua semplicità ha dato modo allo studente di testare la stabilità delle *API* e produrre della documentazione interna che riportava quali fossero i metodi da utilizzare e quali fossero invece poco stabili, sperimentali o addirittura non ancora implementati dal produttore.

1.3.2 Secondo prototipo: Cloude

Affrontare la discrepanza tra coordinate assolute e coordinate relative

Un solo *Point Cloud* non è sufficiente a ricostruire un oggetto. Ovviamente il dispositivo, registrando la nuvola di punti inquadrata in un determinato istante, riesce a rilevare solamente i punti che "riesce a vedere": quelli presenti nella parte posteriore dell'oggetto scansionato non possono essere "visti" e conseguentemente nemmeno misurati. Se si vuole avere una ricostruzione completa e non solamente di una facciata è necessario prendere più rilevazioni ed integrarle.
Le immagini in figura 1.2 mostrano il *Point Cloud* che descrive la parte anteriore di una scatola rettangolare; dato che la ripresa è stata effettuata da di fronte ed in alto solo le facce superiore ed anteriore sono state memorizzate, mentre delle altre non si hanno dati. I contorni sono stati evidenziati successivamente per permettere una migliore comprensione della forma.

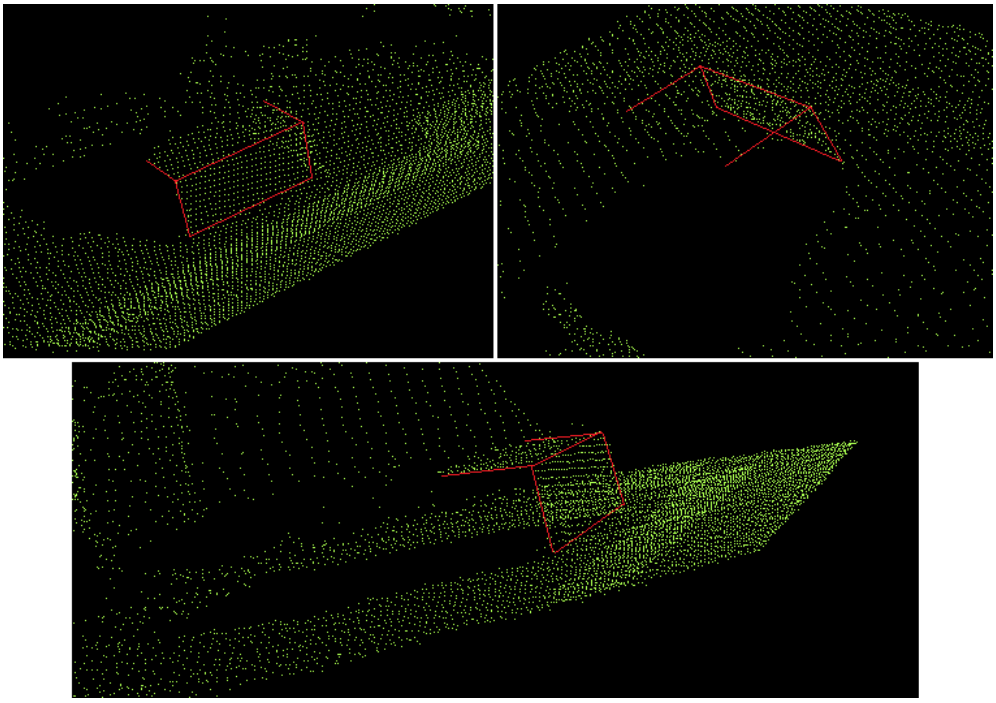

figura 1.2: Un singolo *Point Cloud*

Approccio

Il principale scopo di questo prototipo, denominato *Cloude*, è esplorare una via che porti alla sovrapposizione di diversi *Point Cloud* parziali in una unica nuvola di punti che rappresenti l'oggetto nella sua totalità.
L'approccio che ha portato alla nascita di questo prototipo è il seguente:

- Permettere all'utente di scattare alcune "foto" all'oggetto, quindi di rilevare diversi *Point Cloud*, questi ultimi però sono tutti parziali, ovvero rappresentano solo una facciata.

- Associare ad ogni cattura di un *Point Cloud* parziale la posizione e la rotazione del dispositivo.
- Usare la posizione relativa al *Point Cloud* per traslare e ruotare lo stesso punto a punto, trasformando così le coordinate da relative ad assolute.
- Le nuvole di punti risultanti sono tutte scritte in coordinate relative allo stesso sistema fissato di assi cartesiani, e sono quindi sovrapponibili le une con le altre.

La sovrapposizione di tutti i *Point Cloud* fornisce una ricostruzione coerente dell'oggetto ripreso che è il prodotto finale di *Cloude*.

1.3.3 Terzo prototipo: Samba

Il prototipo precedente generava delle ricostruzioni riconoscibili, ma piuttosto imprecise. Una analisi dello stesso ha fatto emergere diverse criticità che sono state documentate, assieme alle possibili soluzioni, all'interno di un documento descrittivo. Quest'ultimo è stato alla base dello sviluppo di *Samba*.
Di seguito verranno descritti i vari difetti che affliggono *Cloude* e come essi siano stati risolti in *Samba*.

Eccessiva complessità dell'elaborazione

Cloude sfrutta un metodo delle librerie *Tango* che trasforma le coordinate di un singolo punto in coordinate assolute fruttando la posizione relativa a cui si trovava il dispositivo, permette di scrivere poco codice, ma ha una elevata complessità. Ciò comporta un sensibile rallentamento dell'elaborazione dei *Point Cloud*. Un *cloud* medio conta intorno ai 9000 punti e con questo approccio richiede mediamente 1,5-2 secondi per essere completamente elaborato, tempo non accettabile per lo scopo per cui l'applicazione è pensata.
In *Samba* è stato cambiato radicalmente approccio:

- Ad ogni *Point Cloud* viene associata una matrice di trasformazione e non la posizione stessa.
- In questo modo è sufficiente moltiplicare ogni punto (vettore) per la matrice, che viene calcolata una sola volta per ogni *Point Cloud*.

Si sono così ridotti i tempi di elaborazione da 1,5-2s a circa 200ms (sullo stesso dispositivo).

Bassa qualità delle ricostruzioni

Nelle ricostruzioni generate da *Cloude* gli oggetti appaiono deformati, spesso i vari *Point Cloud* non si sovrappongono correttamente generando fenomeni di *ghosting*, talvolta rendendo addirittura irriconoscibile l'oggetto.
Questo è dovuto ad una scorretta stima della posizione del dispositivo, che induce il calcolo di una erronea matrice di trasformazione, e quindi ad un errato posizionamento delle nuvole di punti all'interno dello spazio.
Il fenomeno in questione è detto *"drifting"*: i *device Tango*, esattamente come le più comuni applicazioni in realtà aumentata, utilizzano la tecnica del *Motion Tracking* che consiste nel calcolare la propria posizione frequentemente ed in maniera relativa alla

coordinate acquisite nella stima precedente. Per quanto queste stime siano estremamente precise generano una catena di piccoli errori che sommati tra loro molto presto portano ad una importante discrepanza tra la posizione stimata dal dispositivo e quella reale. Ad esempio partendo da una determinata posizione e camminando in cerchio è praticamente impossibile che la traiettoria stimata passi nuovamente per il punto di partenza. Ciò è un limite fisico dei dispositivi, ed è nella pratica impossibile da

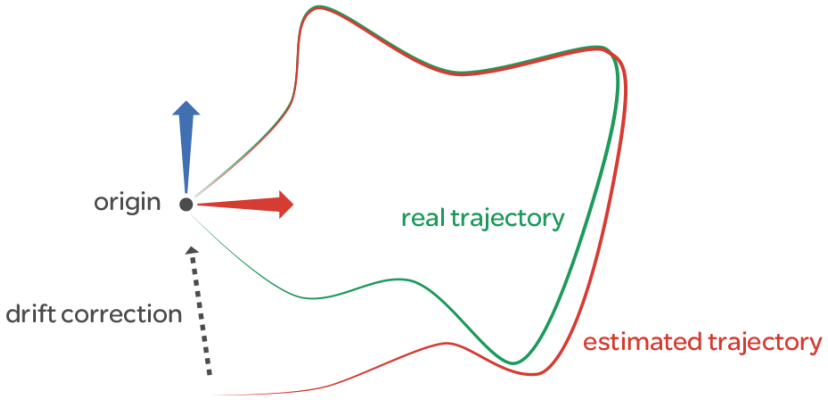

figura 1.3: Motion Tracking

eliminare, in quanto sarebbe necessario azzerare completamente gli errori relativi.
La tecnologia *Tango* però fornisce un altro meccanismo di localizzazione: l'*Area Learning*. Le applicazioni ideate per questo tipo di dispositivi infatti hanno la possibilità di mantenere memoria degli spazi che visitano, e successivamente usare queste informazioni per localizzarsi.
Il meccanismo è piuttosto simile a quello della memoria spaziale umana: una persona portata bendata all'interno di un edificio sconosciuto, una volta liberata, non avrà alcun mezzo per intuire dove si trovi; se invece la stessa persona fosse condotta all'interno della propria abitazione, alla prima sbirciata noterebbe qualche particolare che le farebbe immediatamente recuperare l'orientamento.
Allo stesso modo il *tablet* è in grado memorizzare alcune *features* all'interno dell'ambiente ed usarle come faro per la triangolazione.
Memorizzare completamente un ambiente tuttavia è un'operazione che richiede parecchio tempo e costringe l'utente a muoversi per diversi minuti inquadrando tutti i dettagli del luogo dove si trova. *Samba*, per rendere l'applicazione maggiormente responsiva e più vicina alle esigenze dell'utenza, adotta un approccio ibrido detto *Drift Correction*: inizialmente è richiesto all'utente di inquadrare per una ventina di secondi l'ambiente, in maniera da permettere al *tablet* di memorizzarlo sommariamente, successivamente il *Motion Tracking* è usato per piccoli spostamenti ma viene corretto non appena venga inquadrata qualcuna delle (poche) *features* salvate. In *background*, il processo di *Area Learning* continua, memorizzando sempre nuovi dettagli e conseguentemente aumentando sempre più il livello di conoscenza della stanza in cui ci si trova.

Dimensioni eccessive dei file, ridondanza dei punti sovrapposti

Data la grande mole di punti registrati dai sensori di profondità i *file* contenenti le ricostruzioni generati da *Cloude* sono di dimensione eccessiva, anche più di 10Mb una decina scatti. Considerando che idealmente gli scatti da riprendere potrebbero essere molti e spesso dovranno essere inviati al *Server* tramite connessione a consumo appare subito evidente la necessità di tenere sotto controllo la dimensione di questi *file*.
Inoltre c'è una grossa ridondanza di punti: è comune caso d'uso che una stessa zona venga inquadrata in più scatti, quindi tali *Point Cloud* ruotati ed uniti presenterebbero molti punti con (circa) le stesse coordinate e semplicemente sovrapposti, quindi senza dare alcuna informazione aggiuntiva.
Samba risolve questo problema utilizzando un leggero *voxeling*, ovvero suddividendo lo spazio in cubi o *voxel* di lato prefissato e registrando quali sono i *voxel* che contengono i punti della nuvola. Scegliendo una opportuna definizione, ovvero una opportuna dimensione dei *voxel*, si può ottenere una ricostruzione comunque con un buon livello di dettaglio, ma priva di ridondanza dei punti e quindi meno pesante in termini di memoria.
Dopo diverse prove sperimentali è stata scelta una dimensione dei *voxel* pari a 10 millimetri: una definizione più grossolana porterebbe a perdere troppa informazione, mentre *voxel* più piccoli non portano alcun effettivo miglioramento nella qualità dell'immagine.
Si veda la figura 1.4 per un confronto di alcuni livelli di *voxeling*.

1.3.4 Perfezionamento di Samba, il prototipo finale

Samba, nella sua prima realizzazione, soddisfaceva completamente gli obiettivi fissati per quanto riguarda rilevazione dei punti e ricostruzione degli oggetti, tuttavia è risultato carente nell'interfacciarsi con l'utente.
Quindi è stato pianificato un ulteriore ciclo di raffinamento che ha portato al prototipo finale. Segue una breve lista delle principali migliorie.

Preview dell'inquadratura

Inizialmente l'interfaccia mostrava la sola *preview* della fotocamera a colori, ma essa si è rivelata non sufficiente per far comprendere correttamente all'utente ciò che verrà ricostruito. Oltre a non fornire una chiara idea dei punti che verranno registrati non da la possibilità all'utente di controllare se la ripresa che sta per registrare sia "buona" oppure no.
I *Point Cloud* vengono ricostruiti solamente con i dati del sensore di profondità, il quale sfrutta la tecnologia *infrared* e per questo è soggetto a tutti i limiti fisici di quest'ultima; sono emerse, infatti, grosse difficoltà nel misurare i punti

- di una superficie molto scura.
- di una superficie riflettente o particolarmente lucida.
- all'interno di stanze con illuminazione scarsa o assente.
- all'esterno con luce solare particolarmente forte.

A volte il problema è insormontabile e la ricostruzione non potrà avvenire con successo, altre è sufficiente trovare una buona posizione per permettere al sensore di effettuare le misurazioni. Questo rende necessario mostrare sullo schermo non solo quello che

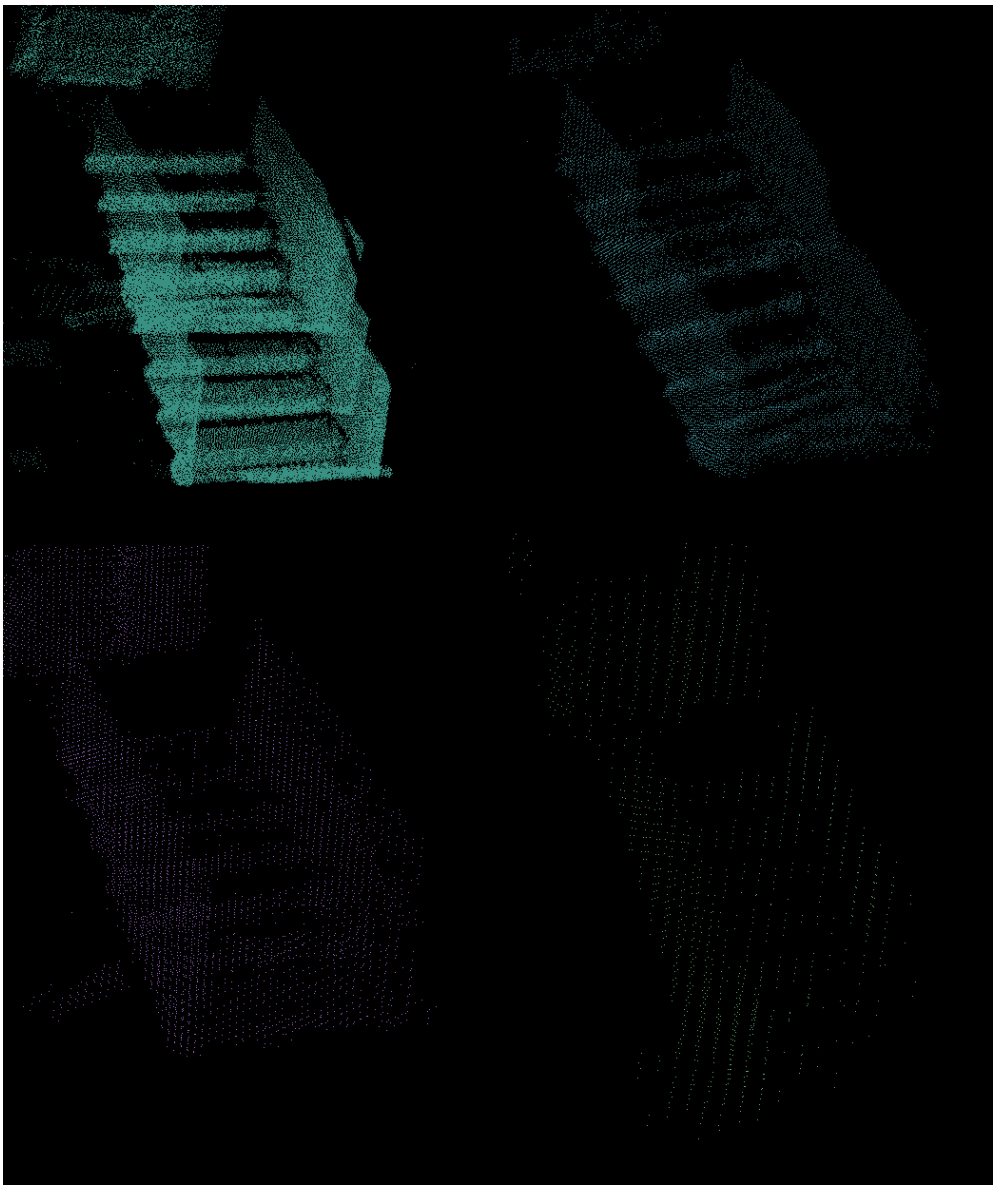

figura 1.4: Rappresentazione tramite *Point Cloud* della stessa scalinata usando *voxeling* a 10, 25, 50, 100 millimetri.

"vede" la fotocamera, ma anche quello che "vede" il dispositivo *infrared*.
A questo scopo è stata parzialmente riusata una applicazione di prova fornita sotto licenza *Open Source* dalla *Google*[1] e che utilizza la libreria grafica *Rajawali*[2].
Il *render* in questione è molto semplice ma efficace ed aumenta il valore aggiunto dell'applicazione dandole un aspetto gradevole e permettendo all'utente di avere aggiornamenti in tempo reale sul *Point Cloud* inquadrato.

[1] https://github.com/googlesamples/tango-examples-java/tree/master/java_point_cloud_example
[2] https://github.com/Rajawali/Rajawali

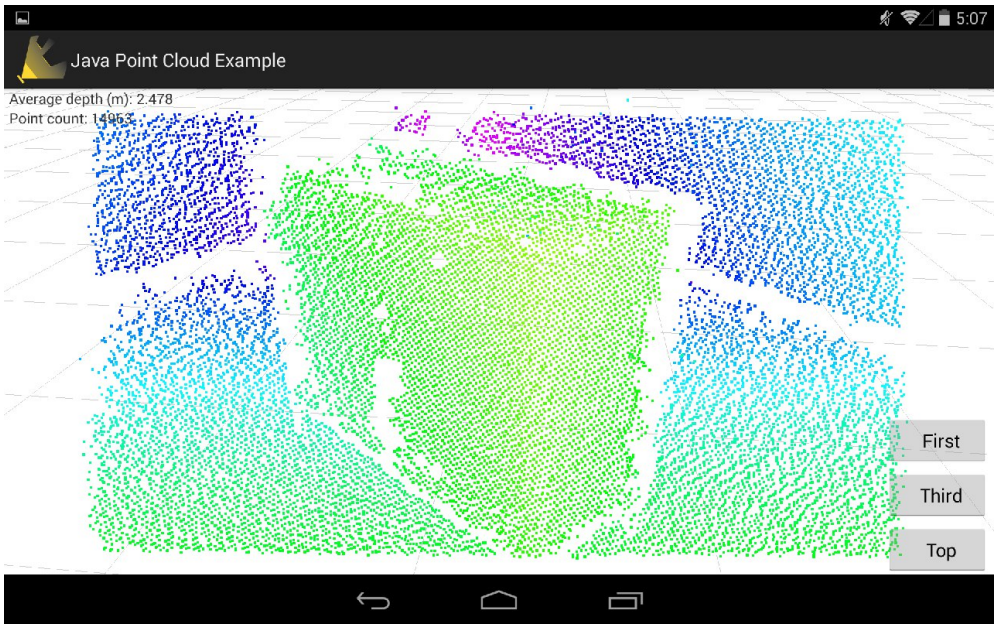

figura 1.5: Il render di `java_point_cloud_example` mentre viene inquadrata una cassettiera

Visualizzazione su tablet del *Point Cloud* ricostruito

Con il *render* descritto nel punto precedente è possibile rappresentare ciò che "vede" il sensore di profondità in tempo reale. Questo tuttavia è diventato presto non più sufficiente: mano a mano che si effettua la ripresa, in *background* viene calcolata la ricostruzione dell'oggetto in analisi, non poterlo visualizzare sul *tablet*, ma solo lato *server* appare quantomeno frustrante e soprattutto non da la possibilità all'utente di avere una idea dello stato in cui è la ricostruzione.
Per questo si è pensato di sfruttare lo stesso meccanismo di *rendering* per far visualizzare all'utente la ricostruzione che sta effettuando. É quindi stata aggiunta la possibilità di alternare tra la visualizzazione in tempo reale e la visualizzazione dell'intera ricostruzione generata.

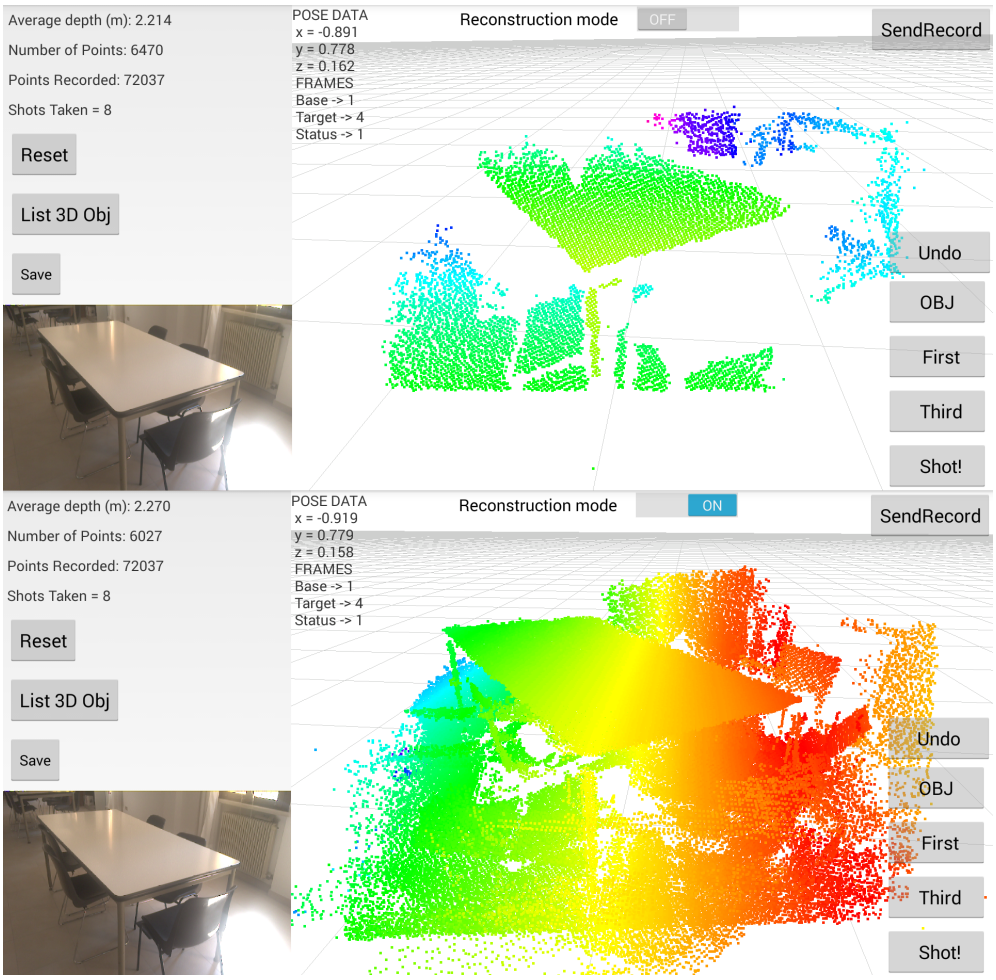

figura 1.6: Le due modalità del render di *Samba*: in alto la visualizzazione in tempo reale, in basso la visualizzazione della ricostruzione.

Aggiunta operazioni di undo

Nonostante gli sforzi per mantenere una alta qualità delle riprese qualcuno dei *Point Cloud* catturati continua a presentare grossi difetti (come errato posizionamento, deformità, grossa presenza di rumore etc). Importante miglioria effettuata in questa fase è stata l'inserimento della possibilità di annullare un certo numero di operazioni; in questo modo le riprese che contengono dei difetti possono essere scartate e ripetute.

Istruzioni per l'utente

Samba, come tutte le applicazioni *Tango*, introduce nuove azioni che l'utente deve compiere per mettere il dispositivo nella condizione di operare al meglio; ed è compito degli applicativi istruire l'utente sul comportamento da tenere.
Ad esempio, durante l'avvio dell'*app*, l'utente deve avere cura di mantenere il *tablet* in posizione verticale ed il più possibile fermo. Questa, come la maggior parte delle indicazioni, possono essere notificate all'utente tramite un *framework*, *TangoUx*, messo

a disposizione dal produttore in grado di integrare segnali e notifiche all'interno del ciclo di vita dell'applicazione stessa.

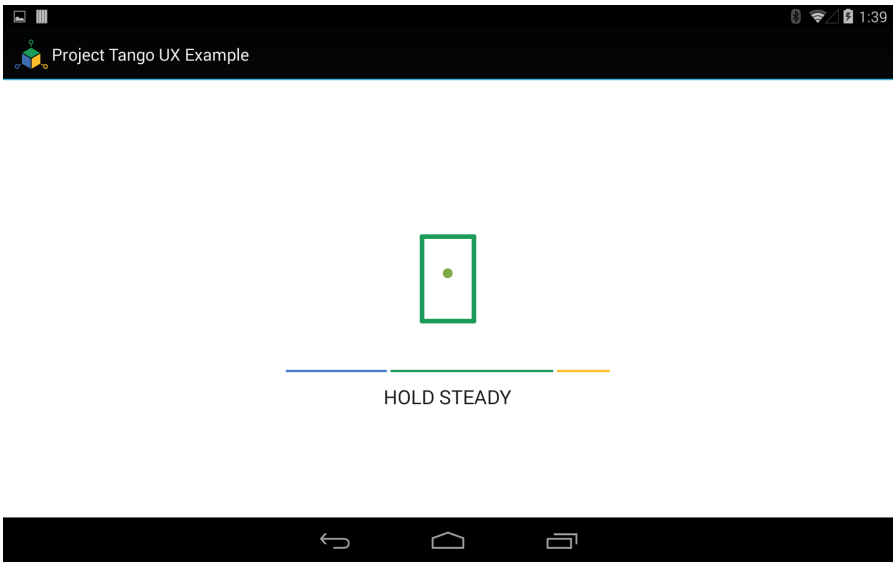

figura 1.7: Una notifica *full-screen* di *TangoUx*

Fanno eccezione le istruzioni che devono essere date all'utente durante la fase di localizzazione. Le *API Tango* non forniscono alcun aiuto per stabilire se il dispositivo si sia orientato o meno. È stato quindi necessario studiare delle euristiche capaci di intuire se il dispositivo "si senta o meno a proprio agio".

Per comprendere la soluzione è necessario prima capire alcuni fatti riguardanti il modo in cui i dispositivi *Tango* attuano l'*Area Learning*.

def 1. *Una* posizione *(o* Pose*) P è una tripletta di valori x_P, y_P e z_P che rappresenta la distanza con segno tra due* Frame Of Reference*.*

def 2. *I* Frame Of Reference *sono dei punti speciali nello spazio, usati da* Tango *come riferimento. Sono di seguito elencati:*

- `COORDINATE_FRAME_DEVICE`*: è la posizione attuale del dispositivo. Ovviamente non è un punto fisso, ma cambia nel tempo a seconda di come si muove il dispositivo. Si chiami questo punto D.*
- `COORDINATE_FRAME_START_OF_SERVICE`*: è il punto in cui si trovava il dispositivo quando è stata avviata l'applicazione, è calcolato ad ogni avvio e rimane costante fino a quando l'applicazione rimarrà attiva. Si chiami questo punto S.*
- `COORDINATE_FRAME_AREA_LREARNING`*: durante il processo di* Area Learning*, il sistema* Tango *fissa nell'ambiente un sistema di assi cartesiani proprio dell'ambiente. Questo punto, O, ne è l'origine. Idealmente dovrebbe essere costante.*

Ad esempio considerando la distanza tra il *frame* del dispositivo e l'origine del sistema di assi cartesiani fissati da *Tango* durante l'apprendimento dell'ambiente si avrà:

$$P_{O \to D} = D - O$$

quindi

$$P_{O \to D} = \begin{pmatrix} x_D \\ y_D \\ z_D \end{pmatrix} - \begin{pmatrix} x_O \\ y_O \\ z_O \end{pmatrix} = \begin{pmatrix} x_D - x_O \\ y_D - y_O \\ z_D - z_O \end{pmatrix}$$

Questa *Pose* $P_{O \to D}$ può quindi essere usata per tracciare i movimenti del *tablet* rispetto all'origine degli assi del piano cartesiano fissato al termine dell'*Area Learning*.

Quindi è solamente necessario stabilire **quando** O sarà correttamente identificato. Per farlo si prenda in considerazione la *Pose* $P_{S \to O}$, ovvero la distanza con segno tra il punto in cui è stata avviata l'applicazione S ed O.
S come detto è costante, mentre O varierà fino a quando non verrà localizzato; inizialmente si avrà $O = S$ in quanto non si ha alcuna informazione utile per determinare O.
Nel grafico in figura 1.8 sono rappresentate le coordinate x, y e z di $P_{S \to O}$ per le prime 9000 stime di posizione, ciò copre un lasso temporale di circa un minuto e mezzo a partire dall'avvio dell'applicazione. Il variare di questo genere di *Pose* è stato diviso in tre fasi:

- Inizialmente la distanza è molto prossima a zero. Prima che avvenga la localizzazione il sistema *Tango* non ha altra scelta che posizionare O nelle stesse coordinate di S.

- Successivamente avverrà un brusco cambiamento nella distanza tra questi due punti perché tutto d'un tratto il sistema riconoscerà qualche *feature* e sposterà immediatamente l'origine degli assi. Questa fase non è ancora stabile in quanto l'ambiente è ancora in fase di riconoscimento e l'origine O verrà traslata spesso.

- Una volta riconosciuto correttamente l'ambiente (nella seconda metà del grafico) si può notare che tutte le coordinate si stabilizzano ad una distanza quasi fissa. Da questo momento in avanti si avrà una buona localizzazione e l'utente potrà iniziare la rilevazione.

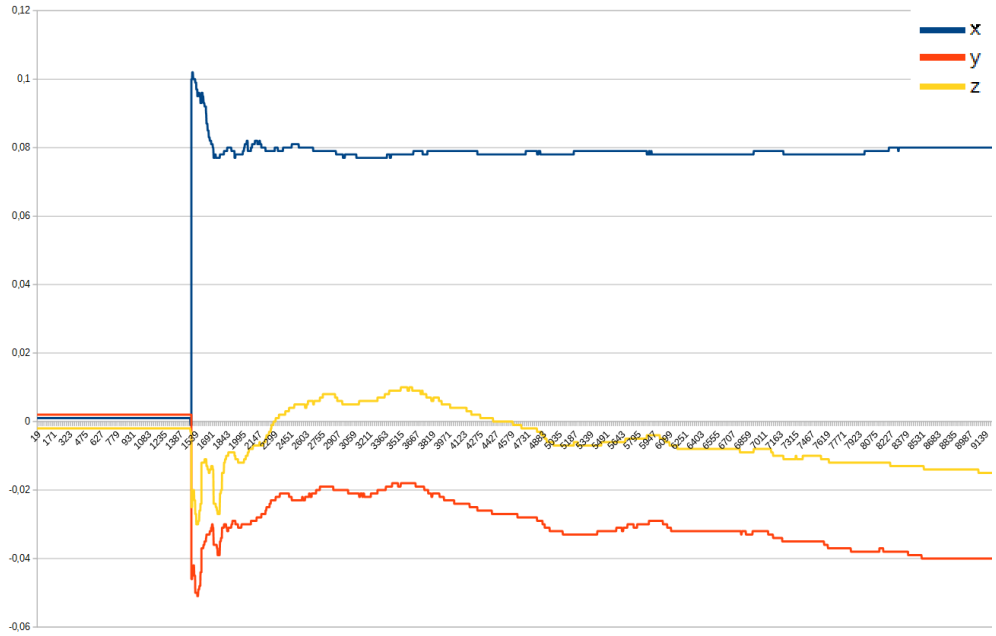

figura 1.8: Grafico della coordinate di S rispetto ad O durante la fase di inizializzazione della *Drift Correction*

La distanza tra O ed S dovrebbe essere idealmente costante. Si è notato sperimentalmente, però, che non è così: a causa di limiti fisici del dispositivo essa continuerà a fluttuare entro un paio di centimetri di raggio. Ciò a volte può creare qualche piccolo errore nella ricostruzione, ma per oggetti piuttosto grandi esso è trascurabile.

1.4 Il Prodotto - lato server

L'applicazione lato server si occupa di "pulire" la ricostruzione dagli elementi inutili, come ad esempio pavimento, rumore, oggetti di sfondo e convertirla in un formato portabile.

1.4.1 Point Cloud Library

PCL o *Point Cloud Library* è una libreria *Open Source* e *Large Scale* per l'elaborazione di 2D e 3D di immagini e *Point Cloud*. Fornisce diversi filtri ed algoritmi in grado di risolvere molti dei problemi che sono stati riscontrati per quando riguarda l'elaborazione delle ricostruzioni.
L'applicativo lato server fa vasto uso di questa libreria.

1.4.2 Generazione mesh

Per poter elaborare facilmente la ricostruzione ed ottenere un modello portabile è necessario trasformare la nuvola di punti in una *mesh*, ovvero una rappresentazione di un oggetto 3D che consiste in un insieme di facce poligonali, solitamente semplici triangoli. Inoltre tutti i più diffusi formati per oggetti tridimensionali come `obj` e `ply`

sono in grado di rappresentare solamente *mesh*.
Per ottenere una buona riproduzione dell'oggetto l'applicativo applica diversi filtri:

- **sparse filter / filter radius**: vengono eliminati i punti isolati ed i punti ad una eccessiva distanza dal centro, i quali sono quasi sempre frutto di errori nelle misurazioni.
- **filter ground**: viene eliminato il pavimento.
- **voxel filter**: viene effettuata una ulteriore operazione di voxeling allo scopo di ridurre la mole di calcoli e regolarizzare il *Point Cloud*.
- **cluster extractor**: si cerca di suddividere la ricostruzione nei vari oggetti da cui è composta, dopodiché si mantiene solamente l'oggetto (o *cluster*) che si trova più al centro.
- **meshing**: i punti rimanenti sono solamente quelli dell'oggetto in analisi. Essi vengono usati per generare la *mesh*.

Al termine di questo processo si ottiene l'oggetto 3D, che può essere convertito nel formato preferito. L'immagine sottostante mostra il processo di elaborazione dell'immagine di una scatola rettangolare. Si possono osservare, da in alto a sinistra ad in basso a destra: la ricostruzione fornita dal *tablet*, il *Point Cloud* risultante da *sparseFilter* e *filterRadius*, l'output della operazione di rimozione del pavimento, il risultato del *voxelFilter*, il *cluster* estratto ed infine la *mesh* dell'oggetto.

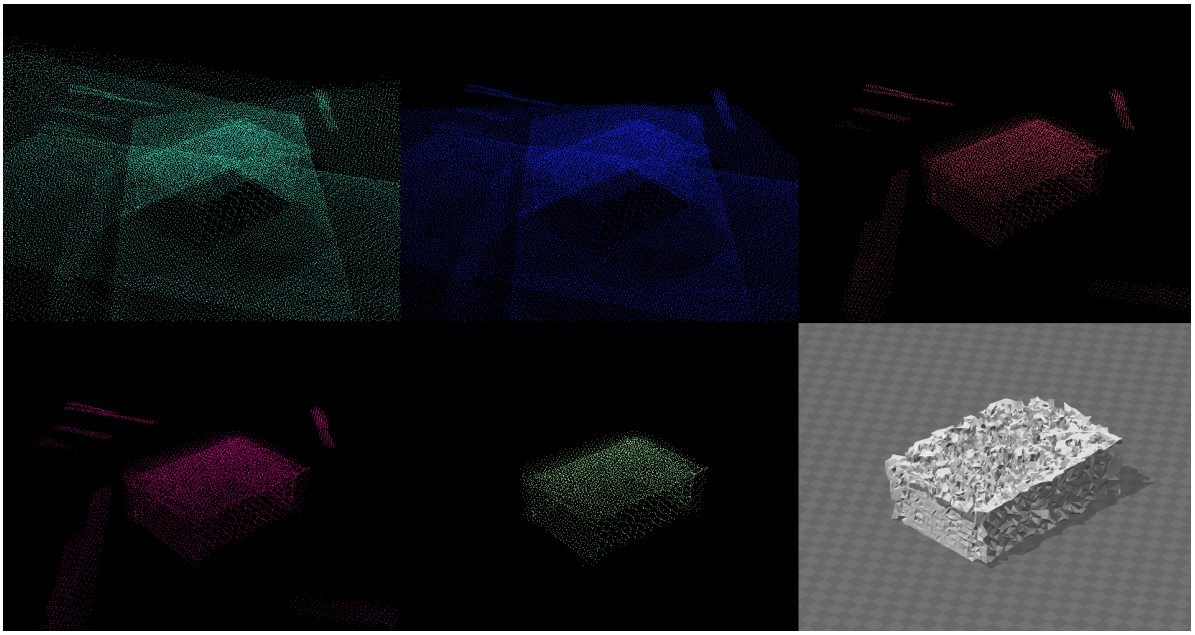

figura 1.9: Input ed Output del processo di meshing

1.4.3 Calcolo del volume

Una volta ottenuta la *mesh* il calcolo del volume è piuttosto immediato. A questo fine è stato sfruttato il risultato di una pubblicazione di *Cha Zhang* e *Tsuhan Chen* dal

titolo *EFFICIENT FEATURE EXTRACTION FOR 2D/3D OBJECTS IN MESH REPRESENTATION*[3]. Il trucco è calcolare, per ogni triangolo che compone la *mesh*, il volume con segno del tetraedro che ha il triangolo stesso come base e il quarto vertice in un punto fissato, scelto internamente alla *mesh*, per evitare eventuali problemi di instabilità numerica. Il segno del volume è dato dalla direzione della normale al piano del triangolo. Questi volumi, sommati tra loro, restituiscono il volume convesso della *mesh*.

1.5 Lavoro svolto

Allo studente è stato richiesto di focalizzarsi sullo sviluppo dell'applicazione *mobile*, lasciando la realizzazione dell'applicazione lato server ad un altro tirocinante.
Il focus del tirocinio si è quindi quasi completamente concentrato sulla progettazione e lo sviluppo di una applicazione in grado di riconoscere e ricostruire oggetti nell'ambiente.

1.6 Organizzazione del testo

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato in appendice B;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- i nomi delle classi o delle componenti del sistema, sono evidenziati in `monospace`, se c'è rischio di ambiguità.

[3] **site: http://research.microsoft.com/en-us/um/people/chazhang/publications/icip01˙ChaZhang.pdf**.

Capitolo 2

Processi e strumenti

La buona definizione dei processi produttivi è fondamentale per evitare il rischio di fallimento.

2.1 Processo sviluppo prodotto

In ambito aziendale si è scelto di provare, per questo progetto, un processo di sviluppo *software* basato sulla filosofia *Lean*.
Dato che l'obiettivo principale era fornire un prototipo ci si è limitati solamente alle tre fasi iniziali dello sviluppo di *Lean*, ovvero *Kick-Off*, *Concept Preview* e *Product Prototype*.

2.1.1 Kick-Off

Questa è la prima fase dello sviluppo *software*, coincide con la prima riunione ufficiale del team di progetto, aperta anche agli *Stakeholder*.
Si pone lo scopo di iniziare la fase l'*allestimento* e l'*avviamento* in cui viene determinata la natura e lo scopo del progetto.

2.1.2 Concept Preview

Fase in cui è reso disponibile un primo campione di prova del prodotto, detto *concept*. Esso può essere incompleto e affetto da errori, ma deve essere in grado di dimostrare agli *stakeholder* le caratteristiche principali che avrà il prodotto finito.
Esso è soggetto a un riesame che ha lo scopo di valutare se è in linea con gli obiettivi definiti nella *Value Proposition*, la *milestone* non può essere raggiunta senza che questo riesame abbia esito positivo.
Questa fase coincide anche con l'inizio della progettazione, che deve essere portata avanti fino ad un livello di dettaglio ritenuto opportuno dal *team*.

2.1.3 Product Prototype

Fase in cui è messo a disposizione il primo prototipo del nuovo prodotto, completo nelle sue funzioni (sviluppo finito) ma non ancora messo a punto mediante verifiche

e correzioni, per garantirne funzionalità e prestazioni. Il prototipo deve essere ad un stato tale da poter essere dato in valutazione agli *stakeholder*.
Esso è soggetto a un riesame che ha lo scopo di valutare se è in linea con gli obiettivi definiti sia *Value Proposition* che nella *Requirements Specification*, la *milestone* non può essere raggiunta senza che questo riesame abbia esito positivo.
Questa fase coincide con il termine della fase di progettazione e l'inizio della fase di esecuzione, cioè l'insieme dei processi necessari a soddisfare i requisiti del progetto.

2.1.4 Fasi successive

Le fasi successive, ovvero *Product Design Freeze* e *Start Of Production*, possono essere avviate nel futuro a partire dal *Product Prototype* se ciò verrà ritenuto opportuno dall'azienda.

2.2 Strumenti

L'azienda ha lasciato grande libertà riguardo agli strumenti da utilizzare per questo progetto, quindi essi sono stati fissati inizialmente e successivamente incrementati al crescere delle necessità.

2.2.1 Codice

Segue la lista degli strumenti utilizzati per la codifica.

- **Java**[1]: Il linguaggio preferito per l'applicazione lato *tablet*. È stato scelto seguendo le *Best Practice* dello sviluppo *Android*.
- **C++**[2]: Il linguaggio preferito per l'applicazione lato *Server*. È stato scelto perché tutte le più diffuse librerie per l'elaborazione dei *Point Cloud* sono disponibili in questo linguaggio.
- **PCL, Point Cloud Library**[3]: Una delle librerie di maggior rilievo nel campo della *Computer Vision*, mette a disposizione innumerevoli funzionalità per elaborazione ed ottimizzazione dei *Point Cloud*.
- **Tango API**[4]: Le *API* ufficiali per lo sviluppo di applicazioni *Tango*.
- **PHP**[5]: Il linguaggio usato per ricevere ed inviare le richieste *http* quando si necessita comunicazione tra *Server* e dispositivo.

2.2.2 IDE ed editor

Segue la lista degli ambienti per la codifica utilizzati durante il progetto.

- **Android Studio**[6]: L'*IDE* ufficiale per le applicazioni *Android*.
- **QT**[7]: L'*IDE* scelto per lo sviluppo del codice *C++*.

[1] https://www.java.com
[2] www.cplusplus.com
[3] http://pointclouds.org
[4] https://developers.google.com/tango/apis/overview
[5] https://secure.php.net
[6] https://developer.android.com/studio/index.html
[7] https://www.qt.io/ide

- **Sublime Text 2**[8]: L'*editor* di testo usato per scrivere gli *script php*.

2.2.3 Framework

Segue la lista dei *Framework* usati per durante il tirocinio.

- **Gradle**[9]: È stato usato come *tool* di *build* per tutta l'applicazione *Android*.
- **Rajawali3D**[10]: È stato usato come *framework* grafico per la realizzazione del *render*.
- **OkHttp**[11]: È stato usato come *framework* di riferimento per le richieste *http*, come indicato nella *Android Best Practices*.
- **TangoUx**[12]: *Famework* messo a disposizione dalla *Google* assieme alle *API Tango*. È stato usato per gestire le notifiche all'utente relative ai comportamenti che deve tenere per permettere il buon funzionamento del dispositivo e dei sensori.
- **Jni**[13]: Questo *framework* è stato solamente impostato per sviluppi futuri, permette di implementare metodi *Java* nativamente in *C/C++*. Non è usato nel prototipo presentato.

[8] https://sublimetext.com/2
[9] https://gradle.org
[10] https://github.com/Rajawali/Rajawali
[11] http://square.github.io/okhttp
[12] https://developers.google.com/tango/ux/ux-framework
[13] http://docs.oracle.com/javase/7/docs/technotes/guides/jni

Capitolo 3

Studio di fattibilità ed analisi dei rischi

Iniziare, quasi da zero, un progetto così impegnativo basandosi su di una tecnologia al limite dell'essere sperimentale espone a gravi rischi di fallimento. Per ciò in questa fase sono state investite molte risorse.

3.1 Introduzione al progetto

Data la natura innovativa del progetto è stato necessario produrre diversi prototipi ed effettuare l'*Analisi dei Rischi* e lo *Studio di Fattibilità* in diverse fasi.
Questo approccio è stato estremamente utile per far emergere rischi dovuti sia alla non piena maturità delle *API*, sia ai limiti fisici del dispositivo in dotazione.

3.2 Studio di fattibilità

Prima di iniziare il progetto è stato effettuato un accurato studio di fattibilità basato sulla ricerca di progetti con funzionalità simili e sulla lettura delle numerose discussioni e pubblicazioni presenti nel *web* a riguardo di *Project Tango*.
Da esso è emersa la presenza di diverse applicazioni dotate di sottoinsiemi delle funzionalità che si stanno cercando di sviluppare.

3.2.1 Applicazioni per il meshing

Sul mercato sono disponibili diverse applicazioni in grado di effettuare il *meshing* 3D di ambienti, ma non di registrare *Point Cloud*. Alcune di queste sono:

- **Tango Constructor**[1]: applicazione rilasciata dal produttore sotto licenza proprietaria.
- **RTAB-Map**[2]: Applicazione della *introlab*[3] rilasciata sotto licenza *BSD* che fa uso delle librerie native offerte dalla *Google*.

[1] https://developers.google.com/tango/tools/constructor
[2] http://introlab.github.io/rtabmap/
[3] https://introlab.3it.usherbrooke.ca/mediawiki-introlab/index.php/Main_Page

- **Esempio Google di ricostruzione per Unity**[4]: Esempio rilasciato dal produttore sotto licenza *Apache* per unity e con codice in `C#`.

Queste applicazioni sono ottime in quanto a qualità della ricostruzione, quasi tutte forniscono anche una ottima memorizzazione delle *texture*, ma lavorare con le *mesh* per quanto riguarda rimozione di oggetti di sfondo, pavimento etc appare piuttosto complesso.

3.2.2 Applicazioni per il Motion Tracking / Drift Correction

Ci sono diversi esempi che dimostrano la buona qualità del *Motion Tracking* e della *Drift Correction*. Eccone alcuni:

- **Tango Blaster**[5]: Una semplice avventura grafica che permette all'utente di spostare il proprio avatar in un ambiente completamente virtuale semplicemente spostando il dispositivo. Questa semplice applicazione dimostra che l'*Hardware Tango* è in grado di tracciare la propria posizione, così come di distruggere orde di robot.

- **java_motion_tracking_example**[6]: Un esempio fornito sotto licenza *BSD* dalla *Google* che mostra graficamente la differenza tra la posizione stimata usando solamente il *Motion Tracking* e quella corretta con metodi di *Area Learning*.

Queste due applicazioni dimostrano che è possibile usare le *API Tango* per ricavare un buon tracciamento della posizione.

3.2.3 Applicazioni per la registrazione di punti

Punto centrale di questo studio di fattibilità era trovare evidenze della possibilità di registrare un numero consistente di punti e poi sovrapporli per creare una ricostruzione tridimensionale. Tra le applicazioni studiate le due su cui è stata riposta la maggiore attenzione sono le seguenti (entrambe disponibili nella *repository GitHub* di *GoogleSamples*[7]):

- **java_point_cloud_example**: Questa applicazione permette di catturare in tempo reale una nuvola consistente di punti e posizionarli nello spazio relativamente alla posizione del dispositivo; fornisce quindi evidenze riguardo alla possibilità di catturare un numero tale di punti da poter essere usati per ricostruire una facciata di un oggetto.

- **java_point_to_point_example**: Quest'ultimo esempio dimostra invece che i punti possono essere scritti in coordinate "assolute", o meglio rispetto ad un sistema di riferimento fisso indipendente dalla posizione del *device*. Fornisce infatti la possibilità di selezionare due punti tramite la pressione del dito, e poi ne calcola la distanza e rappresenta la retta congiungente nello spazio.

Alla luce dei test sulle applicazioni sopracitate il progetto appare fattibile e quindi è stato possibile dare il via al ciclo di vita del progetto *software*.

[4] https://github.com/googlesamples/tango-examples-unity
[5] https://play.google.com/store/apps/details?id=com.projecttango.tangoblaster
[6] https://github.com/googlesamples/tango-examples-java/tree/master
[7] https://github.com/googlesamples/tango-examples-java/tree/master

3.3 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto ad elaborare delle possibili soluzioni per far fronte a tali rischi.

3.3.1 Rischi generali

1. Immaturità di API/librerie/documentazione
Probabilità: Alta.
Gravità: Media.
Descrizione: Le tecnologie adottate sono innovative e tuttora in fase di sviluppo, molte sono ancora segnalate come *"Sperimentali e soggette a cambiamenti"*. Per questo le librerie usate potrebbero rivelarsi instabili o potrebbero mancare di adeguata documentazione.
Contromisure: Iscrizione ai vari canali di segnalazione e supporto offerti da *Google* per gli sviluppatori, sviluppo di piccoli esempi giocattolo per testare le funzionalità offerte dalle *API* da cui è stata generata della documentazione interna.

2. Limiti fisici del dispositivo
Probabilità: Alta.
Gravità: Media.
Descrizione: Il dispositivo è dotato di sensori infrarossi e sfrutta la riflessione della luce per determinare la distanza dei punti che è in grado di individuare. Superfici riflettenti o molto scure possono compromettere la qualità della misurazione, allo stesso modo situazioni di illuminazione scarsa o troppo intensa.
Contromisure: Accurata analisi della documentazione fornita dal produttore[8], test preventivi nelle situazioni critiche utilizzando una semplice applicazione di prova fornita da *Google*[9].

3. Scarsa conoscenza dello sviluppo Android
Probabilità: Alta.
Gravità: Bassa.
Descrizione: La scarsa conoscenza nello sviluppo di applicazioni *Android* può compromettere la buona riuscita del progetto.
Contromisure: Lo studente si è impegnato a documentarsi a riguardo e ha familiarizzato con la documentazione offerta per gli sviluppatori.

3.3.2 Rischi specifici

Successivamente alla realizzazione del primo prototipo si è ritenuto opportuno incrementare l'Analisi dei Rischi per tenere conto delle nuove incertezze emerse.
Segue la lista dei rischi individuati in questa fase:

4. Difficoltà nel Motion Tracking
Probabilità: Media.
Gravità: Alta.
Descrizione: Determinare la posizione e l'orientamento del dispositivo in maniera

[8] https://developers.google.com/tango/overview/depth-perception
[9] https://github.com/googlesamples/tango-examples-java/tree/master/java_point_cloud_example

assoluta è fondamentale per permettere la ricostruzione dell'oggetto inquadrato. Il *device* fornisce ad intervalli regolari la sua posizione tramite una tripletta di coordinate e la sua rotazione rappresentata come un quaternione. La somma di piccoli errori relativi nella stima della posizione crea un fenomeno detto *drifting* che comporta importanti errori nella stima finale. Questo rischio può portare al fallimento del progetto, in quanto se non opportunamente mitigato renderebbe le ricostruzioni tridimensionali totalmente errate.
Contromisure: Si è per questo deciso di adottare una tecnica denominata *Area Learning*. Il dispositivo quindi riconoscerà alcune *features*, ovvero dei punti fissi, rispetto ai quali determinerà la sua posizione.

5. Necessità di azioni specifiche da parte dell'utente
Probabilità: Alta.
Gravità: Alta.
Descrizione: Tutte le applicazioni che usano la tecnologia *Tango* interagiscono strettamente con i movimenti e la posizione dell'utente. La scarsa diffusione di questa tecnologia fa si che la maggior parte dell'utenza non sia a conoscenza del comportamento che deve tenere. Azioni compiute dall'utente in maniera scorretta possono compromettere il buon funzionamento dell'applicazione.
Contromisure: Tutto lo sviluppo dell'applicazione deve tenere conto di questo fatto. Devono essere fornite chiare informazioni all'utente e si devono studiare soluzioni che non costringano l'*user* ad un comportamento troppo antiintuitivo.

Capitolo 4

Analisi dei requisiti

Un progetto software innovativo pone il progettista dinanzi a molti canti di Sirena che rischiano di indurlo ad investire risorse per scopi futili. Per questo l'Analisi dei Requisiti è stata stilata nelle prime fasi del processo e poi seguita il più scrupolosamente possibile.

4.1 Casi d'uso

Per lo studio dei casi d'uso del prodotto sono stati realizzati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Per ogni *use case* sono stati inoltre riportati:

- Attori Principali
- Precondizioni
- Descrizione/flusso degli eventi
- Postcondizioni

Seguono i diagrammi riguardanti l'applicativo lato *tablet*, ovvero quello di cui lo studente si è occupato di persona.

4.1.1 UC0: Scenario principale

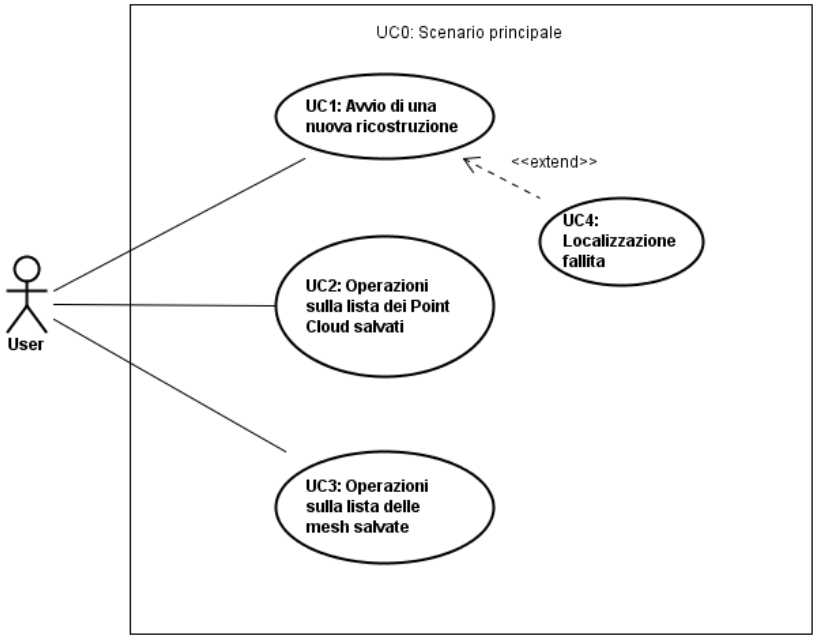

figura 4.1: Use Case - UC0: Scenario principale

Attori Principali: Utente.

Precondizioni: L'utente ha avviato l'applicazione su un dispositivo *Tango* ed ha fornito tutti i permessi necessari, ovvero:

- Area Learning (permesso speciale per dispositivi *Tango*).
- Lettura e scrittura su disco.
- Utilizzo fotocamera.
- Accesso ad internet.

Inoltre deve aver avviato l'applicazione in un ambiente sufficientemente illuminato.

Descrizione: La schermata principale, mentre è immediatamente in atto il processo di localizzazione, mette a disposizione il *render* dei punti e tutti gli strumenti per permettere all'utente di effettuare la rilevazione e di accedere agli altri menù.

Postcondizioni: Il sistema è pronto per permettere una nuova interazione con l'utente.

4.1.2 UC1: Avvio di una nuova ricostruzione

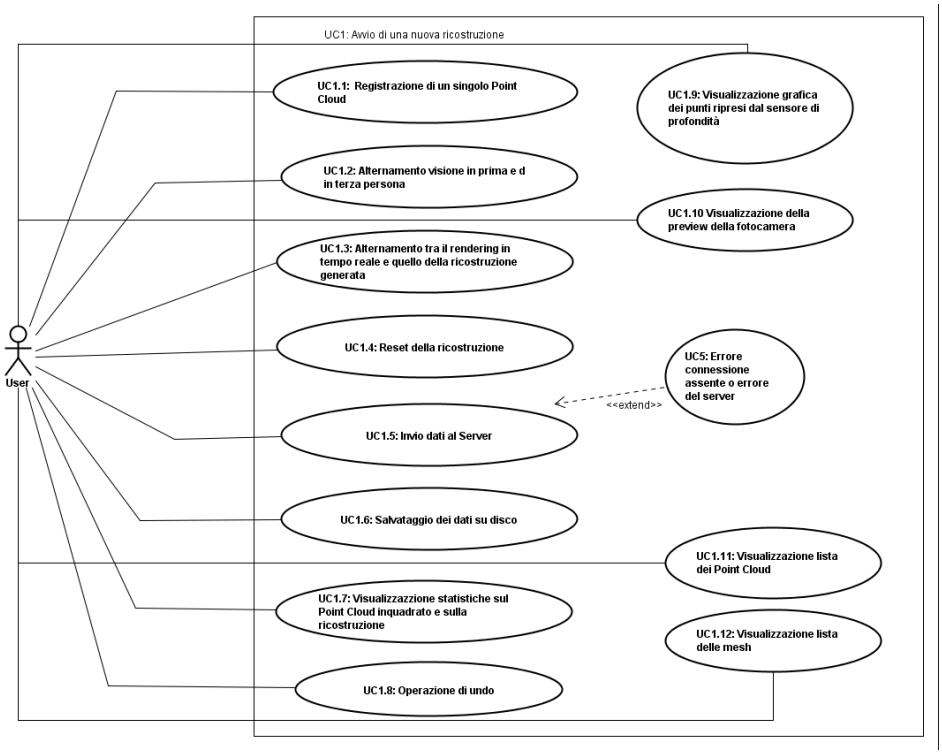

figura 4.2: Use Case - UC1: Avvio di una nuova ricostruzione

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione ed è rimasto in attesa della localizzazione nella schermata principale.

Descrizione: L'utente ha a disposizione tutti gli strumenti per registrare, controllare e perfezionare la ricostruzione dell'oggetto da ispezionare. Inoltre gli sono forniti i tasti per passare alle altre funzionalità.

Postcondizioni: L'utente ha terminato una registrazione inviandola al *Server*, salvandola su disco, oppure scartandola.

4.1.3 UC1.1: Registrazione di un singolo Point Cloud

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale ed intende registrare un nuovo scatto.

Descrizione: L'utente inquadra il soggetto della rilevazione e preme il tasto "Shot".

Postcondizioni: Il sistema ha catturato il *Point Cloud* inquadrato e l'ha aggiunto alla ricostruzione attualmente in corso.

4.1.4 UC1.2: Alternanza visione in prima ed in terza persona

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale e sta osservando il *render*.

Descrizione: L'utente, usando i tasti "First" o "Third" alterna tra la visuale in prima ed in terza persona per il *render*.

Postcondizioni: Il *render* mostra sullo schermo i suoi contenuti nella modalità scelta dall'utente.

4.1.5 UC1.3: Alternanza tra il rendering in tempo reale e quello della ricostruzione generata

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale ed ha già iniziato la rilevazione, avendo quindi salvato in memoria almeno un Point Cloud singolo.

Descrizione: L'utente, usando l'interruttore denominato "Reconstrucion Mode", alterna tra la visualizzazione dei punti attualmente catturati dal sensore di profondità e quella della ricostruzione in corso.

Postcondizioni: Il *render* mostra sullo schermo i contenuti selezionati dall'utente.

4.1.6 UC1.4: Reset della ricostruzione

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale ed ha intenzione di scartare interamente la rilevazione effettuata fino a quel momento.

Descrizione: L'utente premendo sul pulsante "Reset" azzera i punti salvati e rende il dispositivo pronto per una nuova rilevazione.

Postcondizioni: Il dispositivo non ha più alcuna ricostruzione in corso ed è pronto ad iniziarne una nuova.

4.1.7 UC1.5: Invio dati al Server

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale, ha effettuato una rilevazione che lo soddisfa ed intende inviarla al *Server*.

Descrizione: L'utente premendo sul pulsante "Send Data To Server" invia la ricostruzione corrente al *Server* in formato `pcd`.

Postcondizioni: La ricostruzione corrente è stata inviata al *Server*, ma non eliminata dalla memoria. Sarà quindi possibile continuare la rilevazione.

4.1.8 UC1.6: Salvataggio dei dati su disco

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale, ha effettuato una rilevazione che lo soddisfa ed intende salvarla su disco.

Descrizione: L'utente, premendo sul pulsante "Save" ed inserendo un nome per il *file*, salva la ricostruzione corrente su disco in formato `pcd`.

Postcondizioni: La ricostruzione corrente è stata salvata su disco, ma non eliminata dalla memoria. Sarà quindi possibile continuare la rilevazione.

4.1.9 UC1.7: Visualizzazione statistiche

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale.

Descrizione: L'utente può consultare le statistiche riguardanti il *Point Cloud* inquadrato e la ricostruzione nell'angolo in alto a sinistra dello schermo. Tali statistiche sono:

- Numero dei punti attualmente inquadrati.
- Distanza media dei punti attualmente inquadrati.
- Numero degli scatti presi fino a quel momento.
- Numero di punti presenti nella ricostruzione fino a quel momento.
- *Frame Of Reference* e *Status* della rilevazione.
- Posizione x, y e z del dispositivo.

Postcondizioni: L'utente ha visualizzato le statistiche.

4.1.10 UC1.8: Operazione di undo

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, è rimasto in attesa della localizzazione nella schermata principale, ha effettuato uno o più *shot* che ritiene errati o di bassa qualità ed intende scartarli.

Descrizione: L'utente, premendo sul pulsante "Undo", scarta l'ultimo *Point Cloud* registrato.

Postcondizioni: L'ultimo *Point Cloud* registrato viene scartato.

4.1.11 UC2: Operazioni sulla lista dei Point Cloud salvati

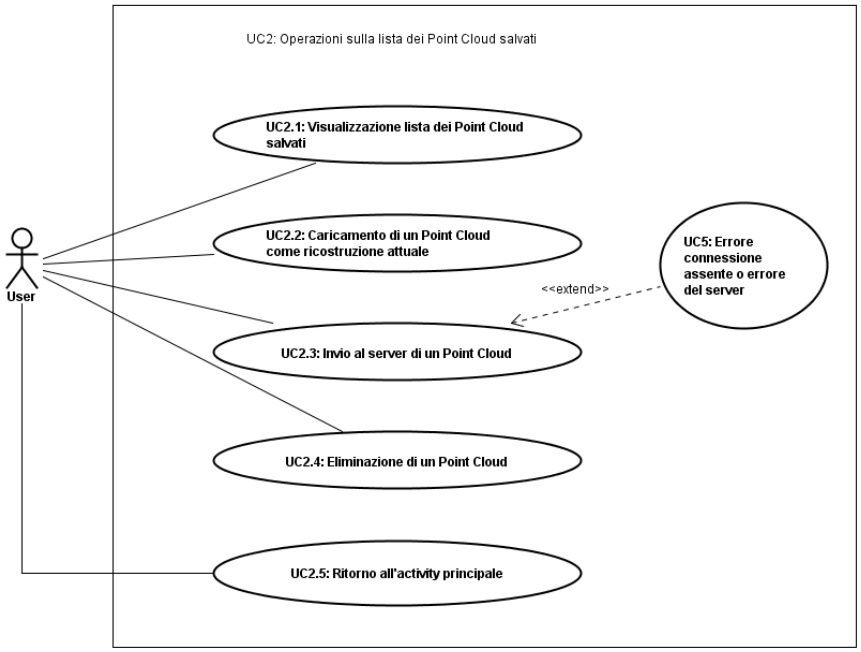

figura 4.3: Use Case - UC2: Operazioni sulla lista dei Point Cloud salvati

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione ed ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco.

Descrizione: L'utente vede sullo schermo la lista delle ricostruzioni 3D salvate su disco su cui può effettuare diverse azioni.

Postcondizioni: Il sistema è pronto per ricevere una nuova interazione.

4.1.12 UC2.1: Visualizzazione lista dei PointCloud salvati

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco.

Descrizione: L'utente consulta la lista dei *Point Cloud* salvati su disco.

Postcondizioni: Nessuna.

4.1.13 UC2.2: Caricamento di un Point Cloud come ricostruzione attuale

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco ed intende caricare una di queste come ricostruzione attuale.

Descrizione: L'utente, premendo sul nome del *file* scelto, lo carica come ricostruzione corrente.

Postcondizioni: Il sistema ritorna all'*activity* principale (quella degli UC1.*) con la ricostruzione caricata da file come ricostruzione corrente.

4.1.14 UC2.3: Invio al Server di un Point Cloud

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco ed ha intenzione di inviare al server uno dei *Point Cloud* salvati.

Descrizione: L'utente applica una lunga pressione sul nome del *file* scelto, apparirà un menù; da quest'ultimo l'utente seleziona "Send To Server" e la ricostruzione sarà mandata al *Server* in formato `pcd`. Generalmente questa funzione viene sfruttata se quando si effettua una rilevazione non si ha immediatamente la possibilità di inviare i dati tramite *Internet*.

Postcondizioni: Il *File* selezionato viene correttamente spedito al *Server*.

4.1.15 UC2.4: Eliminazione di un Point Cloud

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco ed ha intenzione di cancellare uno dei *File* salvati.

Descrizione: L'utente applica una lunga pressione sul nome del *file* scelto, apparirà un

menù; da quest'ultimo l'utente seleziona "Delete" ed il *File* selezionato viene cancellato.

Postcondizioni: Il *File* selezionato è stato cancellato e non è più presente su disco.

4.1.16 UC2.5: Ritorno all'activity principale

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle ricostruzioni 3D salvate su disco ma desidera ritornare all'*activity* principale.

Descrizione: L'utente preme sul tasto "Back" e ritorna all'*activity* principale.

Postcondizioni: L'utente ritorna all'*activity* principale.

4.1.17 UC3: Operazioni sulla lista delle mesh salvate

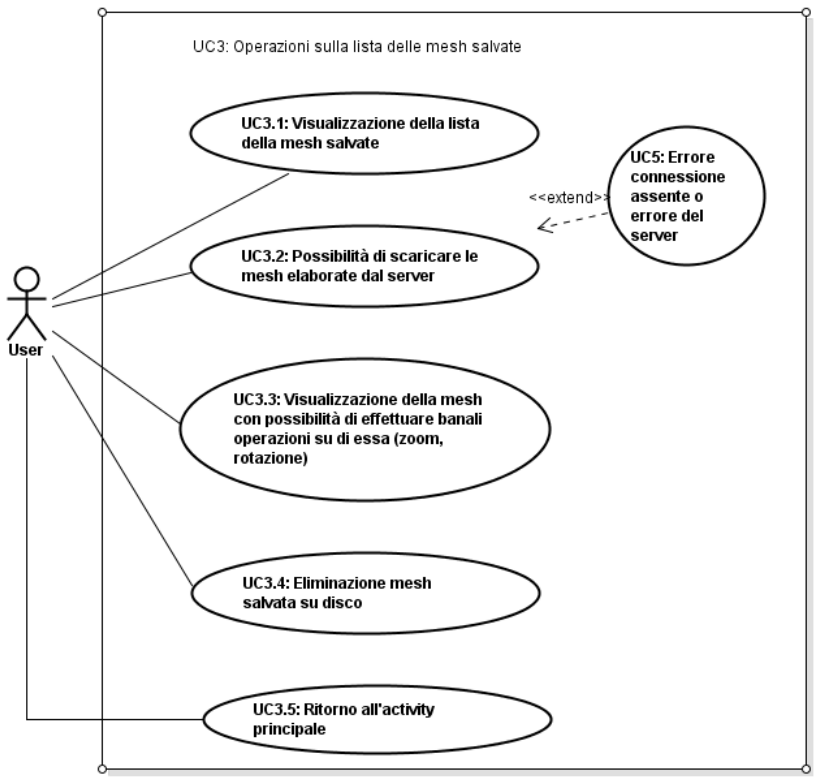

figura 4.4: Use Case - UC3: Operazioni sulla lista delle mesh salvate

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione ed ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco.

Descrizione: L'utente vede sullo schermo la lista *mesh* salvate su disco su cui può effettuare diverse azioni.

Postcondizioni: Il sistema è pronto per ricevere una nuova interazione.

4.1.18 UC3.1: Visualizzazione della lista delle mesh salvate

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione ed ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco.

Descrizione: L'utente consulta la lista delle *mesh* salvate su disco.

Postcondizioni: Nessuna.

4.1.19 UC3.2: Possibilità di scaricare le mesh elaborate dal Server

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco ed ha intenzione di aggiornare la lista di *mesh* aggiungendo le altre presenti sul *Server*.

Descrizione: L'utente preme sul simbolo di *refresh* in alto a destra e ricarica la lista di *mesh* eventualmente scaricando quelle sul *Server* ma non sul dispositivo.

Postcondizioni: L'utente ha a disposizione una lista aggiornata di *mesh*.

4.1.20 UC3.3: Visualizzazione grafica delle mesh

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco ed intende visualizzare una specifica *mesh* in 3D.

Descrizione: L'utente preme sul nome della *mesh* che intende visualizzare, a questo punto si apre un piccolo ambiente grafico 3D dove l'utente può osservare la ricostruzione ed effettuare banali operazioni su di essa.

Postcondizioni: Nessuna.

4.1.21 UC3.4: Eliminazione mesh salvata su disco

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco.

Descrizione: L'utente effettuerà le operazioni necessarie per cancellare la *mesh*.

Postcondizioni: Il *File* selezionato è stato cancellato e non è più presente su disco.

4.1.22 UC3.5: Ritorno all'activity principale

Attori Principali: Utente.

Precondizioni: L'utente ha aperto l'applicazione, ha premuto sul pulsante per visualizzare la lista delle *mesh* salvate su disco ma desidera ritornare all'*activity* principale.

Descrizione: L'utente preme sul tasto "Back" e ritorna all'*activity* principale.

Postcondizioni: L'utente ritorna all'*activity* principale.

4.1.23 UC4: Localizzazione fallita

Attori Principali: Utente.

Precondizioni: Le operazioni di localizzazione non sono andate a buon fine.

Descrizione: L'utente non sarà in grado di procedere alla rilevazione, sarà mostrato un messaggio d'errore.

Postcondizioni: Non può essere effettuata alcuna rilevazione.

4.1.24 UC5: Errore connessione assente o errore del Server

Attori Principali: Utente.

Precondizioni: L'utente cerca di compiere una operazione che richieda comunicazione con il *Server* senza disporre di una connessione ad internet.

Descrizione: L'utente sarà avvisato del fallimento dell'operazione ma potrà ritentare in seguito.

Postcondizioni: La comunicazione tra dispositivo e *Server* non va a buon fine.

4.2 Requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R [F|Q|V|P] [D|O] dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

P = prestazionale

O = obbligatorio

D = desiderabile

Nelle tabelle seguenti sono riassunti i requisiti e il loro tracciamento con gli *use case* delineati in fase di analisi.

4.2.1 Requisiti Funzionali

Requisito	**Descrizione**	**Fonti**
RFO-1	L'utente può effettuare una nuova rilevazione	UC1
RFO-1.1	L'utente può riprendere, mediante la pressione di un tasto, il *Point Cloud* attualmente inquadrato.	UC1.1
RFO-1.2	Il *Point Cloud* catturato deve essere correttamente aggiunto alla ricostruzione corrente.	UC 1.1
RFO-1.3	L'utente può modificare la visualizzazione del *Point Cloud* a piacimento e scegliere tra la visualizzazione in tempo reale del sensore di profondità e quella della ricostruzione salvata.	UC1.2, UC1.3
RFO-1.4	Il sistema è in grado di scartare la ricostruzione corrente ed iniziarne una nuova	UC1.4
RFD-1.5	Dopo aver scartato una ricostruzione il sistema è in grado di iniziare la successiva senza dover ripetere le operazioni di localizzazione.	UC1.4
RFO-1.6	L'utente può inviare i dati al *Server*.	UC1.5, UC2.3
RFD-1.7	Nel caso in cui non sia disponibile la connessione internet mentre l'utente sta cercando di inviare la ricostruzione corrente al *Server* deve essere mostrato un opportuno messaggio d'errore.	UC1.5, UC5
RFO-1.8	Il sistema può salvare i dati della ricostruzione corrente su disco nella cartella interna dell'applicazione, il formato deve essere `pcd`.	UC1.6
RFO-1.9	Il sistema deve calcolare in tempo reale le principali statistiche riguardanti: posizione del dispositivo, ricostruzione corrente e nuvola di punti inquadrata.	UC1.7
RFO-1.10	Il sistema deve permettere operazioni di undo	UC1.8
RFO-2	Il sistema deve permettere operazioni sui *file* `pcd` salvati su disco.	UC2

RFO-2.1	Il sistema deve essere in grado di fornire la lista di tutti i *Point Cloud* salvati.	UC2.1
RFO-2.2	Il sistema deve essere in grado di aprire un *file* `pcd` e caricarlo come ricostruzione corrente	UC2.2
RFO-2.3	Il sistema deve essere in grado di eliminare un *Point Cloud* salvato.	UC2.
RFO-3	Il sistema deve essere in grado di permettere operazioni sui file di *mesh* salvati su disco.	UC3
RFO-3.1	Il sistema deve essere in grado di fornire la lista di tutte le *mesh* salvati.	UC3.1
RFO-3.2	Il sistema deve essere in grado di dare la possibilità di scaricare le *mesh* elaborate dal *Server*.	UC3.2
RFD-3.3	Nel caso in cui non sia disponibile la connessione internet mentre l'utente sta cercando di scaricare la lista di *mesh* dal *Server* deve essere mostrato un opportuno messaggio d'errore.	UC3.2, UC5
RFO-4	L'applicazione deve fornire una interfaccia che permetta all'utente di svolgere semplicemente tutte le operazioni riportate nei casi d'uso.	UC0, decisione interna
RFO-4.1	L'interfaccia deve fornire un insieme di pulsanti per permette all'utente di impartire ordini al sistema.	UC1.2 UC1.3 UC1.5 UC1.6 UC1.8 UC1.11 UC1.12 UC2.2 UC2.3 UC2.4 UC2.5 UC3.2 UC3.3 UC3.4 UC3.5
RFO-4.1.1	L'interfaccia deve fornire un pulsante per permettere la registrazione di un singolo *Point Cloud*.	UC1.1
RFO-4.1.2	L'interfaccia deve fornire un pulsante per permettere di passare con il *render* dalla visione in prima alla visione in terza persona e viceversa.	UC1.2
RFO-4.1.3	L'interfaccia deve fornire un interruttore per permettere di alternare tra la visualizzazione in tempo reale e quella dell'oggetto ricostruito.	UC1.3
RFO-4.1.4	L'interfaccia deve fornire un pulsante per permettere il reset della ricostruzione.	UC1.4

RFO-4.1.5	L'interfaccia deve fornire un pulsante per permettere l'invio dei dati al *Server*.	UC1.5
RFO-4.1.6	L'interfaccia deve fornire un pulsante per permettere il salvataggio dei dati su disco.	UC1.6
RFO-4.1.7	L'interfaccia deve fornire un pulsante per permettere le operazioni di undo.	UC1.8
RFO-4.1.8	L'interfaccia deve fornire un pulsante per permettere di passare alla visualizzazione dei file contenenti i *Point Cloud*.	UC1.11
RFO-4.1.9	L'interfaccia deve fornire un pulsante per permettere di passare alla visualizzazione dei file contenenti le *mesh*.	UC1.12
RFO-4.1.10	L'interfaccia deve fornire un pulsante per permettere il caricamento di un *Point Cloud* come ricostruzione attuale.	UC2.2
RFO-4.1.11	L'interfaccia deve fornire un pulsante per permettere l'invio al *Server* di un *Point Cloud* dall'*activity* che lista i *file* `pcd`.	UC2.3
RFO-4.1.12	L'interfaccia deve fornire un pulsante per permettere l'eliminazione di un *Point Cloud* salvato su disco.	UC2.4
RFO-4.1.13	L'interfaccia deve fornire un pulsante per permettere il ritorno dalla lista dei *Point Cloud* all'*activity* principale.	UC2.5
RFO-4.1.14	L'interfaccia deve fornire un pulsante per permettere di scaricare delle *mesh* elaborate dal *Server*.	UC3.2
RFO-4.1.15	L'interfaccia deve fornire un pulsante per permettere di eliminare una *mesh* salvata su disco.	UC3.4
RFO-4.1.16	L'interfaccia deve fornire un pulsante per permettere il ritorno dalla lista delle *mesh* all'*activity* principale.	UC3.5
RFO-4.2	L'interfaccia deve fornire delle statistiche riguardanti il *Point Cloud* e la ricostruzione corrente in tempo reale.	UC1.7
RFO-4.3	L'interfaccia deve fornire opportuni strumenti per visualizzare dati dei sensori e le varie ricostruzioni in maniera grafica.	UC1.9 UC1.10 UC3.3
RFO-4.3.1	L'interfaccia deve fornire la possibilità di visualizzare sullo schermo del dispositivo la *preview* della fotocamera a colori.	UC1.10
RFO-4.3.2	L'interfaccia deve fornire la possibilità di visualizzare sullo schermo del dispositivo un *render* di tipo *OpenGL* in grado di mostrare *Point Cloud*.	UC1.9

RFO-4.3.2.1	Il render deve permettere l'operazione di rotazione quando possibile tramite *swipe* del dito.	UC1.9
RFO-4.3.2.2	Il render deve permettere l'operazione di zoom tramite *pinch* delle dita.	UC1.9
RFO-4.3.3	L'interfaccia deve fornire la possibilità di visualizzare sullo schermo del dispositivo un *render* per le *mesh* 3D.	UC3.3
RFO-4.3.3.1	Il render deve permettere l'operazione di rotazione quando possibile tramite *swipe* del dito.	UC3.3
RFO-4.3.3.2	Il render deve permettere l'operazione di zoom tramite *pinch* delle dita.	UC3.3

tabella 4.1: Tabella del tracciamento dei requisti funzionali

4.2.2 Requisiti Qualitativi

Requisito	**Descrizione**	**Fonti**
RQO-1	Separazione tra *business logic* e interfaccia grafica.	Obiettivi qualitativi interni.
RQO-1.1	Separazione tra gestione del ciclo di vita delle *Tango API* rispetto al ciclo di vita delle *activity*.	Obiettivi qualitativi interni.
RQO-1.2	Separazione tra gestione del ciclo di vita del *render* dei punti rispetto al ciclo di vita delle *activity*.	Obiettivi qualitativi interni.
RQD-2	Il prodotto deve superare tutti i test di sistema.	Obiettivi qualitativi interni.

tabella 4.2: Tabella del tracciamento dei requisti qualitativi

4.2.3 Requisiti di Vincolo

Requisito	**Descrizione**	**Fonti**
RVO-1	L'applicazione deve essere pienamente compatibile con il sistema *Android* ed i dispositivi *Tango*.	Decisioni interne.
RVO-2	L'applicazione deve essere pienamente disponibile in lingua Inglese.	Decisioni interne.

tabella 4.3: Tabella del tracciamento dei requisti di vincolo

4.2.4 Requisiti Prestazionali

Requisito	Descrizione	Fonti
RPO-1	Le operazioni di ricostruzione e visualizzazione del *Point Cloud* devono essere svolte efficientemente	Richiesta committente
RPO-1.1	L'elaborazione del *Point Cloud* deve essere svolta in un tempo finito e senza grosse variazioni tra una rilevazione e l'altra.	UC1.1
RPD-1.2	L'elaborazione del *Point Cloud* deve essere sufficientemente ottimizzata da poter permettere almeno 5-6 catture al secondo.	UC1.1
RPD-1.3	Il cambio di modalità del render deve essere effettuato senza *delay* in quanto è una operazione molto frequente durante la rilevazione.	UC1.2 UC 1.3
RPD-2	I *file* e le strutture dati utilizzate per salvare e spedire le ricostruzioni ed i singoli *Point Cloud* devono essere ottimizzati.	Richiesta committente analisi dei rischi
RPD-2.1	I *file* `pcd` generati devono essere di dimensioni ridotte e non devono presentare punti uguali ripetuti.	UC1.5 UC1.6 UC2
RPD-2.2	I pacchetti da inviare al *Server* devono essere di dimensioni adeguate ed il *file* da inviare deve essere diviso e non essere inviato tutto in una volta.	UC 1.6 UC2.3 UC3.2
RPD-3	Il *render* dei *Point Cloud* deve presentarsi fluido, non scattoso e rappresentare sempre quello che il dispositivo sta inquadrando momento per momento.	UC1.9
RPD-4	Il *render* delle *mesh* deve presentarsi fluido e non scattoso.	UC3.3
RPD-5	L'interfaccia deve essere sempre responsiva e non bloccarsi mentre c'è una elaborazione in corso.	Richiesta committente

tabella 4.4: Tabella del tracciamento dei requisti prestazionali

Capitolo 5

Progettazione

In questo capitolo verranno descritte le principali scelte progettuali prese per lo sviluppo dell'ultimo prototipo prodotto. Per una descrizione dettagliata dell'architettura del sistema si rimanda alla specifica tecnica in appendice A.

5.1 Strutture dati

Per comprendere il sistema è necessario innanzitutto studiare le strutture dati principali che lo compongono.

5.1.1 Punti e Point Cloud

SambaPoint

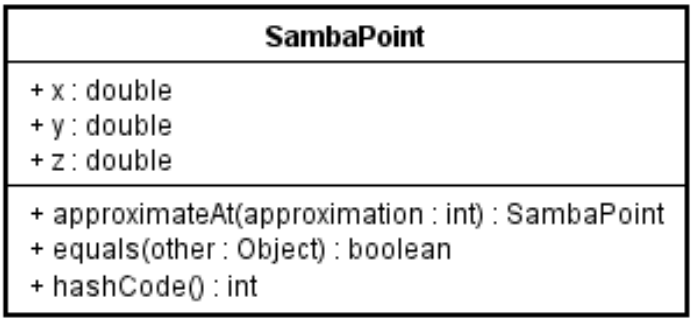

figura 5.1: Diagramma UML della classe SambaPoint

La struttura di base, onnipresente nel sistema, è il `SambaPoint`: è un semplice punto nello spazio definito dalle sue tre coordinate x, y e z.
La classe `SambaPoint` è una estensione di `Vector3`, della nota libreria grafica *Android Rajawali3D*, al fine di aumentarne la compatibilità con le utilità grafiche offerte da quest'ultima. Come in `Vector3` i campi dati delle coordinate sono pubblici, per aumentare l'efficienza.

`SambaPoint` aumenta le funzionalità della sua superclasse offrendo la possibilità di confronto coordinata a coordinata ed una funzione di approssimazione.

Point Cloud di una ricostruzione 3D

Al fine di memorizzare un oggetto complesso mediante una nuvola di punti si è resa necessaria una nuova struttura dati, più complessa di un semplice insieme di `SambaPoint`.
La necessità principale è quella di poter "sommare" due nuvole di punti a patto di conoscere la rotazione e la traslazione relative tra le due. Matematicamente è una semplice trasformazione, ma la sua implementazione su *Point Cloud* molto corposi non è banale; inoltre si è provato sperimentalmente che proprio questa trasformazione è un grosso collo di bottiglia per l'efficienza del sistema: trasformare in maniera poco efficiente una nuvola di migliaia di punti può richiedere anche qualche secondo di elaborazione per ogni rilevazione, rendendo il sistema inutilizzabile.
Alla luce di ciò si è scelto di implementare questa struttura dati come *ConcreteStrategy* di uno *Strategy Pattern*, lasciando così la porta aperta a future implementazioni che possano migliorare l'efficienza degli algoritmi.

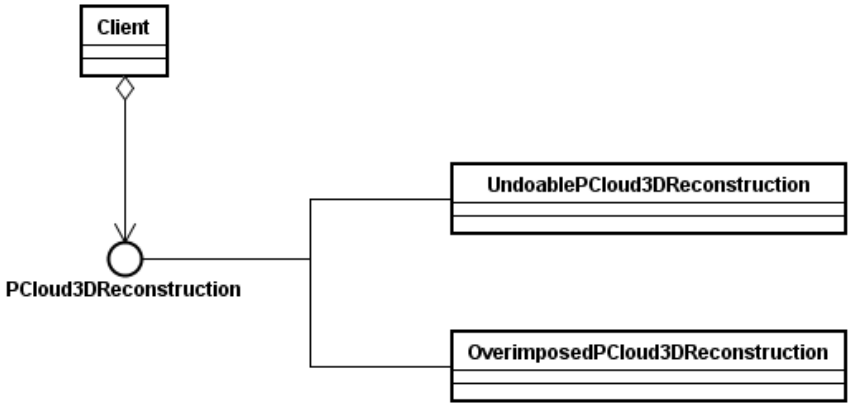

figura 5.2: Diagramma UML della classe dell'implementazione dello *Strategy Pattern* per le ricostruzioni 3D

La figura 5.2 mostra il diagramma (parziale) dell'architettura usata. Ad esempio si può notare che, oltre alla strategia inizialmente pianificata (ovvero `OverimposedPCloudReconstruction`), è già stata aggiunta una seconda che permette operazioni di *undo*.
Una qualsiasi implementazione di `PCloud3DReconstruction` rappresenta una nuvola di punti composta da una o più rilevazioni sommate assieme.

5.1.2 Stato dell'applicazione

L'utilizzo medio del prodotto prevede la registrazione di un gran numero di riprese, che devono essere conservate in memoria ed elaborate per fornire un *output* di qualità.

ReconstructionCache

`ReconstructionCache` è una struttura dati piuttosto complessa che tiene memoria di tutte le riprese effettuate e compie ottimizzazioni sui punti immagazzinati.
Essa inoltre è uno stato condiviso tra molti *thread* che lavorano concorrentemente; quindi si occupa anche della sincronizzazione e di proteggere opportunamente l'accesso agli stessi in maniera da evitare le potenziali *race condition*.
La classe è un *Singleton* in modo da garantire che tutte le componenti che fanno riferimento a quest'ultima facciano riferimento alla stessa istanza.
Verso l'esterno essa rappresenta la grande mole di dati che contiene come un unico insieme di punti (che ad esempio può essere rappresentato graficamente come in figura 1.1). Mentre, al suo interno divide i dati in tre differenti strutture.

- Una pila per i *Point Cloud* grezzi ricevuti dai sensori *Tango*, scritti in coordinate relative e che pertanto non possono essere semplicemente sovrapposti gli uni agli altri.
- Una struttura dati di ricostruzione 3D (come ad esempio `UndoablePCloudReconstruction`), che contiene i dati di molte nuvole di punti, correttamente ruotati e sovrapposti tra loro.
- Un insieme di punti già ottimizzati: ovvero correttamente approssimati, non ridondanti etc.

Un *Point Cloud* appena catturato viene immediatamente inserito nella pila dei dati grezzi, poi i vari processi di cui è composto il sistema si occuperanno di "spostarlo" da una struttura all'altra fino a farlo giungere nell'insieme dei punti ottimizzati. Nella sezione 5.1.3 verranno esplicitati i meccanismi con cui ciò avviene.

5.1.3 Servizi

Come detto nella sezione precedente le operazioni sulle nuvole di punti sono piuttosto dispendiose e quindi impiegano qualche tempo a concludere: non possono, quindi, essere eseguite sullo stesso *tread* che gestisce il resto dell'applicazione. Per questo sono eseguite concorrentemente.

MergingService e VoxelingService

`MergingService` e `VoxelingService` sono i due servizi che eseguono operazioni sui dati raccolti. La figura 5.3 mostra, in un diagramma informale non UML, il ciclo di vita dei dati all'interno di `ReconstructionCache`.

- Inizialmente i sensori *Tango* inviano i, dati così come li raccolgono, alla `ReconstructionCache`; vengono posti nella pila.
- Quando la pila è non vuota viene lanciato `MergingService` su un suo *thread*. Quest'ultimo fa il *pop* della pila, ottenendo i dati precedentemente inseriti dal sensore, che sono composti dall'insieme dei punti e da una matrice che descrive la trasformazione che è necessario applicare.
- Una volta ottenuti i dati richiede l'accesso alla `Reconstruction` ed effettua la sovrapposizione del *Point Cloud* grezzo con quello (eventualmente vuoto) salvato all'interno della `Reconstruction` stessa.

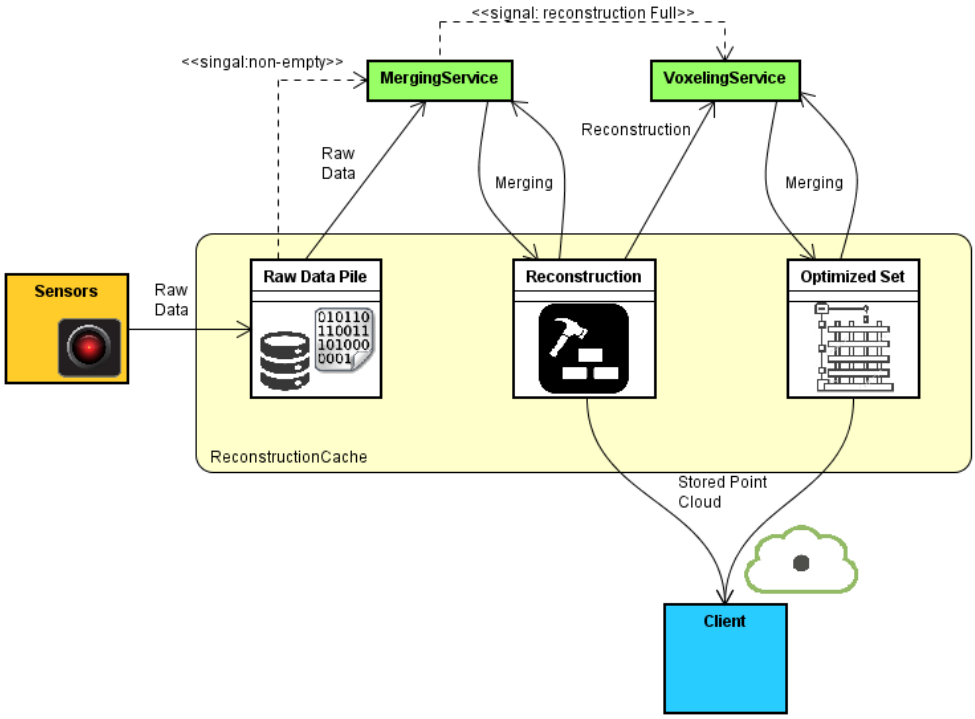

figura 5.3: Diagramma informale del funzionamento dei servizi di Samba

Si noti che la `Reconstruction` attualmente viene bloccata dal processo di *merging* durante le operazioni di trasformazione del *Point Cloud* grezzo, anche se non ce ne sarebbe bisogno. Ciò è voluto in quanto le operazioni di *merging* saranno soggette a miglioramenti e cambiamenti, quindi potrebbe in futuro rivelarsi necessario fare delle assunzioni sulla `Reconstruction` e non limitarsi alla trasformazione geometrica. A tal proposito si veda la sezione 7.2.1.

- La quantità di punti all'interno di `Reconstruction` cresce molto rapidamente. Pertanto il servizio di *merging*, quando il numero di punti supera un tetto prefissato lo segnala al `VoxelingService`.

- `MergingService` ripete le precedenti operazioni fino a quando non avrà svuotato la pila, al che si addormenterà in attesa di un nuovo segnale.

- Il servizio di *voxeling*, svuota `Reconstruction`, ottimizza i punti (approssimazione e rimozione dei duplicati) e poi aggiunge il risultato ad `Optimized Set`, ovviamente eliminando altri eventuali duplicati. In questo modo all'interno di `Optimized Set` ci sarà un *Point Cloud* con punti disposti in una griglia, non sovrapposti e soprattutto in numero molto minore.

- Quando i dati vengono inseriti all'interno di `Reconstruction` vengono già ritenuti "buoni". Infatti quando un *Client* richiede tutti i punti del *Point Cloud* ricostruito si vede ritornare l'unione tra `Optimized Set` e `Reconstruction`.

5.1.4 Manager

Sensori *Tango* e *render* grafico hanno un loro ciclo di vita che deve essere gestito all'inteno di quello delle *activity* di *Android*.
Al fine di separare la logica dell'applicazione dalla sua interfaccia sono stati strutturati dei manager per gestire questi due aspetti: da parte delle *activity* è sufficiente chiamare i metodi di *start* e *stop* coerentemente con il proprio ciclo di vita.

5.2 Design Pattern utilizzati

I *design pattern* sono soluzioni a problemi ricorrenti. Adottare i *design pattern* semplifica l'attività di progettazione, favorisce il riutilizzo del codice e rende l'architettura più mantenibile.
Nella realizzazione di *Samba* sono stati usati i *design pattern* descritti in questa sezione.

5.2.1 MVP

Scopo dell'utilizzo

È stato scelto il *design pattern Model View Presenter* per separare la logica dell'applicazione dalla sua rappresentazione e per seguire le *Android Best Practices*.

Diagramma

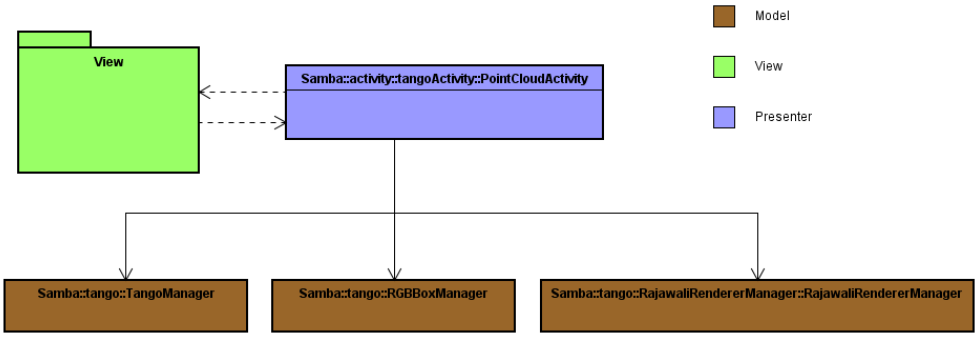

figura 5.4: Design Pattern MVP, applicazione in Samba

Il diagramma spiega la struttura generale di come il pattern MVP è stato utilizzato all'interno di Samba prendendo come esempio una specifica implementazione. Tale implementazione è relativa alla struttura generale del sistema.

Contesto dell'utilizzo

Il pattern *MVP* è usato per l'architettura generale del sistema.

5.2.2 Strategy

Scopo dell'utilizzo

Il *Design Pattern Strategy* è stato usato per separare la dichiarazione di alcuni algoritmi dalla loro implementazione. Ad esempio negli stadi iniziali non era stato fissato un formato ufficiale per i *file* di *output*; quindi si è lasciata aperta la possibilità di modificarlo in un secondo momento.

Diagramma

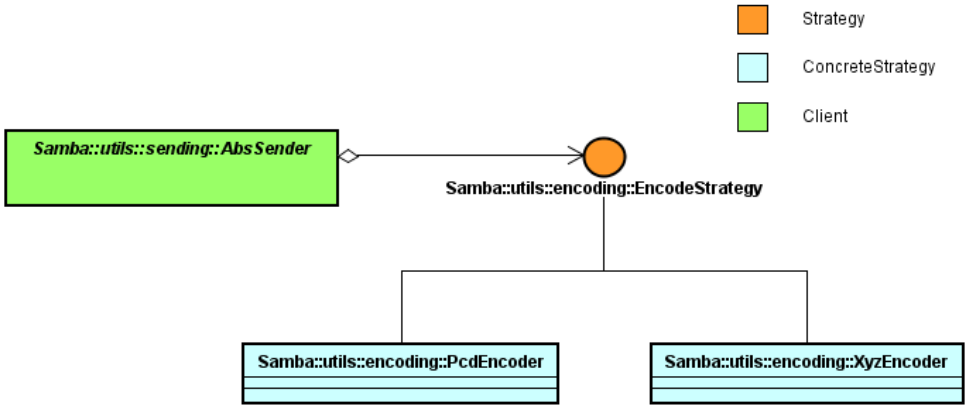

figura 5.5: Design Pattern Strategy, applicazione in Samba

Contesto dell'utilizzo

L'efficienza è un aspetto centrale del sistema, per questo molti algoritmi sono stati implementati mediante *Strategy* per permettere migliorie future.
Ad esempio è stato usato uno *Strategy Pattern* per gestire il formato dei *file* in *output*, la rotazione dei *Point Cloud*, le ottimizzazioni sui punti, i servizi per spedire e ricevere informazioni dal *Server* etc.

5.2.3 Observer

Scopo dell'utilizzo

Il *Design Pattern Observer* è stato usato per permettere l'aggiornamento della *view* da parte dei *manager*.

Diagramma

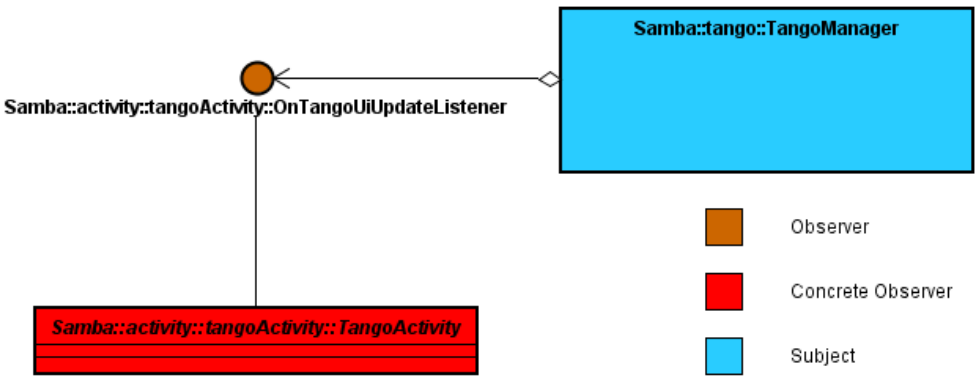

figura 5.6: Design Pattern Observer, applicazione in Samba

Contesto dell'utilizzo

È stato usato per permettere l'aggiornamento della view da parte dei *Manager* di:

- ciclo di vita *Tango*.
- render grafico.
- *preview* della fotocamera.

Nel diagramma in figura 5.2.3 si può osservare il *design pattern* relativo al *manager* del ciclo di vita dei sensori *Tango*.

5.2.4 Singleton

Scopo dell'utilizzo

Il *Design Pattern Singleton* è stato usato per controllare gli accessi alle classi che mantengono gli stati condivisi tra i vari processi.

Diagramma

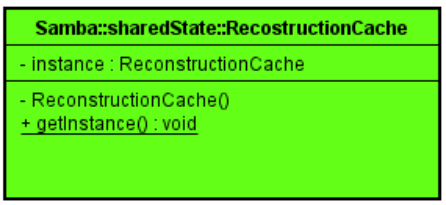

figura 5.7: Design Pattern Singleton, applicazione in Samba

Contesto dell'utilizzo

È stato usato nelle classi che mantengono gli stati condivisi tra i vari processi. Ovvero:

- `Samba.sharedState.ReconstructionCache`
- `Samba.sharedState.ReconstructionManager`

Capitolo 6

Test di Sistema

Data la natura prototipale del progetto ai test di Sistema è stata riservata una grande attenzione. Durante lo sviluppo dei prototipi giocattolo e comunque durante tutta la fase di progettazione e ed ideazione è stato continuamente incrementato un documento nella *wiki* interna all'azienda al fine di immagazzinare li tutti i test di sistema che devono essere soddisfatti prima di ritenere "buono" un prototipo. Segue la lista di tutti i Test di Sistema con relativo stato di soddisfacimento.

Test	**Descrizione**	**Stato**
TS1.0	L'app al primo avvio deve mostrare la richiesta di permessi per l'*area learning*.	Success.
TS1.1	L'app deve caricare correttamente il render dei punti, il quadratino della telecamera e tutta l'*UI*.	Success.
TS1.2	Il pulsante *"List 3D object"* deve portare alla corretta *activity*.	Success.
TS1.3	Il pulsante *"OBJ"* deve portare alla corretta *activity*.	Success.
TS2.0	All'avvio di *PointCloudActivity* deve comparire lo *splash screen* di Tango.	Success.
TS2.1	Segue la fase di localizzazione in cui l'utente non dovrebbe poter scattare foto e deve essere mostrato un avviso fintantoché la localizzazione non sarà avvenuta.	Success.
TS2.2	A localizzazione avvenuta deve venire mostrato un avviso contenente la precisione stimata della localizzazione.	Success.
TS3.0	La *preview* della fotocamera deve mostrare correttamente quello che inquadra.	Success.
TS3.1	La preview deve essere sempre disponibile nell'*activity* principale. (nelle versioni precedenti a causa di un bug a volta la fotocamera non era disponibile se ci si ritornava all'*activity* principale da un'altra *activity*.)	Fallito.

TS4.0	Nello schermo deve essere disponibile il rendering dei punti attualmente visualizzati dal dispositivo *Tango*. Deve aggiornarsi in tempo reale e non effettuare salti o particolari fluttuazioni.	Success.
TS4.1	Mediante *toggle* dell'interruttore *"Reconstruction mode"* deve essere possibile visualizzare gli *shot* catturati fino a quel momento e tornare indietro alla visione standard.	Success.
TS4.2	Con i pulsanti *"third person"* e *"first person"* deve essere possibile passare dalla visione in prima a quella in terza persona.	Success.
TS4.3	Con *pinch/swipe* deve essere possibile cambiare zoom/orientamento del *rendering* (orientamento solo in terza persona).	Success.
TS4.4	Premendo il tasto *"shot"* il sistema deve rilevare e salvare i punti ruotati secondo l'orientamento/posizione del dispositivo.	Success.
TS4.5	Dopo un fissato numero di shot si deve attivare il servizio di *voxeling*. La ricostruzione visualizzata apparirà infatti più rada.	Success.
TS4.6	Premendo il tasto *"undo"* deve essere sempre disponibile quella operazione. (Almeno una).	Success.
TS4.7	Le operazioni di *"shot"* e *"undo"* devono modificare correttamente numero di *shot* presi e numero di punti salvati.	Success.
TS4.8	Dopo un certo numero di shot scattati ad un oggetto esso deve apparire ben formato quando ricostruito.	Success.
TS4.9	Premendo il tasto *"reset"* deve essere possibile eliminare il *Point Cloud* attualmente registrato ed iniziarne uno nuovo.	Success.
TS5.0	I pulsanti devono essere tutti correttamente funzionanti.	Success.
TS5.1	La *dashboard* deve contenere: punti visualizzati, distanza media, posizione x, y e z, *frames-of-reference*.	Success.
TS5.2	I valori della *dashboard* devono essere aggiornati correttamente in tempo reale.	Success.
TS6.0	Una volta memorizzato una ricostruzione a *Point Cloud* (come nel test 4) deve essere possibile salvarlo con in tasto *"save"* ed inserendo un nome per il *file*.	Success.
TS6.1	Premendo *"list 3d object"* deve apparire la lista dei *Point Cloud* precedentemente salvati.	Success.
TS6.2	Ognuno di questi *file* deve poter essere caricato in memoria ed eliminato dal disco.	Success.
TS7.0	Premendo il tasto *"send record"* l'*app* deve inviare l'oggetto attualmente in costruzione al *Server*.	Success.

TS7.1	Premendo il tasto *"OBJ"* l'app deve visualizzare la lista delle *mesh* attualmente disponibili sul *Server*.	Success.
TS7.2	Da questo menù deve essere possibile selezionarne una per visualizzare ricostruzione e volume.	Success.
TS7.3	Deve essere anche possibile cancellare la *mesh*.	Success.

tabella 6.1: Test di sistema

Capitolo 7

Conclusioni

Al di là del formalismo informatico, lo scopo principale di questo progetto era indagare sul possibile uso della tecnologia *Tango* nel campo ispettivo come effettivo supporto allo studio di beni materiali. Il prototipo realizzato sembra confermare che cioè è possibile.
I risultati ottenuti sono stati piuttosto soddisfacenti e con qualche raffinamento appare possibile inserire l'applicazione in un contesto produttivo.

7.1 Prove pratiche

Il prototipo prodotto è stato testato in numerosi ambienti e su diversi oggetti. Nella quasi totalità dei casi i risultati sono stati più che sufficienti per quanto riguarda la qualità del *Point Cloud* ricostruito.
Per quanto riguarda invece il calcolo del volume i risultati non sono ancora totalmente sufficienti: il volume ottenuto è sempre dello stesso ordine di grandezza del volume reale, ma spesso è affetto da un errore relativo tra il 30 ed il 50% ed un errore del genere non è affatto tollerabile. Tale divario però è facilmente appianabile migliorando la qualità delle elaborazioni dei *Point Cloud* e delle *mesh* lato *Server*.

7.2 Sviluppi futuri

Il progetto è nato in tempi molto recenti, dopo circa due mesi di sviluppo è stato prodotto un prototipo soddisfacente. Molti dei problemi riscontrati durante il percorso di *stage* sono stati risolti, grazie ai prototipi e alle prove pratiche sono state molteplici anche le idee per rendere l'applicazione ancora più completa. Riporto qui solo alcune di queste.

7.2.1 ICP su tablet

Uno più gravi problemi delle ricostruzioni 3D effettuate tramite sovrapposizione di *Point Cloud* è il *ghosting*. Si tratta dello sdoppiamento di alcune "facce" dell'oggetto ricostruito.
Nell'esempio in figura 7.1 sono stati evidenziati in verde gli spigoli corretti di una scatola rettangolare, mentre con colore rosso quelli dovuti al *ghosting*; si può chiaramente notare che le facce laterali appaiono sdoppiate e ciò può portare a significativi errori

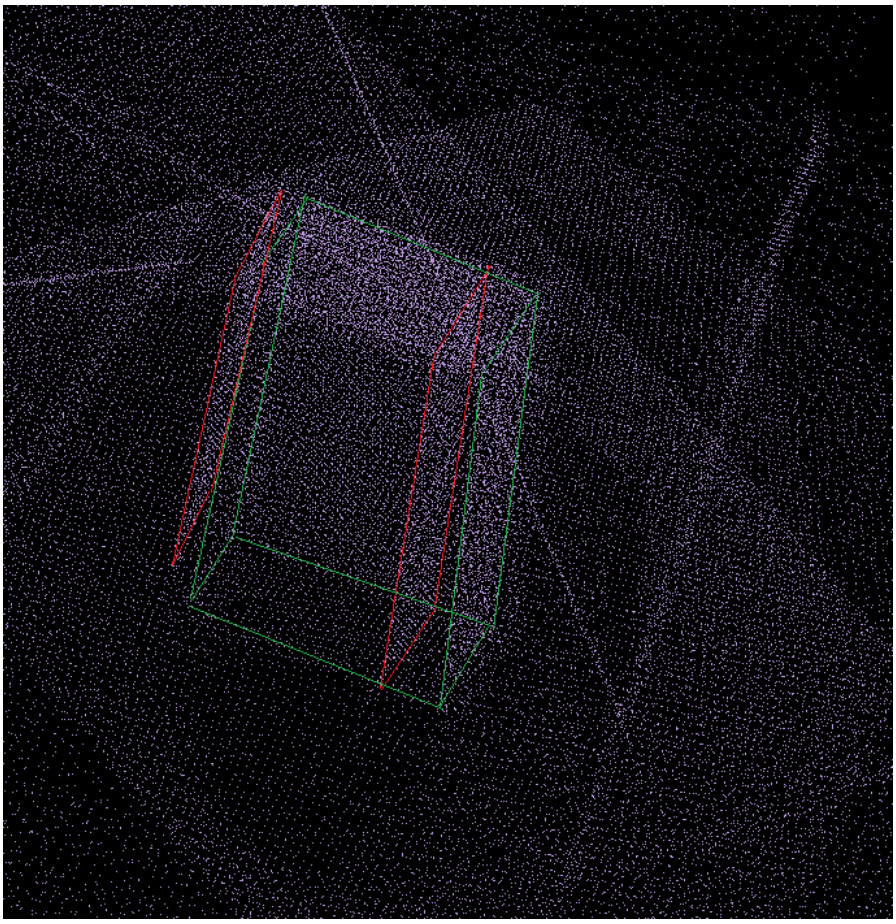

figura 7.1: Point Cloud che presenta problemi di *ghosting* di una scatola rettangolare

nella ricostruzione dell'oggetto e soprattutto nel calcolo del volume.
Questo fenomeno è dovuto ad errori di stima nella posizione del dispositivo, e per quanto si cerchi di ridurli essi rimarranno sempre. Si tratta di un altro limite fisico dei dispositivi *Tango*, che in questo caso è aggirabile.
ICP o *Iterative Closest Point* è un algoritmo che cerca di minimizzare le differenze tra due nuvole di punti. Applicando *ICP* su due *Point Cloud* che non si sovrappongono perfettamente permetterebbe di ottenere una matrice di trasformazione da applicare ad uno dei due per farlo combaciare all'altro. L'algoritmo in questione ha molte implementazioni in *C++*, tra cui una presente proprio all'interno della libreria *PCD* utilizzata lato *Server*. Questo fatto ha dato modo di testare la sua effettiva efficacia.
Il problema è che spedire ogni singola ripresa al *Server* ed aspettare una risposta sembra una strada non percorribile: per una rilevazione intera servono più di 20 riprese, senza connessione internet il servizio non sarebbe disponibile etc.
Per questo un possibile sviluppo futuro potrebbe essere quello di implementare *ICP* lato *tablet*. Ci sarebbero due vie percorribili: importare una delle tante implementazioni in *C++* ed accedervi dal codice *Java* mediate *JNI* oppure implementare da capo l'algoritmo nativamente in *Java*. Entrambe le ipotesi vanno attentamente valutate tenendo conto anche della potenza di calcolo e del consumo di batteria del dispositivo.

7.2.2 Integrazione C++/Jni lato tablet

Google fornisce oltre a delle ricche librerie *Java* anche delle *API* in linguaggio *C/C++*. Alcune funzioni esposte da queste ultime non sono presenti in quelle *Java* oppure sono molto più efficienti. Sarebbe quindi necessario, negli sviluppi futuri, predisporre una interfaccia *Jni* in maniera da integrare codice *Java* e *C++* all'interno della stessa applicazione.
Tutto il progetto ne gioverebbe, specialmente per quanto riguarda le *performance*; inoltre si potrebbe pensare di importare parti della libreria *PCD* in maniera da automatizzare alcuni processi.

7.2.3 Texture dei punti

Il prodotto fornisce delle buone ricostruzioni 3D per quanto riguarda la forma e le dimensioni dell'oggetto; ai fini ispettivi, di fatto, non c'è bisogno d'altro. Ciononostante le ricostruzioni visualizzate sia su *tablet* che su *computer* essendo formate da soli punti sono spesso di difficile comprensione da parte dell'utenza. Per rispondere a queste esigenza potrebbe essere opportuno pensare ad aggiungere ad ogni singolo punto una opportuna texture in maniera da rendere più immediato il riconoscimento dell'oggetto da parte dell'utente.
Questo sviluppo darebbe un grosso valore aggiunto in quando migliora grandemente l'aspetto grafico del sistema e lo rende quindi anche più vendibile.
Alcuni esempi di *Point Cloud texturizzati* sono già presenti in rete sotto licenza *Open Source*, quindi è possibile pensare al riuso degli stessi.

7.2.4 Rimozione artefatti

Un altro problema che affligge le ricostruzioni 3D effettuate da *Samba* è il rumore causato da forti fonti di luce o superfici riflettenti.
In molte riprese infatti appaiono dei piani sospesi a mezz'aria che si sommano li uni agli altri rendendo qualche volta la ricostruzione praticamente inutilizzabile. Lato *Server* essi sono spesso eliminabili dalla libreria *PCD*, ma lato *tablet* rendono la visualizzazione dei *Point Cloud* ricostruito ancora più caotica e difficilmente usabile.
Un possibile sviluppo è quindi quello di usare le caratteristiche stesse di questi artefatti (come essere isolati, sempre perfettamente planari etc) per filtrarli già durante la ripresa del singolo *Point Cloud* lato *tablet*. In figura 7.2 sono stati evidenziati in rosso alcuni degli artefatti.

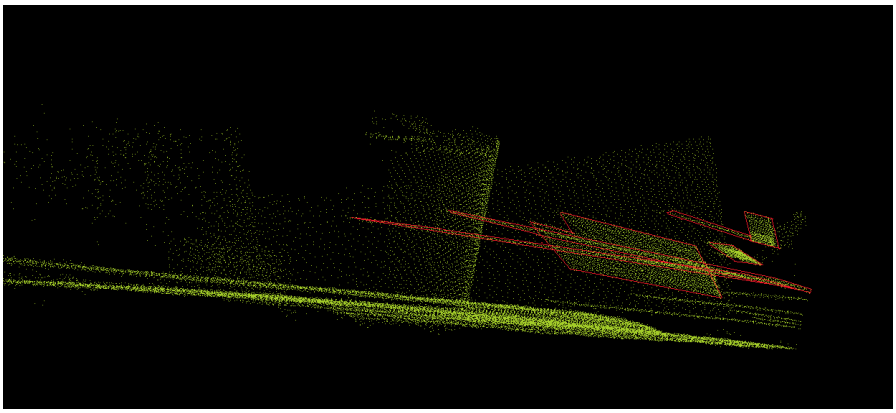

figura 7.2: Point Cloud che presenta problemi di artefatti di un bidone conico, vista laterale

7.2.5 Controllo di forma

Oltre alla creazione del modello 3D e del calcolo del volume potrebbe rivelarsi molto utile per gli ispettori avere uno strumento automatico per confrontare la forma dell'oggetto ispezionato con un modello "perfetto" del bene stesso. Ad esempio potrebbe essere usato per confrontare componenti meccaniche con i loro modelli *CAD* al fine di individuare eventuali deformazioni subite durante il trasporto.

7.2.6 Integrazione con l'applicazione Vic

L'azienda fornisce ai suoi dipendenti ispettori una applicazione che permette di automatizzare diverse mansioni. Tra le varie funzioni che mette a disposizione c'è quella di avviare una nuova ispezione o *job* permettendo all'utente di compilare diversi *form* per fornire così in tempo reale un rapporto dettagliato e standardizzato del lavoro svolto. In questo ambito una buona integrazione delle tecnologie *Tango* potrebbe fornire molto valore aggiunto.

7.2.7 Ricostruzione continua

Le ricostruzioni di *Samba* vengono effettuate "foto per foto", ovvero l'utente deve osservare l'oggetto da diverse angolazioni ed effettuare delle rilevazioni proprio come se si scattassero molte foto.
Un ottimo sviluppo sarebbe fare in modo che queste rilevazioni fossero effettuate in maniera automatica e continua, ciò darebbe molti vantaggi al sistema:

- Ci sarebbero molti più dati, quindi la rilevazione sarebbe di maggiore qualità.
- L'applicazione sarebbe più vicina ai bisogni dell'utente.
- Si genererebbero molti *Point Cloud* quasi uguali sovrapposti offrendo la possibilità di eliminare parte del rumore con metodi statistici (è improbabile che due scatti successivi dello stesso oggetto senza muovere il tablet siano affetti dagli stessi artefatti).

Apre però anche a nuovi problemi:

- Maggiore carico di informazioni che deve essere gestito ed ottimizzato.

- Maggior consumo della batteria.
- Necessità di euristiche per determinare quale sia un *Point Cloud* buono: l'utente non può più visionarli e scartare quelli affetti da errore.

7.3 Problemi ancora irrisolti

7.3.1 Surriscaldamento e consumo della batteria

Il prototipo prodotto ha un consumo della batteria piuttosto elevato ed a volte porta al surriscaldamento del dispositivo. Ciò secondo una prima analisi è dovuto ai seguenti fattori:

- Utilizzo combinato dei quattro sensori principali del *device*: fotocamera a colori, fotocamera *Fish-eye*, sensore *IR* ed accelerometro/giroscopio. Questi sensori sono tutti necessari per il *Motion Tracking* e la cattura dei *Point Cloud*.
- Grande mole di dati da elaborare: un singolo *Point Cloud* può contare anche 9000 punti, essi devono essere elaborati e renderizzati in tempo reale.
- Necessità di pesanti elaborazioni parallele: per permettere all'utente una interfaccia fluida e non rallentare i calcoli è necessario usare molti processi paralleli. Ogni scatto registrato è elaborato su di un proprio *Thread* ed è presente un servizio che ottimizza ad intervalli regolari la ricostruzione corrente (indispensabile per tenere sotto controllo la complessità delle strutture dati).
- *Rendering real-time*: come discusso in sezione 1.3.4 è un fondamentale requisito di usabilità avere una preview dei dati catturati dal *depth sensor*. Ciò implica l'utilizzo di un *render 3D OpenGL* che richiede una grande quantità di risorse.
- Connessione dati: la necessità di un *backend Server* comporta l'utilizzo della connessione internet. Inoltre dato il tipo di utilizzo per cui è stata ideata l'applicazione userà spesso la connessione *mobile* per inviare i dati.

Risolvere queste criticità era oltre gli obiettivi dello *stage*, ma è comunque stato stilato un documento contenente alcune contromisure che potranno essere usate per mitigarne gli effetti:

- Effettuare le operazioni più dispendiose usando le librerie native in *C* ed integrarne i risultati mediante una interfaccia *Jni*.
- Una volta migrate le operazioni di elaborazione dei punti da *Java* a *C* si dovrebbero avere un sensibile incremento nelle prestazioni che permetterebbe di limitare l'elaborazione a solo due processi: uno per gestire la trasformazione dei punti (rotazione, traslazione, sovrapposizione) ed uno per le operazioni di ottimizzazione (*voxeling*, rimozione rumore, rimozione ridondanze).
- Gestire la priorità del servizio di ottimizzazione: è trasparente all'utente e non deve necessariamente essere performante. Quindi esso può avere una minore priorità nell'utilizzo della *CPU* cercando di ridurre i *burst* di dati.
- Il *render* non ha lo scopo di rappresentare tutti i punti salvati, ma di fornire all'utente una idea di quello che "vede" il sensore di profondità. Si può pensare quindi di approssimare parti della nuvola di punti a figure geometriche semplici, ad esempio il pavimento può essere approssimato ad un piano. Ciò ridurrebbe il carico di lavoro per processore e scheda grafica.

7.3.2 Preview della fotocamera

Le *API* forniscono degli strumenti per automatizzare la *preview* della fotocamera a colori. Essi tuttavia si sono rivelati estremamente pesanti come carico computazionale e se combinati al resto dell'applicazione prodotta possono portare a rallentamenti imprevisti.
Per questo si è scelto di ottenere questa anteprima tramite metodi di basso livello. Essi tuttavia richiedono, da parte del programmatore, uno sforzo ben maggiore e quelli applicati in *Samba* vanno rivisti.
Un *bug* noto è che alla ripresa della attività, qualche volta la *preview* non viene attivata ed il riquadro a lei riservato rimane nero.

7.4 Consuntivo finale

Il periodo di *stage* ha avuto avvio il 13 Giugno 2016 ed è terminato il 16 Agosto 2016 invece che il 5 Agosto. Lo slittamento della data di fine è dovuto ad impegni universitari dello studente ed a un breve periodo di chiusura estiva dell'azienda. Il tirocinio ha avuto una durata totale di 320 ore.
In tabella 7.1 sono riportate le ore preventivate per ogni attività ed esse sono confrontate con le ore effettivamente impiegate.

Attività	**Preventivo**	**Consuntivo**	**Diff**	**Diff %**
Studio preliminare	20	24	+4	+20%
Ideazione modello di soluzione	20	16	-4	-20%
Studio di fattibilità/analisi dei rischi	20	32	+12	+60%
Analisi dei requisiti	20	12	-8	-40%
Prototipi preliminari	40	86	+46	+115%
Progettazione	40	30	-10	-25%
Codifica	120	80	-40	-33,3%
Verifica e validazione	30	25	-5	-16,6%
Prove pratiche	10	15	+5	+50%
Totale	320	320	+0	+0%

tabella 7.1: Distribuzione ore preventivo e consuntivo

Le ore totali sono state rispettate e non hanno avuto variazioni.
La loro ripartizione interna, invece, ha subito grosse variazioni a causa di sovrastime e sottostime commesse durante la fase di pianificazione.
Le attività che hanno subito variazioni più importanti sono state codifica e sviluppo dei prototipi preliminari. Questi ultimi hanno richiesto più del doppio del tempo preventivato a causa della difficoltà del compito richiesto, ma soprattutto perché lo studente ha intrapreso una strada sbagliata che ha portato ad un prototipo non funzionante e che è stato interamente scartato. Si può notare però che la maggiore attenzione riposta nei prototipi preliminari ha fatto risparmiare diverse ore alla codifica: infatti molto del codice presente nei primi prototipi è stato riusato senza cambiamenti, o con modifiche modeste.
Anche lo studio di fattibilità ha richiesto più tempo del previsto in quanto si è rivelato necessario cercare e provare molte applicazioni, alcune delle quali non manutenute o non aggiornate.

In figura 7.3 vengono riportati i diagrammi di *Gantt* dove è possibile confrontare la pianificazione preventivata e quella effettiva.

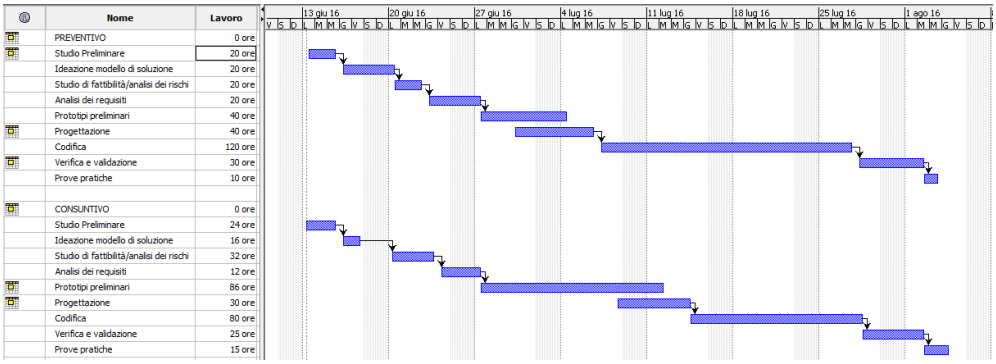

figura 7.3: Gantt della pianificazione preventiva e consuntiva

7.5 Raggiungimento degli obiettivi

7.5.1 Obiettivi generali

Gli obiettivi generali riportati nel piano di lavoro erano i seguenti:

- Obbligatori
  - *ob01*: studio delle soluzioni esistenti;
  - *ob02*: ideazione di una soluzione per il riconoscimento di un oggetto;
  - *ob03*: implementazione prototipo in grado di riconoscere un oggetto;
  - *ob04*: calcolo del volume dell'oggetto (approssimato);
  - *ob05*: implementazione app come indicato nella sezione *Struttura applicazione* del documento *Relazione app per Project Tango* fornito dall'azienda;
- Desiderabili
  - *de01*: generare modello 3D in un formato portabile (obj, ply, vtk);
  - *de02*: stima della precisione con cui viene ricostruito un oggetto;
  - *de03*: perfezionare calcolo del volume (cavità nell'oggetto, eliminazione dati di sfondo/rumore);
  - *de04*: stima della precisione nel calcolo del volume;
- Opzionali
  - *op01*: ottimizzazione della comunicazione con il server nel caso risultasse necessaria.

I requisiti obbligatori sono stati tutti raggiunti.
Per quanto riguarda quelli desiderabili invece:

- *de01*: È stato pienamente soddisfatto.

- *de02*: Sono stati forniti molti modelli grafici, da cui è possibile effettuare delle considerazioni sulla precisione con cui vengono ricostruiti gli oggetti.
- *de03*: Non soddisfatto.
- *de04*: Sono stati condotti degli studi, ma si è ritenuto che l'applicazione fosse ad uno stato ancora troppo poco avanzato perché la statistica avesse senso.

Il requisito opzionale *op01* non è stato raggiunto.

7.5.2 Requisiti

Dagli obiettivi posti nel piano di lavoro sono stati ricavati i requisiti esposti nel capitolo 4. Seguono le tabelle di soddisfacimento di questi ultimi divisi per categorie.

Requisiti funzionali

Requisito	**Soddisfacimento**
RFO-1	Soddisfatto
RFO-1.1	Soddisfatto
RFO-1.2	Soddisfatto
RFO-1.3	Soddisfatto
RFO-1.4	Soddisfatto
RFD-1.5	Soddisfatto
RFO-1.6	Soddisfatto
RFD-1.7	Non soddisfatto
RFO-1.8	Soddisfatto
RFO-1.9	Soddisfatto
RFO-1.10	Soddisfatto
RFO-2	Soddisfatto
RFO-2.1	Soddisfatto
RFO-2.2	Soddisfatto
RFO-2.3	Soddisfatto
RFO-2	Soddisfatto
RFO-3.1	Soddisfatto
RFO-3.2	Soddisfatto
RFD-3.3	Non soddisfatto
RFO-4	Soddisfatto
RFO-4.1	Soddisfatto
RFO-4.1.1	Soddisfatto
RFO-4.1.2	Soddisfatto
RFO-4.1.3	Soddisfatto
RFO-4.1.4	Soddisfatto
RFO-4.1.5	Soddisfatto
RFO-4.1.6	Soddisfatto
RFO-4.1.7	Soddisfatto
RFO-4.1.8	Soddisfatto
RFO-4.1.9	Soddisfatto
RFO-4.1.10	Soddisfatto

RFO-4.1.11	Soddisfatto
RFO-4.1.12	Soddisfatto
RFO-4.1.13	Soddisfatto
RFO-4.1.14	Soddisfatto
RFO-4.1.15	Soddisfatto
RFO-4.1.16	Soddisfatto
RFO-4.2	Soddisfatto
RFO-4.3	Soddisfatto
RFO-4.3.1	Soddisfatto
RFO-4.3.2	Soddisfatto
RFO-4.3.2.1	Soddisfatto
RFO-4.3.2.2	Soddisfatto
RFO-4.3.3	Soddisfatto
RFO-4.3.3.1	Soddisfatto
RFO-4.3.3.2	Soddisfatto

tabella 7.2: Tabella del soddisfacimento dei requisti funzionali

Segue una tabella riassuntiva di questa tipologia di requisiti.

Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	44	44	100%
Desiderabili	2	0	0%

tabella 7.3: Tabella riassuntiva del soddisfacimento dei requisiti funzionali

Requisiti qualitativi

Requisito	**Soddisfacimento**
RQO-1	Soddisfatto
RQO-1.1	Soddisfatto
RQO-1.2	Soddisfatto
RQD-2	Non soddisfatto (fallito un test di sitema)

tabella 7.4: Tabella del soddisfacimento dei requisti qualitativi

Segue una tabella riassuntiva di questa tipologia di requisiti.

Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	3	3	100%
Desiderabili	1	0	0%

tabella 7.5: Tabella riassuntiva del soddisfacimento dei requisiti qualitativi

Requisiti di vincolo

Requisito	Soddisfacimento
RVO-1	Soddisfatto
RVO-2	Soddisfatto

tabella 7.6: Tabella del soddisfacimento dei requisti di vincolo

Segue una tabella riassuntiva di questa tipologia di requisiti.

Tipologia	Totali	Soddisfatti	Percentuale
Obbligatori	2	2	100%

tabella 7.7: Tabella riassuntiva del soddisfacimento dei requisiti di vincolo

Requisiti prestazionali

Requisito	Soddisfacimento
RPO-1	Soddisfatto
RPO-1.1	Soddisfatto
RPD-1.2	Soddisfatto
RPD-1.3	Soddisfatto (parzialmente)
RPD-2	Soddisfatto
RPD-2.1	Soddisfatto
RPD-2.2	Soddisfatto
RPD-3	Soddisfatto
RPD-4	Soddisfatto
RPD-5	Soddisfatto

tabella 7.8: Tabella del soddisfacimento dei requisti di prestazionali

Segue una tabella riassuntiva di questa tipologia di requisiti.

Tipologia	Totali	Soddisfatti	Percentuale
Obbligatori	2	2	100%
Desiderabili	8	8	100%

tabella 7.9: Tabella riassuntiva del soddisfacimento dei requisiti prestazionali

Attenzione particolare è stata posta al requisito *RDP-1.2*. Esso richiede che sia possibile effettuare almeno 5-6 catture al secondo. Studi sperimentali condotti in diversi ambienti confermano che le rilevazioni impiegano in media circa 220 millisecondi ad essere elaborato, quindi circa 4,5 riprese al secondo. È però possibile in ogni caso far effettuare un numero arbitrario di rilevazioni, che verranno elaborate su *thread* paralleli, e anche se la loro elaborazione richiede più di un secondo è quindi possibile effettuare le

5-6 riprese a secondo richieste a patto di essere disposti ad aspettare qualche tempo per la loro elaborazione. Per questo il requisito è segnalato come parzialmente soddisfatto.

Soddisfacimento requisiti

Segue una tabella generale che riassume la copertura di tutti i requisiti.

Tipologia	Totali	Soddisfatti	Percentuale
Obbligatori	51	51	100%
Desiderabili	11	8	72,7%

tabella 7.10: Tabella riassuntiva del soddisfacimento di tutti i requisiti

7.6 Conoscenze acquisite

Dal punto di vista formativo l'attività di *stage* è stata estremamente positiva. Ha arricchito il mio bagaglio personale di competenze professionali.

La richiesta di app *mobile* è in continuo aumento. Per questo l'apprendimento della progettazione e sviluppo delle applicazioni mobili *Android* è certamente una pietra miliare nel campo *IT* in questi anni.

L'approccio ad una tecnologia sperimentale ed ancora di nicchia come *Tango Project* crea dei vantaggi in ambito occupazionale in quanto gli sviluppatori non sono molti. Inoltre il rilascio del primo *smartphone* commerciale dotato dei sensori *Tango* è stato annunciato da un noto marchio per l'autunno 2016; se dovesse prendere campo anche in ambito *customer* ci sarebbe certamente una grande richiesta di sviluppatori con esperienza vista la scarsità di applicazioni dedicate a questo tipo di *Hardware*. Altro aspetto positivo è stato l'inserimento all'interno della comunità degli sviluppatori *Tango* sia su *StackOverflow* che su *Google plus*; lo studente ha avuto modo di confrontarsi con addetti *Google* e con altri sviluppatori sia in ambito accademico/di ricerca che in ambito industriale.

In ambito aziendale si è usato *Java* come linguaggio di programmazione ed *Android Studio* come *IDE*. La curva di apprendimento di questi strumenti è stata piuttosto rapida grazie all'esperienza già maturata in ambito accademico con *Java* ed *ItelliJ*, su cui è basato *Android Studio*. Più complessa si è rivelata l'assimilazione e la comprensione del *Framework Jni*, sia a causa della sua intrinseca complessità sia al fatto che *Android Studio* lo supporta solo in *release* sperimentale. Infatti in azienda è stato realizzato sono un piccolo prototipo che dimostra l'agibilità di questa via, ma poi la tecnologia *Jni* è stata abbandonata in favore di uno sviluppo interamente *Java*. In ogni caso molte applicazioni *mobile*, specialmente in ambito grafico, ne fanno uso, quindi ai fini del curriculum si è rivelata comunque un'esperienza fruttuosa.

In generale ritengo l'approccio a librerie grafiche sia *mobile* che per *PC* estremamente interessante ai fini della formazione personale.

L'apprendimento delle potenzialità della libreria *PCL* e la gestione dei *Point Cloud* è altrettanto importante, anche perché è uno dei pochi ambiti in cui l'Italia spicca in ambito di *Computer* grafica.

Appendice A

Specifica Tecnica

A.1 Metodo e formalismo di specifica

Nell'esposizione dell'architettura del prodotto si procederà con un approccio di tipo top-down. Si descriverà quindi l'architettura iniziando dal generale ed andando al particolare; descrivendo prima i componenti, per poi descrivere nel dettaglio le singole classi.
Per ogni componente saranno descritti brevemente il tipo, l'obiettivo e la funzione e saranno specificati eventuali figli, classi ed interazioni con altri componenti. Ogni classe sarà dotata di una breve descrizione e ne saranno specificate le responsabilità, le classi ereditate, le sottoclassi e le relazioni con altre classi.

A.2 Legenda

Tutti i diagrammi usano la convenzione di colori descritta nella legenda in figura A.1 al fine di migliorarne la leggibilità. I livelli di annidamento sono da intendere per la totale struttura dei package e non solo per il singolo schema.
Si noti in particolare che le classi e componenti di colore arancio rappresentano classi di librerie esterne al sistema, ma che vengono talora rappresentate per maggiore chiarezza.

figura A.1: Legenda

A.3 Architettura generale

L'architettura generale è di tipo *Client-Server*, la comunicazione avviene tramite semplici richieste *http* e alcune notifiche vengono inviate tramite *FireBase*.
Questo documento tuttavia esporrà solamente la parte di progettazione riguardante l'applicativo lato *client*.
L'architettura generale della applicazione *Android* è di tipo *MVP* ovvero *Model View Presenter*. Questo genere di architettura è stato scelto alla luce delle *Android Best Practices*.
Il *Model* contiene tutta la *business logic* dell'applicazione.
Il *Presenter* si occupa sia di osservare il modello che di aggiornare/osservare la vista. Nel caso specifico il *Presenter* è composto dall'insieme delle *activity* necessarie al sistema.
La *View* è composta da *file xml* che rappresentano *template* di visualizzazione e sono completamente passivi. Per questo non verranno trattati nella sezione A.4.
Il diagramma in figura A.2 rappresenta informalmente la struttura generale del sistema e non rispecchia la reale nomenclatura e struttura del *package*.

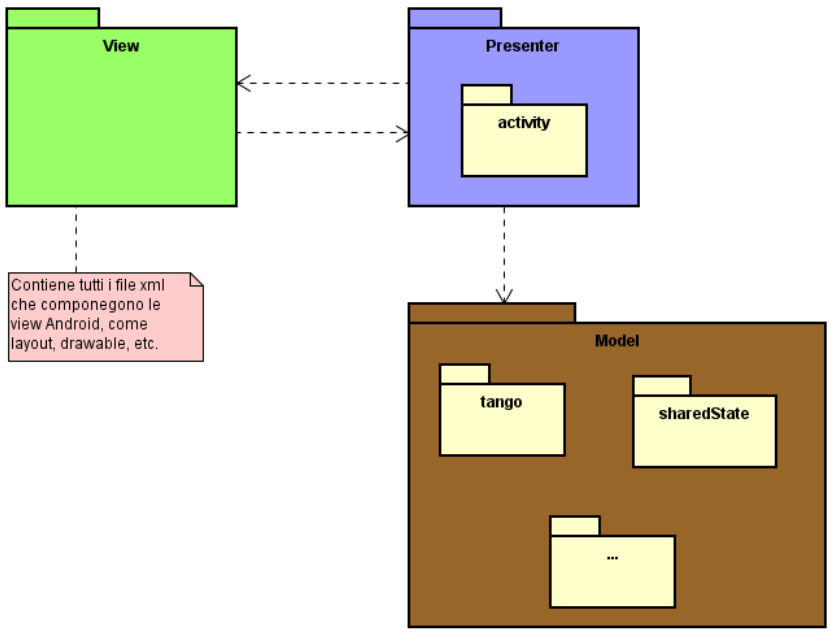

figura A.2: Architettura generale del sistema

A.4 Componenti e classi

La volontà di realizzare un prototipo ha avuto molto peso anche nella fase di progettazione. Per questo semplicità e rapidità di sviluppo sono stati obiettivi prioritari, anche sacrificando parzialmente riuso e testabilità. Ciò è stato ritenuto accettabile in quanto il prodotto non ha lo scopo di essere incrementato fino a divenire un prodotto finito,

ma solo quello di essere premessa sperimentale/prototipale per un progetto futuro.
Di seguito viene riportata la lista delle componenti del sistema.

A.4.1 Samba

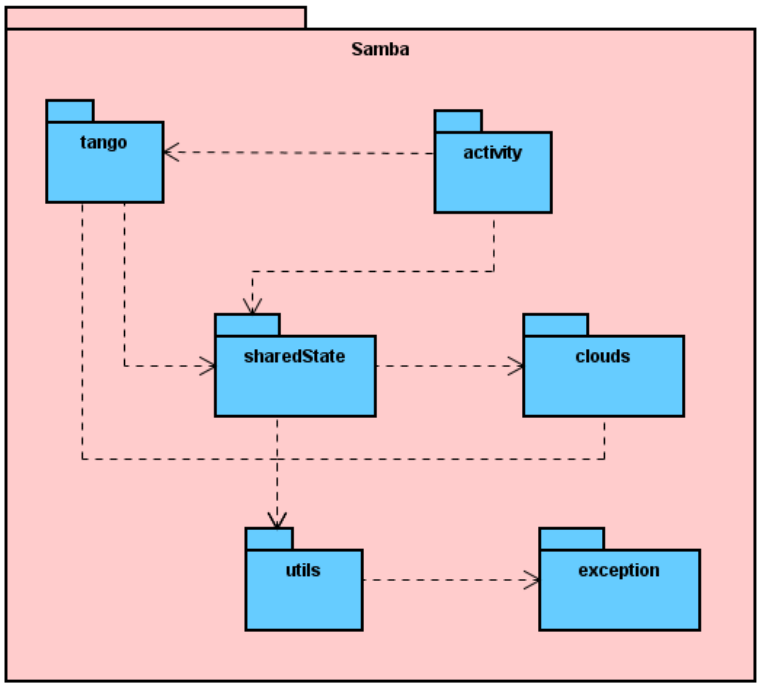

figura A.3: Componente Samba

Descrizione

È il livello principale del sistema lato *tablet*.

Package figli

- Samba.activity (A.4.2)
- Samba.tango (A.4.9)
- Samba.sharedState (A.4.18)
- Samba.clouds (A.4.21)
- Samba.utils (A.4.38)
- Samba.exception

A.4.2 Samba.activity

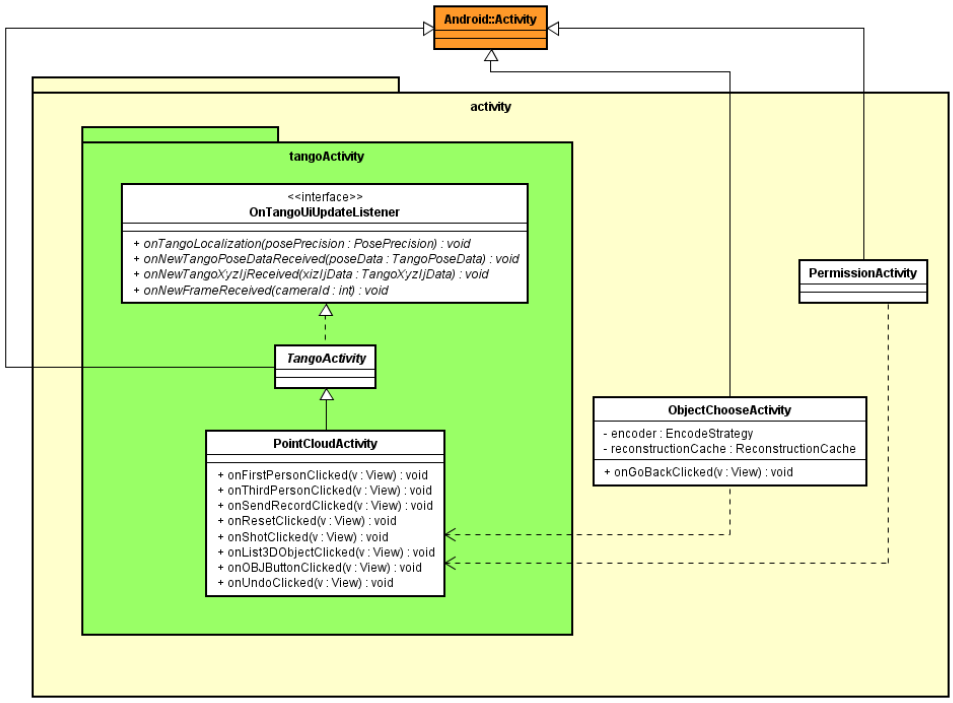

figura A.4: Componente Samba.activity

Descrizione

Questo *package* contiene tutte le *activity* necessarie per l'applicazione. Contiene inoltre la definizione di una interfaccia per le attività che vogliono fare uso dei vari *manager* messi a disposizione (si veda sezione A.4.9).

Package figli

- Samba.activity.tangoActivity (A.4.3)

Classi

- Samba.activity.PermissionActivity (A.4.8)
- Samba.activity.ObjectChooseActivity (A.4.7)

A.4.3 Samba.activity.tangoActivity

Descrizione

Questo *package* serve a contenere tutte le *activity* che vogliono essere attività Tango, ovvero che vogliono poter usare i *manager* messi a disposizione (si veda sezione A.4.9).

Interfacce

- Samba.activity.tangoActivity.OnTangoUiUpdateListener (A.4.4)

Classi

- Samba.activity.tangoActivity.TangoActivity (A.4.5)
- Samba.activity.tangoActivity.PointCloudActivity (A.4.6)

A.4.4 Samba.activity.tangoActivity.OnTangoUiUpdateListener

Descrizione

Interfaccia che deve essere implementata da tutte le *activity* che vogliono fare uso dei *manager Tango* messi a disposizione (si veda sezione A.4.9). Espone metodi pubblici che possono essere chiamati da altre componenti quando avranno la necessità di notificare qualche cambiamento di stato.

Utilizzo

Viene implementate dalle *activity* che vogliono interagire con il ciclo di vita dei sensori *Tango*. Verrà usata per permettere indirettamente alle componenti che gestiscono i sensori *Tango* di aggiornare la *UI*.

Implementata da

- Samba.activity.tangoActivity.TangoActivity (A.4.5)

Relazioni con altre classi

- `Samba.clouds.utils.poseSanity.PosePrecision`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.34)

A.4.5 Samba.activity.tangoActivity.TangoActivity

Descrizione

Classe astratta che estende *Activity* e implementa l'interfaccia che fornisce i *callback* necessari per permettere ai componenti che interagiscono con il ciclo di vita dei sensori *Tango* di modificare indirettamente la *UI*.

Utilizzo

È utilizzata come superclasse astratta di tutte le attività che vogliono interagire con i sensori *Tango*.

Relazioni con altre classi

- `Samba.sharedState.ReconstructionManager`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.20)
- `Samba.tango.CloudRecorder`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi.(A.4.17)

- `Samba.tango.RajawaliRendererManager`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi.(A.4.10)
- `Samba.tango.RGBBoxManager`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi.(A.4.15)
- `Samba.tango.TangoManager`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.16)
- `Samba.tango.RajawaliRendererManager.RajawaliRendererManager`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.11)

Estesa da

- Samba.activity.tangoActivity.PointCloudActivity (A.4.6)

Interfacce implementate

- Samba.activity.tangoActivity.OnTangoUiUpdateListener (A.4.4)

Classi estese

- android.app.Activity

A.4.6 Samba.activity.tangoActivity.PointCloudActivity

Descrizione

Attività principale dell'applicazione prodotta: fornisce un *render* dei punti, una *preview* della fotocamera e pulsanti per accedere a tutte le altre funzionalità dell'applicazione.

Utilizzo

È usata per gestire il ciclo di vita dell'*activity* principale dell'applicazione.

Relazioni con altre classi

- `Samba.activity.ObjectChooseActivity`: relazione entrante, dipendenza, utilizzo della classe per costruire un *Intent*. (A.4.7)
- `Samba.activity.PermissionActivity`: relazione entrante, dipendenza, utilizzo della classe per costruire un *Intent*. (A.4.8)
- `Samba.sharedState.ReconstructionCache`: relazione uscente, composizione. (A.4.19)
- `Samba.sharedState.ReconstructionManager`: relazione uscente, composizione. (A.4.20)
- `Samba.tango.TangoManager`: relazione uscente, composizione. (A.4.16)
- `Samba.tango.RajawaliRendererManager`: relazione uscente, composizione. (A.4.10)
- `Samba.tango.RGBBoxManager`: relazione uscente, composizione. (A.4.15)

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.tango.RajawaliRendererManager.RajawaliRendererManager`: relazione uscente, composizione. (A.4.11)
- `Samba.tango.CloudRecorder`: relazione uscente, composizione. (A.4.17)

Classi estese

- Samba.activity.tangoActivity.TangoActivity (A.4.5)

A.4.7 Samba.activity.ObjectChooseActivity

Descrizione

Attività che può leggere e scrivere su disco, compila la lista dei *File pcd* salvati e permette all'utente di compiere diverse azioni su questi ultimi.

Utilizzo

Viene usata quando l'utente richiede di visualizzare la lista dei *file pcd* salvati su disco, oppure quando vuole caricarli/eliminarli/spedirli al *Server*.

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione uscente, dipendenza, utilizzo della classe per costruire un *Intent*. (A.4.6)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.sharedState.ReconstructionCache` relazione uscente, composizione. (A.4.19)
- `Samba.utils.encoding.EncodeStrategy`: relazione entrante, composizione. (A.4.40)

Classi estese

- android.app.Activity

A.4.8 Samba.activity.PermissionActivity

Descrizione

Attività che ha il solo compito di richiedere all'utente i permessi di utilizzare l'*Area Learning*. *Google* fornire un *Intent* apposito per ottenere tali permessi ed essi devono essere assolutamente garantiti dall'utente prima dell'inizio del processo di localizzazione. Per questo sono richiesti in una attività a parte e che viene lanciata precedentemente rispetto all'attività principale.

Utilizzo

Viene lanciata ad ogni avvio dell'applicazione allo scopo di richiedere i permessi, in caso l'utente non li abbia ancora garantiti, controllarne la presenza altrimenti.

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione uscente, dipendenza, utilizzo della classe per costruire un *Intent*. (A.4.6)

Classi estese

- android.app.Activity

A.4.9 Samba.tango

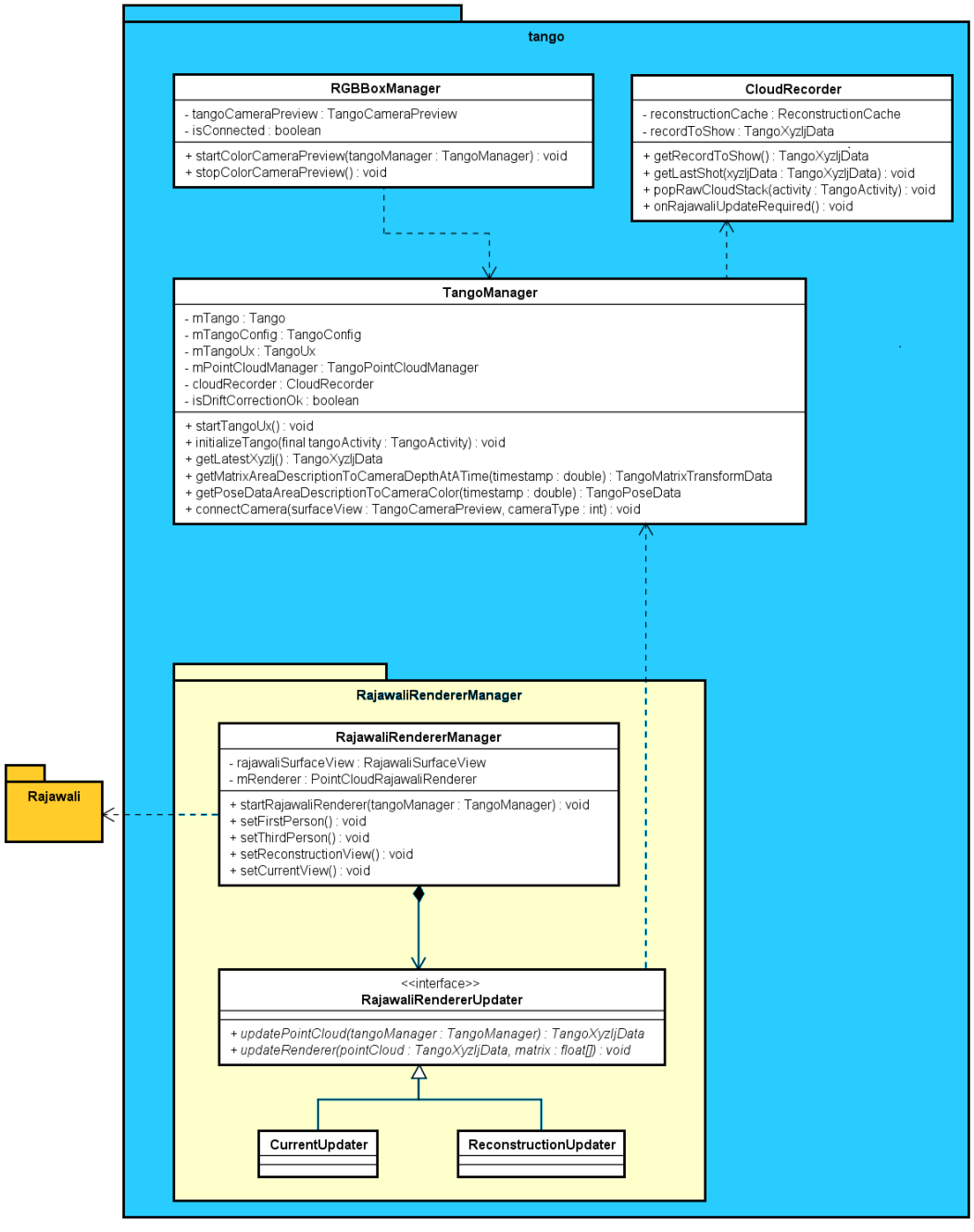

figura A.5: Componente Samba.tango

Descrizione

Questo *package* contiene un insieme di classi che possono essere usate per interagire con il ciclo di vita dei sensori *Tango* e del *renderer* dei punti. Questi *manager* possono essere usati per gestire la *business logic* di una applicazione *Tango* separandola dalla sua rappresentazione grafica.

Package figli

- Samba.tango.RajawaliRendererManager (A.4.10)

Classi

- Samba.tango.RGBBoxManager (A.4.15)
- Samba.tango.CloudRecorder (A.4.17)
- Samba.tango.TangoManager (A.4.16)

A.4.10 Samba.tango.RajawaliRendererManager

Descrizione

Package che contiene il *manager* per gestire il *rendering* dei punti tramite la libreria *Rajawali*.

Interfacce

- Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater (A.4.12)

Classi

- Samba.tango.RajawaliRendererManager.RajawaliRendererManager (A.4.11)
- Samba.tango.RajawaliRendererManager.CurrentUpdater (A.4.13)
- Samba.tango.RajawaliRendererManager.ReconstructionUpdater (A.4.14)

A.4.11 Samba.tango.RajawaliRendererManager.RajawaliRendererManager

Descrizione

Questa classe permette di integrare un servizio di *rendering* di *Point Cloud* all'interno del ciclo di vita di una applicazione *Andorid*. Oltre alla visualizzazione espone metodi per cambiare la modalità del *render*, e di effettuare qualche azione sullo stesso.

Utilizzo

È utilizzata per fornire un *render* nell'attività principale dell'applicazione prodotta. (Come quello in figura 1.6 in tutta la parte destra dello schermo)

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.activity.tangoActivity.TangoActivity`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.5)
- `Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater`: relazione uscente, composizione. (A.4.12)

A.4.12 Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater

Descrizione

Interfaccia che rappresenta il componente *Strategy* del *Design Pattern Strategy*.

Utilizzo

È usata per alternare la modalità del *render* tra la rappresentazione in tempo reale dei dati del sensore e quella della ricostruzione corrente.

Relazioni con altre classi

- `Samba.tango.RajawaliRendererManager.RajawaliRendererManager`: relazione entrante, composizione. (A.4.11)
- `Samba.tango.TangoManager`: relazione uscente, dipendenza. (A.4.16)

Implementata da

- Samba.tango.RajawaliRendererManager.CurrentUpdater (A.4.13)
- Samba.tango.RajawaliRendererManager.ReconstructionUpdater (A.4.14)

A.4.13 Samba.tango.RajawaliRendererManager.CurrentUpdater

Descrizione

Implementazione di *RajawaliRendererUpdater* che rappresenta la visione in tempo reale dei dati del sensore.

Utilizzo

È usata per impostare il *render* alla modalità in tempo reale.

Interfacce implementate

- Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater (A.4.12)

A.4.14 Samba.tango.RajawaliRendererManager.ReconstructionUpdater

Descrizione

Implementazione di *RajawaliRendererUpdater* che rappresenta la visione del *Point Cloud* correntemente ricostruito.

Utilizzo

È usata per impostare il *render* alla modalità ricostruzione.

Interfacce implementate

- Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater (A.4.12)

A.4.15 Samba.tango.RGBBoxManager

Descrizione

Questa classe permette di integrare un servizio di *preview* della fotocamera all'interno del ciclo di vita di una applicazione *Andorid*.

Utilizzo

È utilizzata per fornire una *preview* della fotocamera nell'attività principale dell'applicazione prodotta. (Come quella in figura 1.6 in basso a sinistra)

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.activity.tangoActivity.TangoActivity`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.5)
- `Samba.tango.TangoManager`: relazione uscente, dipendenza. (A.4.16)

A.4.16 Samba.tango.TangoManager

Descrizione

Questa classe permette di integrare i servizi *Tango* all'interno del ciclo di vita di una applicazione *Andorid*. Inoltre imposta correttamente il *framework TangoUx* e fornisce metodi per ricavare statistiche e dati algebrici.

Utilizzo

È utilizzata per fornire all'attività principale dell'applicazione prodotta la possibilità di sfruttare i servizi *Tango*.

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.activity.tangoActivity.TangoActivity`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.5)
- `Samba.tango.RGBBoxManager`: relazione entrante, dipendenza. (A.4.15)
- `Samba.tango.RajawaliRendererManager.RajawaliRendererUpdater`: relazione entrante, dipendenza. (A.4.12)

- `Samba.tango.CloudRecorder`: relazione uscente, dipendenza. (A.4.17)
- `Samba.clouds.utils.poseSanity.PoseSanityChecker`: relazione uscente, composizione. (A.4.31)
- `Samba.clouds.utils.poseSanity.PoseDriftCorrectionStatus`: relazione uscente, dipendenza. (A.4.33)

A.4.17 Samba.tango.CloudRecorder

Descrizione

Mantiene la ricostruzione corrente in un formato comprensibile dal *renderer*.

Utilizzo

Viene usata per effettuare il *rendering* del *Point Cloud* della ricostruzione corrente.

Relazioni con altre classi

- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.activity.tangoActivity.TangoActivity`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.5)
- `Samba.tango.TangoManager`: relazione entrante, dipendenza. (A.4.16)
- `Samba.sharedState.ReconstructionCache`: relazione uscente, composizione. (A.4.19)
- `Samba.clouds.utils.SambaRawData`: relazione uscente, dipendenza. (A.4.37)
- `Samba.utils.encoding.EncodeStrategy`: relazione uscente, composizione. (A.4.40)

A.4.18 Samba.sharedState

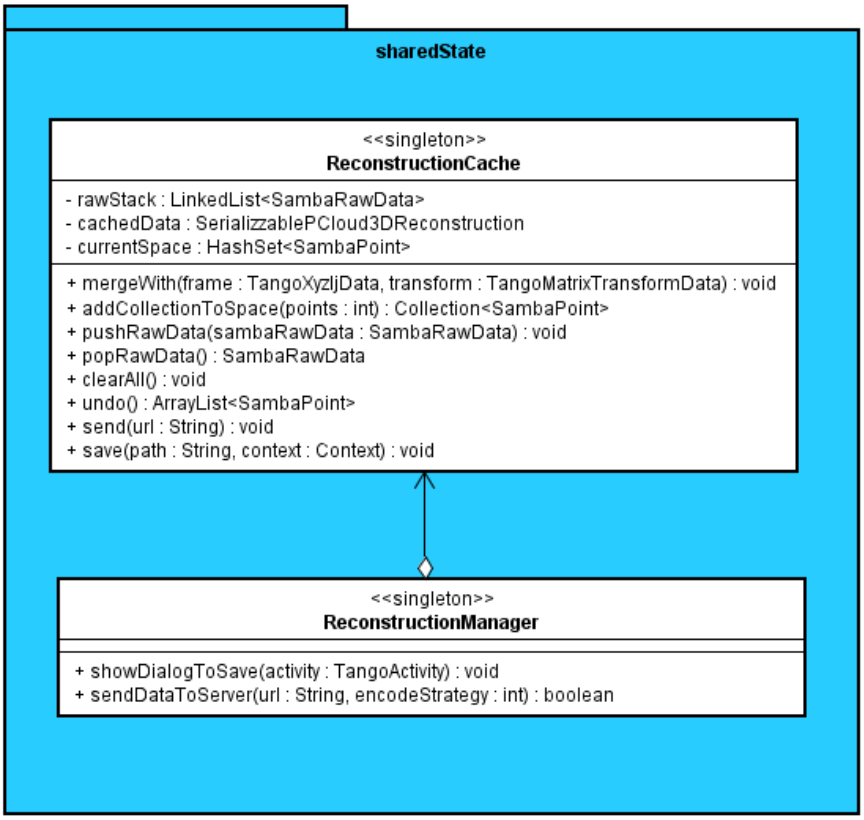

figura A.6: Componente Samba.sharedState

Descrizione

Questo *package* contiene un *singleton* che mantiene tutti gli stati condivisi tra i vari servizi e una classe di utilità.

Classi

- Samba.sharedState.ReconstructionCache (A.4.19)
- Samba.sharedState.ReconstructionManager (A.4.20)

A.4.19 Samba.sharedState.ReconstructionCache

Descrizione

Questo *singleton* mantiene tutte le informazioni condivise tra i vari servizi asincroni presenti nell'applicazione. Fornisce tutti i metodi necessari per accedervi controllatamente, e si occupa anche di sincronizzare gli accessi stessi dove necessario.

Utilizzo

È usata dai servizi presenti nell'applicazione per salvare e modificare i loro stati condivisi.

Relazioni con altre classi

- `Samba.sharedState.ReconstructionManager`: relazione entrante, aggregazione. (A.4.20)
- `Samba.activity.ObjectChooseActivity`: relazione entrante, composizione. (A.4.7)
- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.tango.CloudRecorder`: relazione entrante, composizione. (A.4.17)
- `Samba.utils.services.MergingService`: relazione entrante, composizione. (A.4.49)
- `Samba.utils.services.VoxelService`: relazione entrante, composizione. (A.4.50)
- `Samba.clouds.reconstruction.UndoableReconstruction`: relazione uscente, dipendenza, controllo di tipo. (A.4.25)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.clouds.utils.SambaRawData`: relazione uscente, composizione. (A.4.37)
- `Samba.utils.sending.ReconstructionSender`: relazione uscente, composizione. (A.4.46)
- `Samba.utils.sending.InternalStorageSender`: relazione uscente, composizione. (A.4.47)
- `Samba.utils.sending.ArrListSender`: relazione uscente, composizione. (A.4.45)
- `Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction`: relazione uscente, composizione. (A.4.26)

A.4.20 Samba.sharedState.ReconstructionManager

Descrizione

Singleton di utilità che fornisce delle *macro* per sequenze di operazioni effettuate ricorrentemente su *ReconstructionCache*.

Utilizzo

Viene usato da alcuni componenti per effettuare complesse operazioni sullo stato condiviso.

Relazioni con altre classi

- `Samba.sharedState.ReconstructionCache`: relazione uscente, aggregazione. (A.4.19)
- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, composizione. (A.4.6)
- `Samba.activity.tangoActivity.TangoActivity`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.5)

A.4.21 Samba.clouds

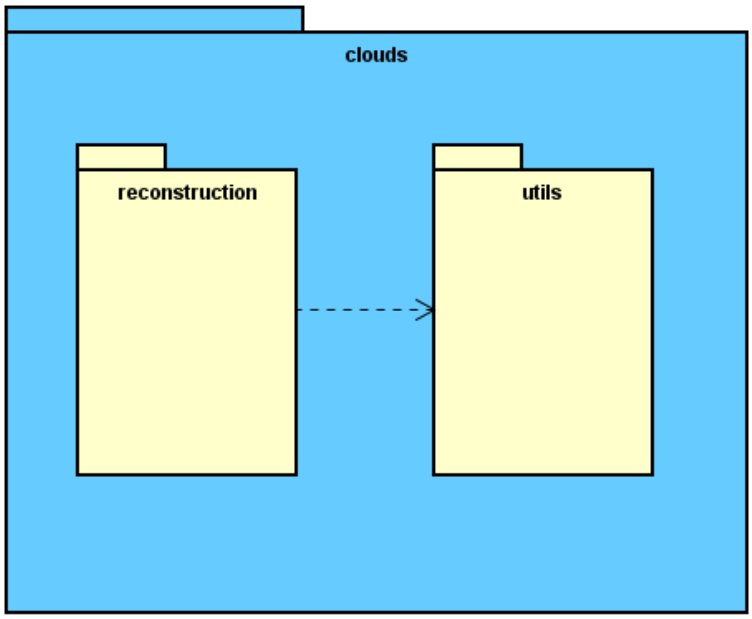

figura A.7: Componente Samba.clouds

Descrizione

Questo *package* contiene tutte le strutture dati usate per rappresentare internamente i *Point Cloud*.

Package figli

- Samba.clouds.reconstruction (A.4.22)
- Samba.clouds.utils (A.4.29)

A.4.22 Samba.clouds.reconstruction

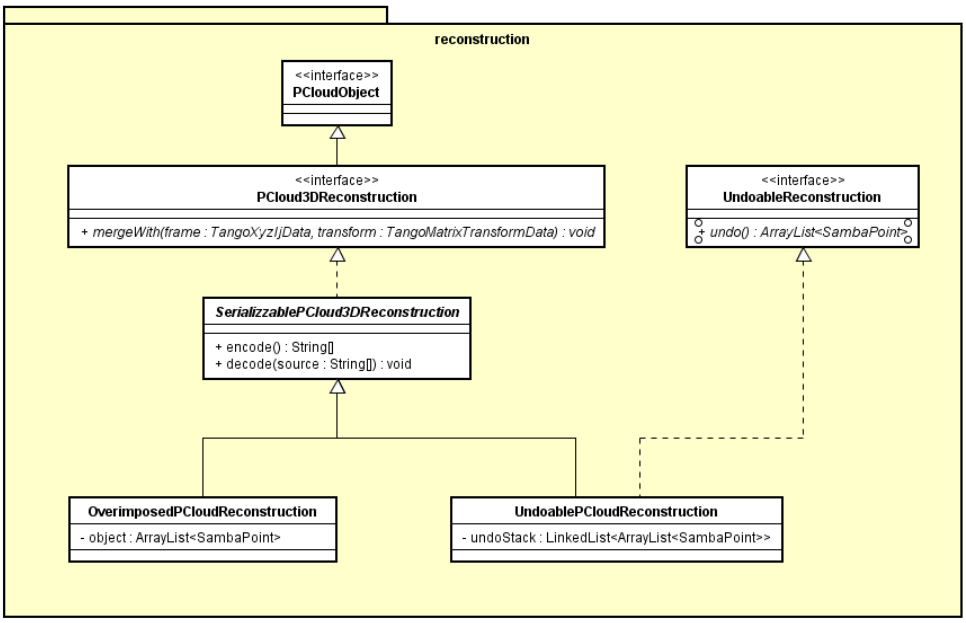

figura A.8: Componente Samba.reconstruction

Descrizione

Per motivi di ottimizzazione si è cercato di lasciare il più possibile aperte le possibilità di modifica della rappresentazione interna dei *Point Cloud* ricostruiti. Questo *package* contiene tutte le classi che rappresenta un *Point Cloud* ricostruito.

Interfacce

- Samba.clouds.reconstruction.PCloudObject (A.4.23)
- Samba.clouds.reconstruction.PCloud3DReconstruction (A.4.24)
- Samba.clouds.reconstruction.UndoableReconstruction (A.4.25)

Classi

- Samba.clouds.reconstruction.OverimposedPCloudReconstruction (A.4.27)
- Samba.clouds.reconstruction.UndoablePCloudReconstruction (A.4.28)

A.4.23 Samba.clouds.reconstruction.PCloudObject

Descrizione

Rappresenta un generico oggetto 3D rappresentato tramite *Point Cloud*.

Utilizzo

È utilizzato come interfaccia alla base della gerarchia dei possibili *Point Cloud*.

Estesa da

- Samba.clouds.reconstruction.PCloud3DReconstruction (A.4.24)

A.4.24 Samba.clouds.reconstruction.PCloud3DReconstruction

Descrizione

Interfaccia che rappresenta un generico *Point Cloud* in grado di essere sovrapposto ad altre nuvole di punti per creare un oggetto tridimensionale completo.

Utilizzo

È usata come interfaccia alla base della gerarchia delle possibili ricostruzioni 3D.

Interfacce estese

- Samba.clouds.reconstruction.PCloudObject (A.4.23)

Implementata da

- Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction (A.4.26)

A.4.25 Samba.clouds.reconstruction.UndoableReconstruction

Descrizione

Interfaccia che espone i metodi necessari per permettere ad una ricostruzione di effettuare operazioni di *undo*. Una *UndoableReconstruction* **non** è una *PCloud3DReconstrucion*.

Utilizzo

È implementata dalle classi che rappresentano una ricostruzione 3D e che necessitano operazioni di *undo*.

Relazioni con altre classi

- `Samba.sharedState.ReconstructionCache`: relazione entrante, dipendenza, controllo di tipo. (A.4.19)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)

Implementata da

- Samba.clouds.reconstruction.UndoablePCloudReconstruction (A.4.28)

A.4.26 Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction

Descrizione

Classe astratta che rappresenta un generico *Point Cloud* in grado di essere sovrapposto ad altre nuvole di punti per creare un oggetto tridimensionale completo ed che può essere inoltre serializzato per salvarlo su disco o inviarlo ad un *Server*.

Utilizzo

È usata come classe astratta base della gerarchia delle possibili ricostruzioni 3D.

Relazioni con altre classi

- `Samba.utils.encoding.EncodeStrategy`: relazione uscente, composizione. (A.4.40)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, composizione. (A.4.19)
- `Samba.utils.sending.ReconstructionSender`: relazione entrante, composizione. (A.4.46)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)

Interfacce implementate

- Samba.clouds.reconstruction.PCloud3DReconstruction (A.4.24)

Estesa da

- Samba.clouds.reconstruction.OverimposedPCloudReconstruction (A.4.27)
- Samba.clouds.reconstruction.UndoablePCloudReconstruction (A.4.28)

A.4.27 Samba.clouds.reconstruction.OverimposedPCloudReconstruction

Descrizione

Rappresenta una ricostruzione 3D in cui i *Point Cloud* sono semplicemente sovrapposti senza alcuna ulteriore ottimizzazione.

Utilizzo

Attualmente non è usata in favore di *UndoablePCloudReconstruction* (vedi A.4.28).

Relazioni con altre classi

- `Samba.clouds.utils.Vector4`: relazione uscente, dipendenza. (A.4.35)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)

Classi estese

- Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction (A.4.26)

A.4.28 Samba.clouds.reconstruction.UndoablePCloudReconstruction

Descrizione

Rappresenta una ricostruzione 3D in cui i *Point Cloud* sono semplicemente sovrapposti ma con la possibilità di annullare un certo numero di operazioni.

Utilizzo

È usata come tipo preferito per contenere i dati dell'oggetto ricostruito, una volta elaborati, ma non ancora *voxellati*.

Relazioni con altre classi

- `Samba.clouds.utils.Vector4`: relazione uscente, dipendenza. (A.4.35)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)

Interfacce implementate

- Samba.clouds.reconstruction.UndoableReconstruction (A.4.25)

Classi estese

- Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction (A.4.26)

A.4.29 Samba.clouds.utils

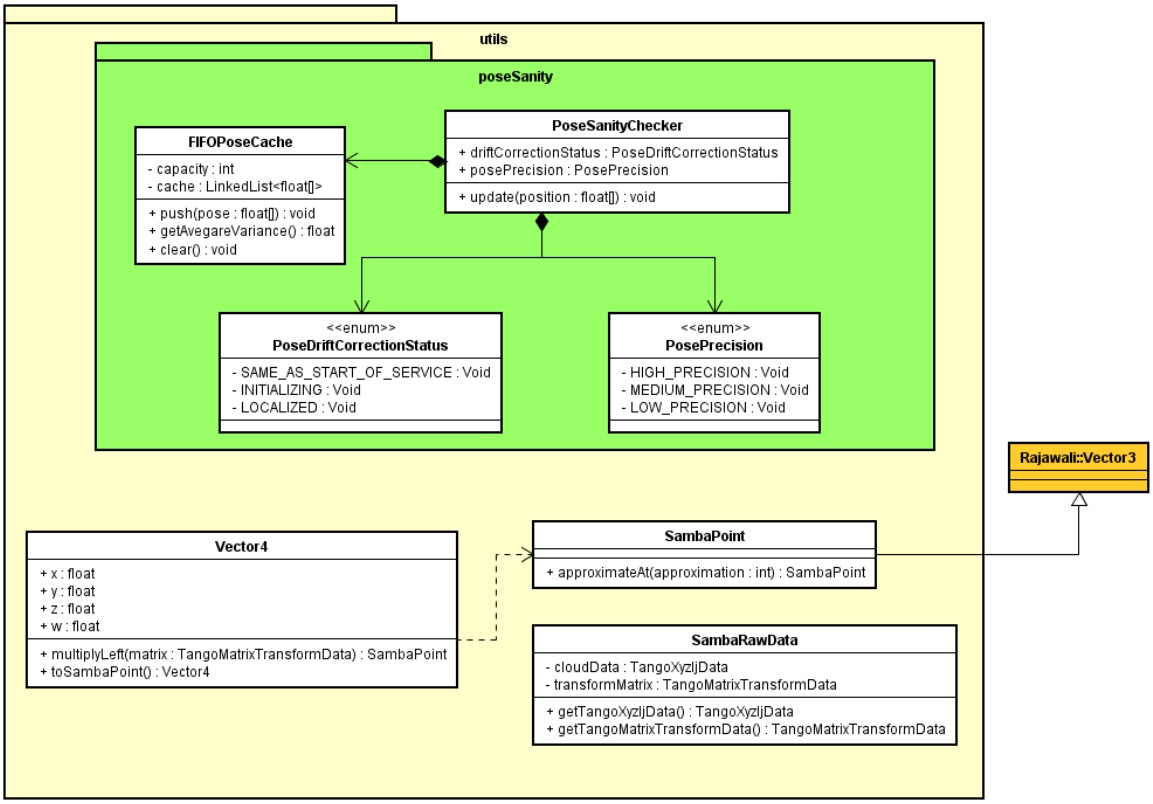

figura A.9: Componente Samba.clouds.utils

Descrizione

Contiene tutte le strutture dati di utilità usate per rappresentazione interna, come punti, *Point Cloud* uniti alla propria matrice associata etc.

Package figli

- Samba.clouds.utils.poseSanity (A.4.30)

Classi

- Samba.clouds.utils.Vector4 (A.4.35)
- Samba.clouds.utils.SambaPoint (A.4.36)
- Samba.clouds.utils.SambaRawData (A.4.37)

A.4.30 Samba.clouds.utils.poseSanity

Descrizione

Questo *package* contiene le classi necessarie per effettuare il *sanity check* della fase di localizzazione.

Classi

- Samba.clouds.utils.poseSanity.PoseSanityChecker (A.4.31)
- Samba.clouds.utils.poseSanity.FIFOPoseCache (A.4.32)

Enumerazioni

- Samba.clouds.utils.poseSanity.PoseDriftCorrectionStatus (A.4.33)
- Samba.clouds.utils.poseSanity.PosePrecision (A.4.34)

A.4.31 Samba.clouds.utils.poseSanity.PoseSanityChecker

Descrizione

Questa classe fornisce un *sanity check* della fase di localizzazione.

Utilizzo

È utilizzata durante la fase di localizzazione per tenere traccia dello stato della stessa. Inoltre è usata per fornire una stima di quanto la localizzazione stessa sia avvenuta in maniera precisa.

Relazioni con altre classi

- `Samba.clouds.utils.poseSanity.FIFOPoseCache`: relazione uscente, composizione. (A.4.32)
- `Samba.clouds.utils.poseSanity.PoseDriftCorrectionStatus`: relazione uscente, composizione. (A.4.33)
- `Samba.clouds.utils.poseSanity.PosePrecision`: relazione uscente, composizione. (A.4.34)
- `Samba.tango.TangoManager`: relazione entrante, composizione. (A.4.16)

A.4.32 Samba.clouds.utils.poseSanity.FIFOPoseCache

Descrizione

Fornisce una coda *FIFO* in grado di tenere in memoria le ultime n posizioni rilevate e usarle per dei calcoli statistici in maniera efficiente.

Utilizzo

È usata come classe di utilità per `PoseSanityChecker` (A.4.31).

Relazioni con altre classi

- `Samba.clouds.utils.poseSanity.PoseSanityChecker`: relazione entrante, composizione. (A.4.31)

A.4.33 Samba.clouds.utils.poseSanity.PoseDriftCorrectionStatus

Descrizione

Enumerazione che rappresenta i possibili stati della *Drift Correction*, quelli discussi in 1.3.4 alla voce "Istruzioni per l'utente".

Valori

- `SAME_AS_START_OF_SERVICE`
- `INITIALIZING`
- `LOCALIZED`

Relazioni con altre classi

- `Samba.clouds.utils.poseSanity.PoseSanityChecker`: relazione entrante, composizione. (A.4.31)
- `Samba.tango.TangoManager`: relazione entrante, dipendenza. (A.4.16)

A.4.34 Samba.clouds.utils.poseSanity.PosePrecision

Descrizione

Enumerazione che rappresenta i possibili livelli di precisione a cui è avvenuta la localizzazione.

Valori

- `HIGH_PRECISION`
- `MEDIUM_PRECISION`
- `LOW_PRECISION`

Relazioni con altre classi

- `Samba.clouds.utils.poseSanity.PoseSanityChecker`: relazione entrante, composizione. (A.4.31)
- `Samba.activity.tangoActivity.OnTangoUiUpdateListener`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.4)

A.4.35 Samba.clouds.utils.Vector4

Descrizione

Classe che rappresenta un vettore di quattro dimensioni. Offre i metodi necessari per i calcoli del sistema.

Utilizzo

È usato per adattare l'interfaccia di un vettore a tre dimensioni quando è necessario moltiplicarlo per una matrice di trasformazione di un *Point Cloud* (che ha quattro dimensioni).

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.36)
- `Samba.clouds.reconstruction.OverimposedPCloudReconstruction`: relazione entrante, dipendenza. (A.4.27)
- `Samba.clouds.reconstruction.UndoablePCloudReconstruction`: relazione entrante, dipendenza. (A.4.28)

A.4.36 Samba.clouds.utils.SambaPoint

Descrizione

Questa classe rappresenta un punto all'interno di un *PointCloud*. È un *wrapper* di `Vector3` della libreria *Rajawali*. Espone quindi le stesse funzionalità ma ne aggiunge altre di indispensabili, come essere confrontabile ed essere *parcellabile*. È stato scelto di creare questo *wrapper* proprio perché `Vector3` non forniva queste caratteristiche.

Utilizzo

È utilizzata ovunque sia richiesto il concetto di *punto* all'interno del sistema.

Relazioni con altre classi

- `Samba.clouds.utils.Vector4`: relazione entrante, dipendenza. (A.4.35)
- `Samba.activity.ObjectChooseActivity`: relazione entrante, dipendenza. (A.4.7)
- `Samba.activity.tangoActivity.PointCloudActivity`: relazione entrante, dipendenza. (A.4.6)
- `Samba.clouds.reconstruction.UndoableReconstruction`: relazione entrante, dipendenza. (A.4.25)
- `Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction`: relazione entrante, dipendenza. (A.4.26)
- `Samba.clouds.reconstruction.OverimposedPCloudReconstruction`: relazione entrante, dipendenza. (A.4.27)
- `Samba.clouds.reconstruction.UndoablePCloudReconstruction`: relazione entrante, dipendenza. (A.4.28)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, dipendenza. (A.4.19)
- `Samba.utils.encoding.EncodeStrategy`: relazione entrante, dipendenza. (A.4.40)

- `Samba.utils.encoding.XyzEncoder`: relazione entrante, dipendenza. (A.4.42)
- `Samba.utils.encoding.PcdEncoder`: relazione entrante, dipendenza. (A.4.41)
- `Samba.utils.sending.ArrListSender`: relazione entrante, dipendenza. (A.4.45)
- `Samba.utils.sending.InternalStorageSender`: relazione entrante, dipendenza. (A.4.47)
- `Samba.utils.services.VoxelService`: relazione entrante, dipendenza. (A.4.50)

Classi estese

- org.rajawali3d.math.vector.Vector3

A.4.37 Samba.clouds.utils.SambaRawData

Descrizione

Classe che permette di impacchettare assieme un `TangoXyzIjData` (dati del sensore di profondità *Tango*) e la sua matrice di rotazione.

Utilizzo

È usata ovunque ci sia bisogno di mantenere dei dati "grezzi" prima di effettuare rotazione e traslazione.

Relazioni con altre classi

- `Samba.sharedState.ReconstructionCache`: relazione entrante, composizione. (A.4.19)
- `Samba.tango.CloudRecorder`: relazione entrante, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.17)
- `Samba.utils.services.MergingService`: relazione entrante, dipendenza, utilizzo interno ad un metodo. (A.4.49)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, dipendenza. (A.4.19)

A.4.38 Samba.utils

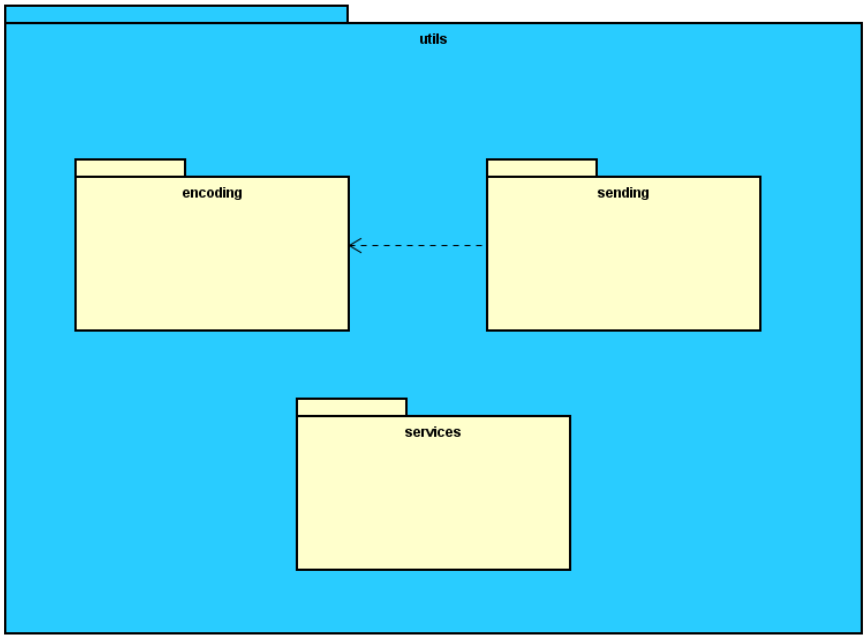

figura A.10: Componente Samba.utils

Descrizione

Questo *package* contiene classi di utilità generale.

Package figli

- Samba.utils.encoding (A.4.39)
- Samba.utils.sending (A.4.43)
- Samba.utils.services (A.4.48)

A.4.39 Samba.utils.encoding

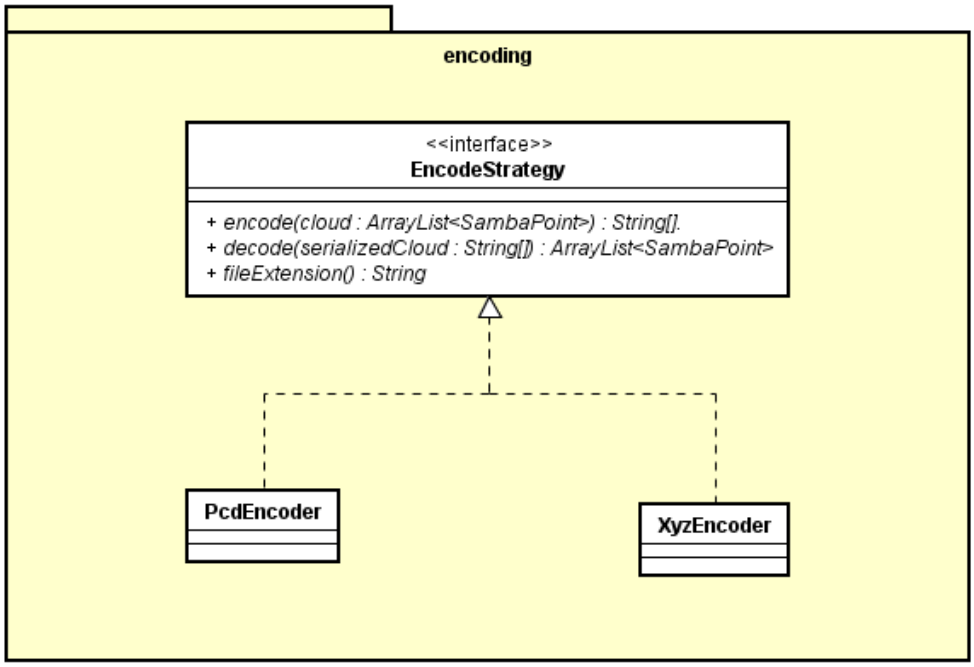

figura A.11: Componente Samba.utils.encoding

Descrizione

Questo *package* contiene le classi necessarie serializzare e deserializzare i *Point Cloud*.

Interfacce

- Samba.utils.encoding.EncodeStrategy (A.4.40)

Classi

- Samba.utils.encoding.PcdEncoder (A.4.41)
- Samba.utils.encoding.XyzEncoder (A.4.42)

A.4.40 Samba.utils.encoding.EncodeStrategy

Descrizione

Interfaccia che rappresenta la strategia con cui effettuare l'*encoding* di un *Point Cloud*.

Utilizzo

È usata come componente *Strategy* di uno *Strategy Pattern*.

Relazioni con altre classi

- `Samba.activity.ObjectChooseActivity`: relazione entrante, composizione. (A.4.7)
- `Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction`: relazione entrante, composizione. (A.4.26)
- `Samba.utils.sending.ArrListSender`: relazione entrante, composizione. (A.4.45)
- `Samba.utils.sending.ReconstructionSender`: relazione entrante, composizione. (A.4.46)
- `Samba.utils.sending.InternalStorageSender`: relazione entrante, composizione. (A.4.47)
- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.36)

Implementata da

- Samba.utils.encoding.PcdEncoder (A.4.41)
- Samba.utils.encoding.XyzEncoder (A.4.42)

A.4.41 Samba.utils.encoding.PcdEncoder

Descrizione

Implementazione della strategia di *encoding* che serializza i *Point Cloud* in formato *pcd*.

Utilizzo

È il formato preferito dal sistema per salvare e spedire i *file*.

Interfacce implementate

- Samba.utils.encoding.EncodeStrategy (A.4.40)

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.36)

A.4.42 Samba.utils.encoding.XyzEncoder

Descrizione

Implementazione della strategia di *encoding* che serializza i *Point Cloud* in formato *xyz*.

Utilizzo

È un formato non utilizzato nel sistema.

Interfacce implementate

- Samba.utils.encoding.EncodeStrategy (A.4.40)

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza, utilizzo come parametro di uno o più metodi. (A.4.36)

A.4.43 Samba.utils.sending

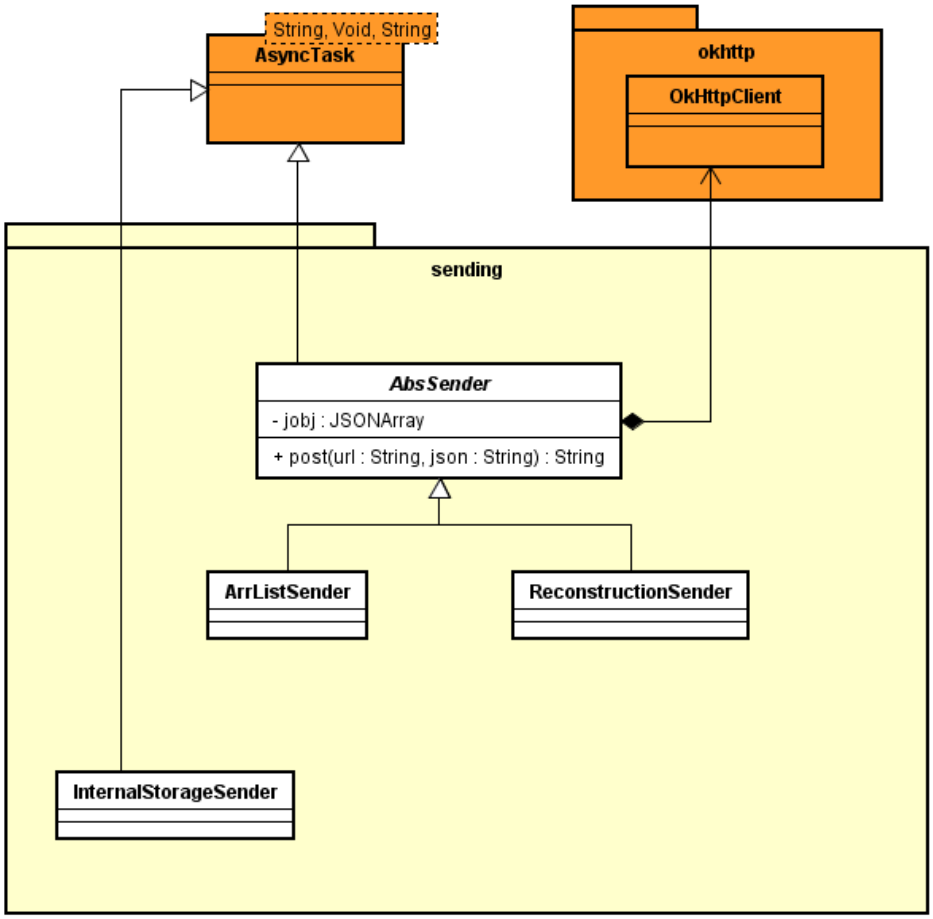

figura A.12: Componente Samba.utils.sending

Descrizione

Questo *package* contiene delle *AsyncTask* che hanno il compito di spedire un *Point Cloud* serializzato ad un *Server* oppure di salvarlo su disco.

Classi

- Samba.utils.sending.AbsSender (A.4.44)
- Samba.utils.sending.ArrListSender (A.4.45)
- Samba.utils.sending.ReconstructionSender (A.4.46)
- Samba.utils.sending.InternalStorageSender (A.4.47)

A.4.44 Samba.utils.sending.AbsSender

Descrizione

Classe astratta per inviare un *Point Cloud* tramite *http* in un *thread* separato dal resto dell'applicazione.

Classi estese

- android.os.AsyncTask

Estesa da

- Samba.utils.sending.ArrListSender (A.4.45)
- Samba.utils.sending.ReconstructionSender (A.4.46)

A.4.45 Samba.utils.sending.ArrListSender

Descrizione

Classe in grado di inviare liste di `SambaPoint`.

Utilizzo

È usata per inviare liste di `SambaPoint`.

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.utils.encoding.EncodeStrategy`: relazione uscente, composizione. (A.4.40)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, composizione. (A.4.19)

Classi estese

- Samba.utils.sending.AbsSender (A.4.44)

A.4.46 Samba.utils.sending.ReconstructionSender

Descrizione

Classe in grado di inviare ricostruzioni 3D.

Utilizzo

È usata per inviare ricostruzioni 3D.

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.utils.encoding.EncodeStrategy`: relazione uscente, composizione. (A.4.40)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, composizione. (A.4.19)
- `Samba.clouds.reconstruction.SerializzablePCloud3DReconstruction`: relazione uscente, composizione. (A.4.26)

Classi estese

- Samba.utils.sending.AbsSender (A.4.44)

A.4.47 Samba.utils.sending.InternalStorageSender

Descrizione

Classe in grado di salvare su disco *Point Cloud* serializzati.

Utilizzo

È usata per salvare su disco *Point Cloud* serializzati.

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza. (A.4.36)
- `Samba.utils.encoding.EncodeStrategy`: relazione uscente, composizione. (A.4.40)
- `Samba.sharedState.ReconstructionCache`: relazione entrante, composizione. (A.4.19)

A.4.48 Samba.utils.services

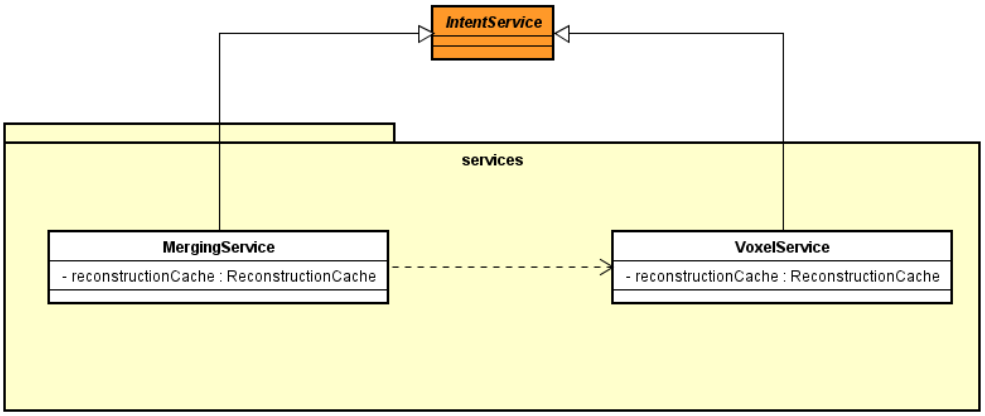

figura A.13: Componente Samba.utils.services

Descrizione

Questo *package* contiene i servizi asincroni che servono ad effettuare operazioni sui dati raccolti dai sensori *Tango*.

Classi

- Samba.utils.services.MergingService (A.4.49)
- Samba.utils.services.VoxelService (A.4.50)

A.4.49 Samba.utils.services.MergingService

Descrizione

Servizio in grado di sovrapporre ad una ricostruzione data un *Point Cloud* a patto che assieme a quest'ultimo venga fornita anche la sua matrice di trasformazione.

Utilizzo

È usato per ricostruire in *background* l'oggetto inquadrato dall'utente.

Relazioni con altre classi

- `Samba.clouds.utils.SambaRawData`: relazione uscente, dipendenza, utilizzo interno ad un metodo. (A.4.37)
- `Samba.utils.services.VoxelService`: relazione uscente, dipendenza. (A.4.50)
- `Samba.sharedState.ReconstructionCache` relazione uscente, composizione. (A.4.19)

Classi estese

- android.app.IntentService

A.4.50 Samba.utils.services.VoxelService

Descrizione

Servizio in grado di ottimizzare una ricostruzione 3D eliminando i punti ridondanti.

Utilizzo

È usato in *background* ottimizzare ed eliminare i punti ridondanti da una ricostruzione 3D.

Relazioni con altre classi

- `Samba.clouds.utils.SambaPoint`: relazione uscente, dipendenza, utilizzo interno ad un metodo. (A.4.36)
- `Samba.utils.services.MergingService`: relazione entrante, dipendenza. (A.4.49)
- `Samba.sharedState.ReconstructionCache` relazione uscente, composizione. (A.4.19)

Classi estese

- android.app.IntentService

Appendice B

Glossario

API

Con *API* o *Application Programming Interface* si intende ogni insieme di metodi e procedure resi disponibili al programmatore. Di solito raggruppati in a formare un set di strumenti specifici per l'espletamento di un determinato compito.

Artefatto

Gli *artefatti* sono degli elementi presenti all'interno di una ricostruzione *Point Cloud* ma non nella realtà, dovuti errori nella rilevazione. Sono generalmente di forma planare e sospesi qualche centimetro al di sopra del pavimento.

Design Pattern

Un *Design Pattern* è una soluzione progettuale ad un problema ricorrente in un certo contesto.

Ghosting

Il problema del *Ghosting* affligge alcune registrazioni effettuate con il prodotto: è dovuto ad una errata stima della posizione del dispositivo e produce ricostruzioni tridimensionali affette da errori. Se visualizzate graficamente il tali ricostruzioni presentano l'oggetto come se fosse sdoppiato, ad esempio come in figura 7.1.

Feature

Una *feature* è un punto particolare nello spazio che il processo di *Area Learning* ritiene facilmente riconoscibile, e che quindi è salvato ed usato come punto di riferimento in un determinato ambiente. Un'area ha generalmente qualche centinaio di *feature*.

Fotocamera Fish-eye

Una Fotocamera *Fish-eye* è un obbiettivo grandangolare che abbraccia un angolo di campo di circa 180 gradi, in particolare quello in dotazione nel dispositivo usato è in bianco e nero.

Mesh

Una *mesh* o *mesh* poligonale è una collezione di vertici, spigoli e facce che definiscono la forma di un oggetto tridimensionale.

Milestone

Letteralmente pietre miliari, vengono tipicamente utilizzate nella pianificazione e gestione di progetti complessi al fine di indicare importanti traguardi intermedi nello svolgimento del progetto stesso. Molto spesso sono rappresentate da eventi, cioè da attività con durata zero o di un giorno, e vengono evidenziate in maniera diversa dalle altre attività nell'ambito dei documenti di progetto.

Motion Tracking

Il *Motion Tracking* è una tecnica che permette al dispositivo di tracciare i suoi movimenti all'interno di un area.

Quaternione

Un quaternione è un oggetto matematico del tipo

$$a + bi + cj + dk$$

dove $a, b, c, d \in \mathbb{R}$ e i, j e k sono simboli che si comportano come l'unità immaginaria dell'insieme dei numeri complessi $\mathbb{C}$.
Una loro possibile applicazione è la modellazione di rotazioni nello spazio, per questo sono molto usati in fisica teorica, ma anche in *Computer* grafica e in robotica.

Point Cloud

Un *Point Cloud* è modello matematico che descrive un oggetto tridimensionale semplicemente come insieme del maggior numero possibile di punti dell'oggetto stesso.
È molto usato in ambito di *Computer vision* e realtà aumentata.

Race Condition

Una *Race Condition* è un fenomeno proprio dei sistemi concorrenti ed avviene quando il risultato finale dell'elaborazione dipende dalla temporizzazione o dalla sequenza con cui vengono eseguiti i processi.

Stakeholder

Gli *Stakeholder*, o portatori di interesse, sono l'insieme delle persone che a vario titolo sono coinvolte nel ciclo di vita del *Software* con influenza sul prodotto.

Tango Project

Il progetto *Tango* è un progetto promosso e portato avanti da *Google*. Si propone di promuovere nuovi tipi di dispositivi dotati di *Hardware* innovativo, come fotocamera *Fish-Eye*, sensore di profondità etc. Grazie ai loro sensori i dispositivi *Tango* sono in grado di avere una certa conoscenza dell'ambiente che li circonda: sono in grado di tracciare il proprio movimento ed orientazione, di ricordare gli ambienti dove sono già stati, di avere una visione tridimensionale degli oggetti.

UML - Unified Modelling Language

Famiglia di notazioni grafiche che si basano su un singolo meta-modello e servono a supportare la descrizione e il progetto dei sistemi software.

Voxel

Un *voxel*, detto anche *pixel* volumetrico, è un elemento di volume che rappresenta un valore di intensità di segnale o di colore in uno spazio tridimensionale.

Bibliografia

[1] http://www.vicworldwide.com

[2] https://get.google.com/tango/developers

[3] https://get.google.com/tango/apps

[4] pointclouds.org

[5] Cha Zhang and Tsuhan Chen, *EFFICIENT FEATURE EXTRACTION FOR 2D/3D OBJECTS IN MESH REPRESENTATION*, Dept. of Electrical and Computer Engineering, Carnegie Mellon University 5000 Forbes Avenue, Pittsburgh, PA 15213, USA.

[6] https://developer.android.com/index.html

[7] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Professor Gilberto Filè, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

In secondo luogo, ringrazio Michele Marchetto, tutor aziendale, così come tutti i colleghi per aver reso lo stage un'esperienza di crescita.

Desidero ringraziare con affetto i miei genitori, Giovanna ed Azelio, e mio fratello Giordano per aver creduto in me in ogni momento.

Ringrazio Francesca per la pazienza portata durante la stesura di questo documento.

Ringrazio i miei compagni di corso per le notti insonni durante i progetti universitari.

Ringrazio i miei amici di sempre, per avermi reso quello che sono.

Padova, Sept 2016 Tommaso Padovan