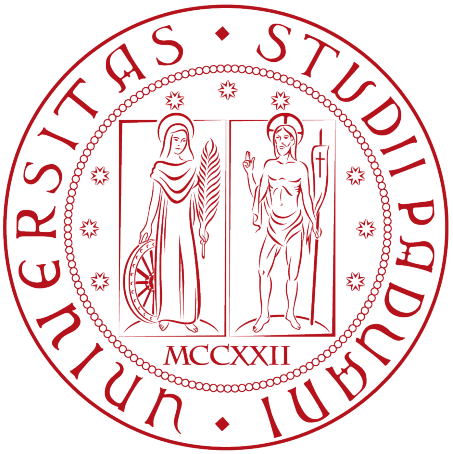
Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica



Project Tango e PCL: un'applicazione pratica alle 'Nuvole di Punti'

Tesi di laurea

Relatore

Prof. Ombretta Gaggi

Laureando

Davide Bortot

1070213

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 Davide Bortot presso l'azienda VIC S.r.l. Obbiettivo del tirocinio era lo sviluppo di un'applicazione prototipale per il *tablet Project Tango* per l'aquisizione di scansioni di oggetti reali sotto forma di 'Nuvola di Punti'.
In primo luogo era richiesto il filtraggio e l'elaborazione dei dati acquisiti in modo da isolare l'oggetto scansionato, e successivamente di creare una *mesh* tridimensionale a partire dai punti rimanenti. Scopo finale era calcolare il volume dell'oggetto tridimensionale così ottenuto.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof.ssa Ombretta Gaggi, 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 Tommaso Padovan, Lorenzo Ceccon e Luca D'Ambros per aver reso lo stage un'esperienza di crescita.

Ringrazio col cuore i miei genitori Giulia e Carlo per aver creduto in me in ogni momento e avermi sempre sostenuto.

Infine ringrazio i miei amici di sempre e gli amici nuovi che ho incontrato nel mio percorso universitario, per avermi reso quello che sono.

Padova, Oct 2016 Davide Bortot

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 alle 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 viene fondata a Venezia 7 anni fa come piccola società di ispezione locale. Fin dall'inizio, l'obiettivo principale di VIC è stato la riduzione del tempo tra ispezione e reporting al cliente. Ora l'obiettivo è raggiunto, perchè 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

I più importanti obbiettivi del controllo qualità effettuato durante un ispezione sono il determinare la corretta forma, peso, quantità e dimensioni degli oggetti da esaminare. Gli ispettori possono scattare 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 dati utili quali:

* Una stima del volume, e se necessario del peso, dell'oggetto.
* L'esito del confronto dell'oggetto con un modello ideale, per evidenziare eventuali 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.
Le operazioni troppo computazionalmente costose da effettuare su *tablet*, quali il filtraggio e *meshing* dei punti acquisiti, sono state delegate ad un *backend server* che effettui le elaborazioni necessarie ed invii i risultati al richiedente, mentre all'applicativo per

tablet è stato affidato il compito di aquisire e visualizzare un oggetto sotto forma di 'Nuvola di Punti' e poterne esaminare la *mesh* ottenuta.

1.3 Cos'è Project Tango

Project Tango è un *tablet* sperimentale prodotto da *Google* in grado di scandire tridimensionalmente l'ambiente circostante e di tracciare la propria posizione rispetto ad esso. Ciò è possibile attraverso il ricco hardware di cui è dotato (si veda la figura 1.1), tra cui:

* una fotocamera RBG-IR
* una *fotocamera Fisheye*
* un sensore di profondità IR
* accelerometro e giroscopio

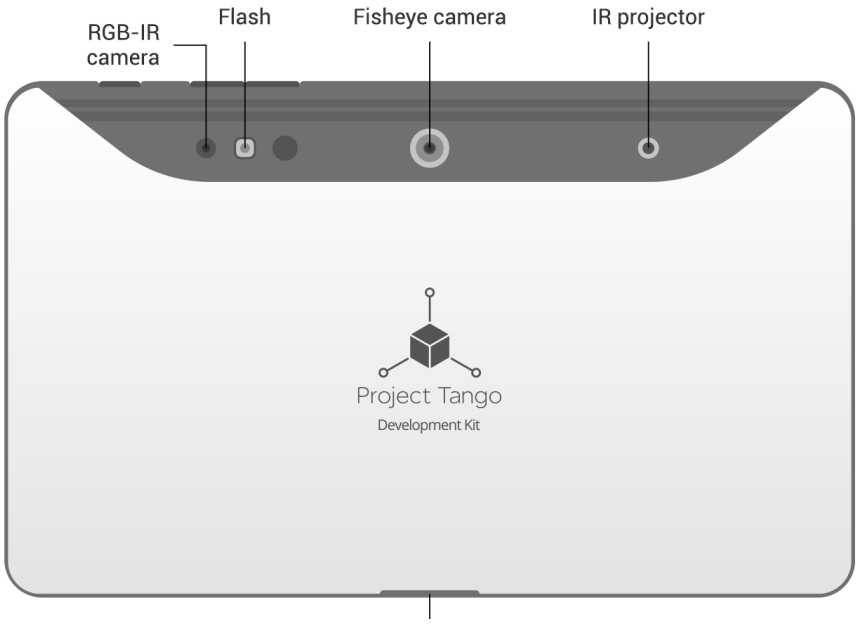


figura 1.1: L' *hardware* di *Project Tango*

Project Tango è quindi specificamente pensato per sviluppare applicazioni che necessitano di comprendere ed estrapolare informazioni dal mondo reale (ad es. realtà aumentata). Le funzionalità del *tablet* sono accessibili attraverso le *Tango API*, le *API* ufficiali per lo sviluppo di applicazioni *Tango*.

1.4 Lavoro svolto

Al tirocinante è stato richiesto di focalizzarsi sullo sviluppo del lato *server* dell'applicazione, in modo che fosse possibile ricevere ed elaborare 'Nuvole di Punti', ed ottenere i risultati elaborati, lavorando nel mentre in stretta collaborazione con un altro stagista, Tommaso Padovan (relatore il Prof. Gilberto Filè) che si occupava dell'applicazione lato *client*. Nell'ultimo periodo di stage la parte *server* era già a buon punto, quindi il focus del tirocinio si è spostato sul miglioramento dell'applicativo lato *client*.

1.5 Il Prodotto - lato client

L'applicazione su *tablet* prodotta realizza, seppur non ancora in maniera completa, le esigenze richieste.
La sua realizzazione ha incontrato molte problematiche talvolta critiche e difficili da prevedere. Per questo, durante lo sviluppo, sono stati implementati più prototipi, al fine di esplorare le potenzialità e soprattutto i limiti del *tablet* e delle *Tango API*.
Lo scopo principale dell'applicazione lato *tablet* è quello di rilevare una corretta 'Nuvola di Punti' dell'oggetto che si vuole esaminare.
Una 'Nuvola di Punti' è una descrizione matematica di un oggetto tridimensionale ottenuta tramite un insieme, il più possibile fitto, di punti che lo compongono, definiti dalle loro coordinate (**x**,**y**,**z**) rispetto ad un fissato sistema di riferimento.
Tale rappresentazione, riferita spesso d'ora in poi con il più elegante termine inglese *Point Cloud*, è facilmente comprensibile all'utente se visualizzata come in figura 1.2.

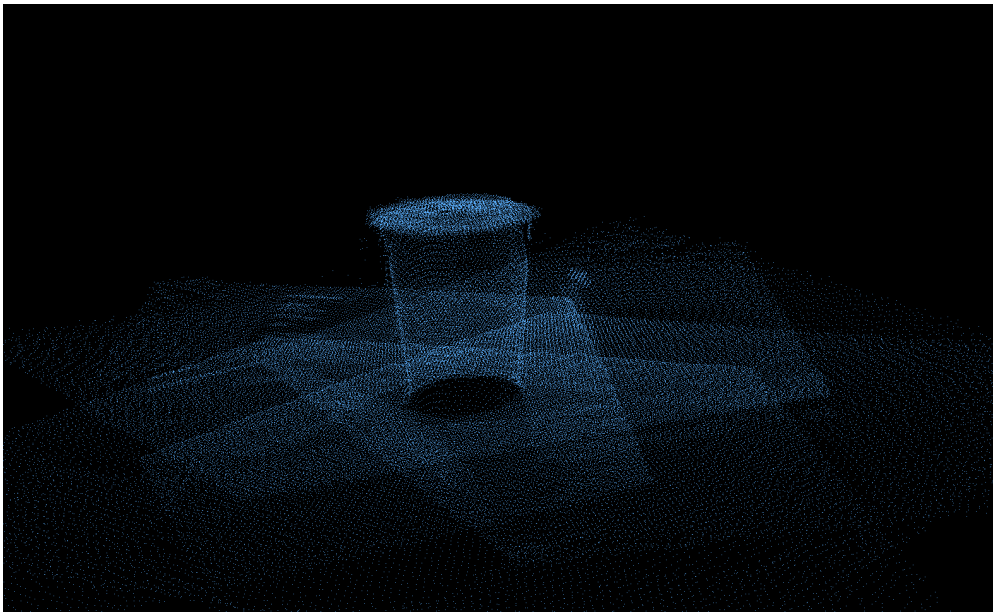


figura 1.2: Esempio di *Point Cloud*: un cestino cilindrico

1.5.1 Il prototipo ad inizio stage: Samba

In questa sezione viene presentato il prototipo nel suo stato all'inizio dello stage, per capirne le funzionalità e limiti e introdurre il lettore alle problematiche che un tale progetto sperimentale ha comportato. Sulla base dei risultati di *Samba* ho iniziato il mio lavoro sulla parte di elaborazione lato *server* dei *Point Cloud* acquisiti; questo stadio prototipale è stato inoltre un occasione di collaborazione con l'altro stagista e il punto di inizio per lo sviluppo del prototipo successivo. *Samba* affronta e risolve alcuni problemi dei prototipi precedenti, come il *"drifting"* e alcuni problemi prestazionali dovuti alla mole elevata di dati acquisiti.

Drift correction

Il prototipo introduce funzionalità che tentano di arginare il problema del *"drifting"*, presente in stadi prototipali precedenti. Il *drifting* è un problema comune nelle applicazioni di realtà aumentata, che come *Project Tango* usano la tecnica del *Motion Tracking*, cioè aggiornano costantemente la propria posizione relativamente alle coordinate acquisite nella posizione precedente, mantenendo così una storia dei movimenti del *device* rispetto all'ambiente circostante.
Ad ogni aggiornamento della posizione è normale ed inevitabile che la misurazione, per quanto precisa, introduca un piccolo errore; la catena di errori sommati porta quindi col passare del tempo ad un'importante discrepanza tra la posizione stimata del *device* e la sua posizione reale, come evidenziato in figura 1.3

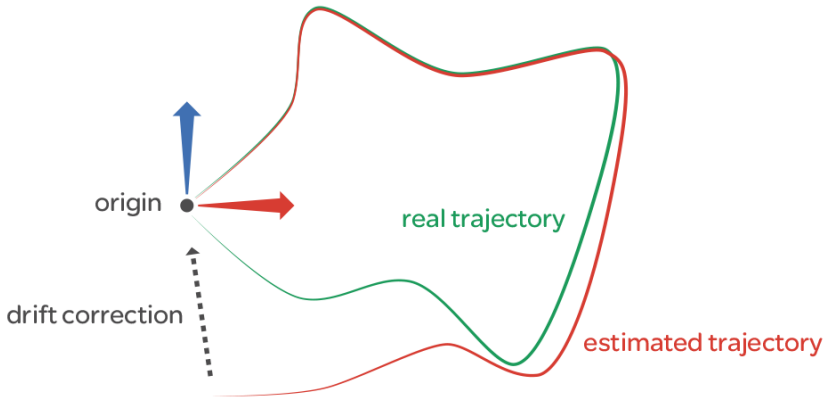
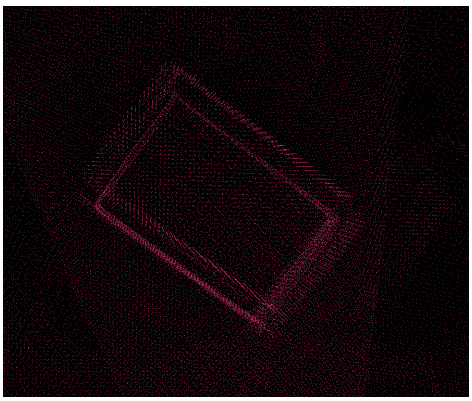


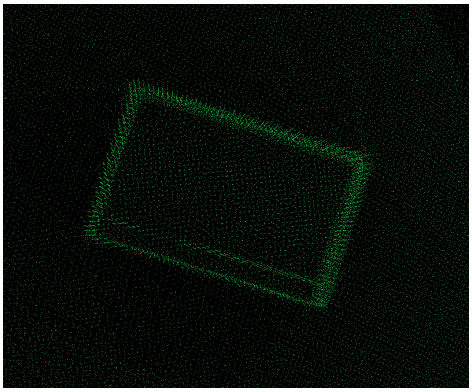
figura 1.3: *Drifting* nel *Motion Tracking*

Per ovviare in parte a questo problema *Samba* utilizza una tecnica chiamata *Drift Correction* ch si appoggia sulle funzionalità di *Area Learning* dei *device Tango*.
L' *Area Learning* consiste nella capacita del dispositivo di estrarre dallo spazio fisico che sta analizzando una serie di punti significativi (o *key features*), facilmente riconoscibili, e di salvare tali informazioni per confrontarle con le successive acquisizioni. In questo modo il dispositivo è capace di riconoscere un'area precedentemente visitata, e può quindi applicare le necessarie correzioni alla popria stima della traiettoria, di qui il nome *Drift Correction*.
Con l'implementazione di questa funzionalità è necessario, prima di iniziare la ricostruzione di un *Point Cloud*, riprendere l'oggetto e i suoi dintorni per un po' di tempo e

da diverse angolazioni, in modo da permettere al *tablet Tango* di costruire una mappa, detta *Area description*, dell'area circostante, e di stabilizzare la traiettoria stimata con le informazioni acquisite.
I risultati si vedono confrontando queste due ricostruzioni di una scatola, vista dall'alto (fig. 1.4a e fig. 1.4b), rispettivamente senza e con *Drift correction*.



(a) Ricostruzione senza *Drift correction*



(b) Ricostruzione con *Drift correction*

figura 1.4: Benefici della *Drift Correction*

Come si nota in figura 1.4b il risultato non è ancora ottimale, infatti un lato della scatola appare sdoppiato e spostato di qualche centimetro; si tratta di un problema di *ghosting* di cui verrà trattato più avanti.

Prestazioni

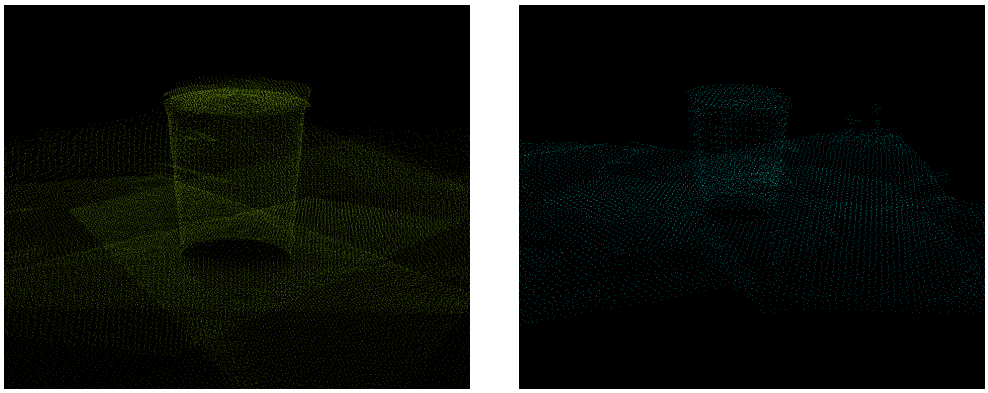
Il prototipo precedente presentava due problemi prestazionali principali:

* La ricostruzione passo per passo del *Point Cloud* finale era troppo lenta
* La mole dei dati trattati al sovrapporsi di più *Point Cloud* era proibitiva

Il primo problema si presentava utilizzando i metodi forniti dalla libreria *Tango* per trasformare le coordinate relative dei punti in coordinate assolute per permetterne la giusta sovrapposizione. Tale metodo, seppur di facile utilizzo, diventava troppo dispendioso all'aumentare delle dimensioni del *Point Cloud*, che raggiunge facilmente gli 80.000 punti.
Per risolvere il problema viene quindi creata, per ogni *Point Cloud*, una *matrice di rototraslazione* che rappresenta lo spostamento e la rotazione del dispositivo rispetto al sistema di riferimento, e viene moltiplicato il vettore delle coordinate di ogni singolo punto per la matrice generata.
In questo modo si sono ridotti i tempi di elaborazione dell'80%;

Il secondo problema era causato dal sovrapporsi di molti punti quasi identici, ad esempio i punti rappresentanti il pavimento. Partendo dall'osservazione che una certa quantità di punti molto vicini può essere trasformata in un unico punto, valore medio di tutti gli altri, senza una significativa perdita d'informazione, *Samba* risolve il problema attraverso una tecnica di *voxeling*.
Lo spazio viene suddiviso in tanti piccoli parallelepipedi (tipicamente cubi) di uguali

dimensioni; tutti i punti che ricadono all'interno di un singolo parallelepipedo, o *voxel*, vengono considerati come un unico punto. Così facendo si riducono sensibilmente le dimensioni del *Point Cloud* senza alterarne negativamente la precisione. Di seguito la differenza visivamente evidente tra lo stesso Point Cloud filtrato prima con voxel cubici di lato 1cm (fig. 1.5a) e poi di lato 3cm (fig. 1.5b), con una differenza di circa 60.000 punti rimossi in più nella seconda elaborazione.



(a) Voxeling leggero **(b)** Voxeling elevato

figura 1.5: Effetti del *voxeling*

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 alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Processi di sviluppo

Una buona definizione dei processi produttivi è fondamentale per il successo del progetto

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.

Capitolo 3

Studio di fattibilità ed Analisi dei rischi

Il progetto si è subito presentato come sperimentale, impegnativo e facilmente soggetto a fallimento. Per questo si è reso necessario studiarne attentamente la fattibilità e valutarne i rischi

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. Seguendo quest'approccio è stato possibile valutare le potenzialità e i limiti del *device Tango* e delle API annesse, e considerare l'applicabilità degli algoritmi general purpose della libreria Point Cloud Library (PCL) al caso particolare del progetto in questione.

3.2 Studio di fattibilità

Prima di iniziare il progetto è stato effettuato un accurato studio di fattibilità basato sullo studio della libreria *PCL* e della vasta quantità di esempi d'utilizzo della stessa reperibili online. Si sono cercate poi soluzioni per il calcolo del volume di una mesh. Per il lato *client* si sono esplorate soluzioni per la visualizzazione di mesh salvate in formato OBJ e per la comunicazione *client-server*.

3.2.1 Elaborazione dei Point Cloud

É parso subito evidente che l'applicativo necessita di un formato standard, trasferibile via Web, per il trattamento di Point Cloud, e delle funzionalità di I/O associate. Inoltre è stato necessario esplorare le funzionalità disponibli per il filtraggio ed il meshing dei Point Cloud acquisiti. Per ultimo si è cercato un affidabile metodo matamatico per il calcolo del volume di una mesh. A tal riguardo è stata studiata estensivamente la documentazione *PCL*, che offre anche numerosi esempi d'utilizzo:

* **The PCD (Point Cloud Data) file format** [13]
* **Reading Point Cloud data from PCD files** [11]

* **Writing Point Cloud data to PCD files** [14]
* **Filtering a PointCloud using a PassThrough filter** [7]
* **Downsampling a PointCloud using a VoxelGrid filter** [4]
* **Removing outliers using a StatisticalOutlierRemoval filter** [12]
* **Plane model segmentation**[10]
* **Euclidean Cluster Extraction**[5]
* **Fast triangulation of unordered point clouds**[6]
* **Fitting trimmed B-splines to unordered point clouds**[8]

Per il calcolo del volume invece, punto focale del progetto, si è studiato quanto descritto in [15]

3.2.2 Visualizzazione di file in formato OBJ

Per visualizzare le mesh prodotte dal *server* su *tablet* è stato necessario ricercare esempi di applicativi che permettessero di trattare oggetti tridimensionali in formato OBJ. A tal riguardo si sono studiate due applicazioni Android:

* **3D viewer**[2]
* **3D Model Viewer Open Source**[1]

In particolare la seconda applicazione essendo open-source (realizzata da Andres Oviedo, il codice è liberamente reperibile dalla repository[1] GitHub) si è rivelata una valida e disponibile implementazione della funzionalità richiesta.

3.2.3 Comunicazione client-server

Per implementare la comunicazione *client-server* la scelta più logica è stata di esplorare la documentazione ufficiale Android, che mette a disposizione il package **Apache HTTP**[3], che è stato però deprecato dalla versione 6.0 di Android, quindi si è studiato come alternativa l'ottima e leggera libreria open source **OkHttp**[9] che risponde alle necessità richieste.

3.2.4 Conclusioni

Alla luce della disponibilità e qualità degli esempi sopracitati il progetto è apparso fattibile. La documentazione e le applicazioni studiate infatti hanno dato prova delle potenzialità della libreria *PCL* e della possibilità concreta di calcolare il volume esatto di una mesh tridimensionale. É quindi stato possibile dare il via al ciclo di vita del progetto software.

[1] **https://github.com/andresoviedo/android-3D-model-viewer**.

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 a elaborare delle possibili soluzioni per far fronte a tali rischi.

3.3.1 Rischi generali

1. Limiti fisici del tablet
Probabilità: Alta.
Impatto: Medio.
Descrizione: Il dispositivo è dotato di sensori IR che sfruttano la riflessione della luce per determinare la distanza dei punti che è in grado di individuare, e di fotocamere RGB. Superfici lucide/riflettenti o tendenti al nero possono compromettere gravemente la qualità della misurazione, così come gli ambienti con illuminazione scarsa o troppo intensa.
Gestione: Attento studio delle potenzialità dell'hardware del *tablet*[2] e del loro utilizzo[3] nelle Tango API.

2. Incomprensibilità dello stato del progetto
Probabilità: Media.
Impatto: Medio.
Descrizione: Venendo inserito in un progetto già avviato, lo studente necessita di comprendere a fondo lo stato dell'arte dell'applicativo.
Gestione: Lo studente deve, consultando la documentazione creata e collaborando strettamente con gli altri elementi del team e con il tutor aziendale, assicurasi di comprendere quanto è stato fatto, come è stato fatto e le necessità future del progetto..

3. Scarsa competenza nello sviluppo Android
Probabilità: Alta.
Impatto: Basso.
Descrizione: Lo studente ha scarse conoscenze dello sviluppo Android, ma conosce il linguaggio Java.
Gestione: Lo studente deve effettuare un adeguato periodo di studio ed assumere familiarità con la documentazione ufficiale.

3.3.2 Rischi specifici

4. Inadeguatezza della libreria *PCL* e relativa documentazione
Probabilità: Bassa.
Impatto: Basso.
Descrizione: La libreria *PCL* espone molti algoritmi general-purpose per la manipolazione di Point Cloud, che non necessariamente si adattano alle necessità specifiche del progetto in questione.
Gestione: La libreria espone anche molte funzionalità basilari che è possibile utilizzare e combinare per ovviare alle necessità che algoritmi specifici non sempre soddisfano.

[2] **https://developers.google.com/tango/hardware/tablet**.
[3] **https://developers.google.com/tango/overview/depth-perception**.

5. Impredicibilità dei Point Cloud
Probabilità: Alta.
Impatto: Medio.
Descrizione: La qualità di un oggetto ricostruito come Point Cloud è impredicibile, a causa dei fenomeni di *drifting*, *ghosting* ed artefatti di cui soffre la ricostruzione. Inoltre la tipologia di oggetti scansionati è altamente variabile.
Gestione: Il processo di filtraggio dev'essere attentamente impostato in modo da trattare al meglio tali imperfezioni e tipologie di oggetti. Bisogna inevitabilmente prendere in considerazione che non sempre il filtraggio può ottenere i risultati voluti, e richiedere quindi una nuova ricostruzione all'utente.

6. Difficoltà nella comunicazione client-server
Probabilità: Media.
Impatto: Basso.
Descrizione: L'applicazione verrà usata in futuro sul campo da addetti aziendali, che potrebbero non disporre di una connessione stabile con il *server*.
Gestione: Bisogna prevedere e controllare una possibile assenza di connessione e regolarsi di conseguenza, fornendo ad esempio all'utente la possibilità di salvare il proprio lavoro e ritentare la comunicazione col *server* in un secondo momento. Dato lo stadio prototipale del progetto e l'utilizzo indoor dell'applicativo, il rischio è altamente improbabile.

7. Difficoltà nel calcolo del volume
Probabilità: Alta.
Impatto: Media.
Descrizione: Il calcolo del volume dipende fortemente dal risultato dell'elaborazione attraverso le funzionalità della *PCL*, in particolare dall'assenza di punti estranei all'oggetto prima di effettuarne il meshing, al quale segue il calcolo..
Gestione: Lo studente deve eseguire una *suite* di test adeguati per valutare la bontà del calcolo del volume e la percentuale d'errore ammissibile, e capire le cause che portano il calcolo a divergere dal volume reale.

Capitolo 4

Analisi dei requisiti

4.1 Casi d'uso - lato client

Per lo studio dei casi di utilizzo del prodotto sono stati creati 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. Vediamo di seguito i casi d'uso che descrivono le funzionalità da realizzare nell'applicativo lato *client*, come da figura 4.1.

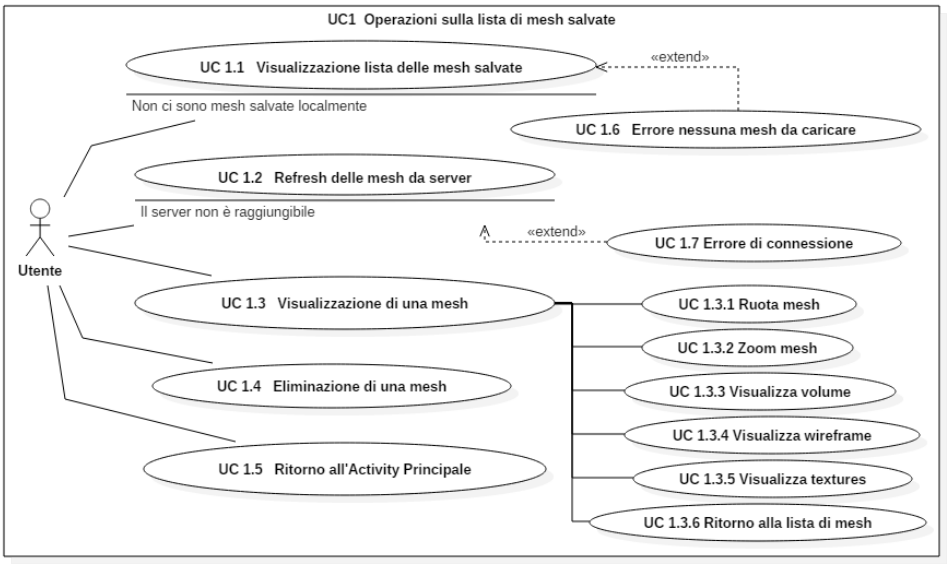


figura 4.1: Use Case - UC1: Operazioni sulla lista di mesh salvate

UC1: Operazioni sulla lista di mesh salvate

Attori Principali: User.

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

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

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

4.1.1 UC1.1: Visualizzazione 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 localmente.

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

Postcondizioni: Viene visualizzata la lista delle mesh salvate localmente.

Extension Points:

* **Non ci sono mesh salvate localmente**: Non c'è nessuna mesh salvata da caricare in lista, viene visualizzato un messaggio informativo (UC 1.6).

4.1.2 UC1.2: Refresh delle mesh da 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. L'utente viene informato dell'avvenuto refreshing.

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

Extension Points:

* **Il server non è raggiungibile**: c'è stato un errore nel tentativo di connettersi al *server*. Viene visualizzato un messaggio d'errore (UC 1.7).

4.1.3 UC1.3: Visualizzazione di una 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 operazioni su di essa.

Postcondizioni: Nessuna.

UC1.3.1: Ruota mesh

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e vuole ruotarla.

Descrizione: L'utente, scorrendo lo schermo con un dito, ruota l'oggetto visualizzato.

Postcondizioni: La mesh è stata ruotata.

UC1.3.2: Zoom mesh

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e vuole ingrandirla/rimpicciolirla.

Descrizione: L'utente, usando due dita, ingrandisce/rimpicciolisce l'oggetto visualizzato.

Postcondizioni: La mesh è stata ingrandita/rimpicciolita.

UC1.3.3: Visualizza volume

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e vuole conscerne il volume calcolato.

Descrizione: L'utente, cliccando un apposito bottone in alto a destra, visualizza un popup con il volume calcolato della mesh.

Postcondizioni: L'utente ha visualizzato il volume della mesh.

UC1.3.4: Visualizza wireframe

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e vuole osservarne solamente il *wireframe*, cioè lo scheletro di triangoli di cui è composta.

Descrizione: L'utente, cliccando un apposito bottone in alto a destra, visualizza

solamente il *wireframe* dell'oggetto.

Postcondizioni: Viene visualizzato il *wireframe* dell'oggetto.

UC1.3.5: Visualizza textures

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e vuole applicarvi delle textures di default.

Descrizione: L'utente, cliccando un apposito bottone in alto a destra, visualizza la mesh alla quale vengono applicate delle semplici textures di default.

Postcondizioni: Viene visualizzata la mesh dell'oggetto, alla quale sono state applicate textures di default.

UC1.3.6: Ritorno alla lista di mesh

Attori Principali: Utente.

Precondizioni: L'utente sta visualizzando una mesh e desidera ritornare alla lista di mesh.

Descrizione: L'utente preme sul tasto "Back" e ritorna alla lista di mesh.

Postcondizioni: L'utente è uscito dalla visualizzazione di una mesh ed è ritornato alla lista delle mesh.

4.1.4 UC1.4: Eliminazione di una mesh

Attori Principali: Utente.

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

Descrizione: L'utente preme sul pulsante "Delete" affianco al nome di una specifica mesh per cancellarla dai *File* salvati localmente.

Postcondizioni: Il *File* selezionato è stato cancellato e non è più presente sulla memoria del dispositivo.

4.1.5 UC1.5: 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.

4.1.6 UC1.6: Errore nessuna mesh da caricare

Attori Principali: Utente.

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

Descrizione: L'utente viene informato dell'assenza di mesh salvate localmente.

Postcondizioni: La lista di mesh non è stata popolata.

4.1.7 UC1.7: Errore di connessione

Attori Principali: Utente.

Precondizioni: L'utente ha selezionato il pulsante di refresh per aggiornare la lista di *mesh* aggiungendo le altre presenti sul *Server*.

Descrizione: L'utente viene informato dell'assenza di connessione con il *Server*.

Postcondizioni: La comunicazione tra dispositivo e *Server* non è possibile.

4.2 Casi d'uso - lato server

Verranno di seguito descritti i casi d'utilizzo dell'applicativo lato *server* delle sue interazioni con il *client*, di cui possiamo avere una visione generale in figura 4.2.

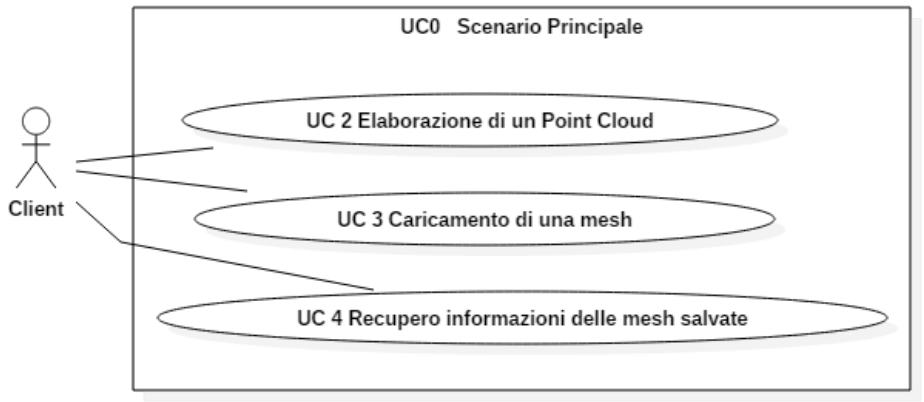


figura 4.2: Use Case - UC0: Scenario Principale

UC0: Scenario Principale

Attori Principali: Client.

Precondizioni: Il client invia una specifica richiesta al *server*.

Descrizione: Il *server* elabora la richiesta del client e ne restituisce i risultati.

Postcondizioni: Il *server* ha elaborato la richiesta ed inviato il responso al *client*.

4.2.1 UC2 Elaborazione di un Point Cloud

La figura 4.3 descrive i casi d'uso per l'elaborazione di un Point Cloud.

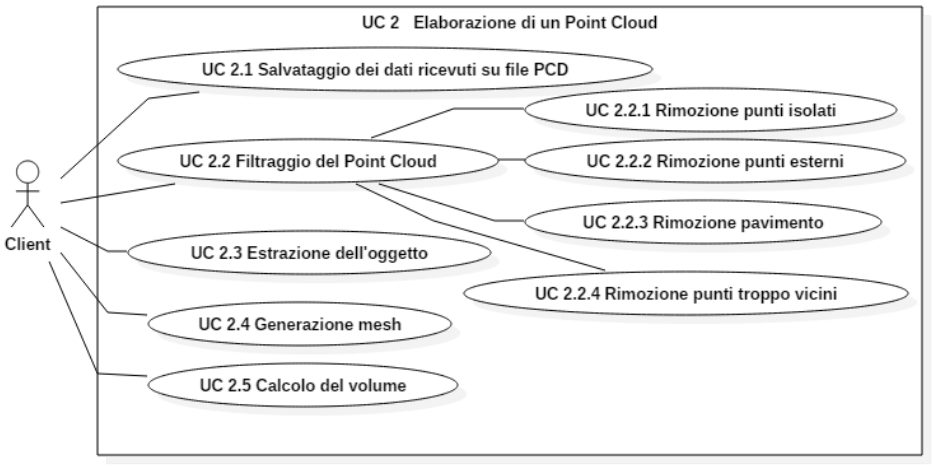


figura 4.3: Use Case - UC2: Elaborazione di un Point Cloud

UC2: Elaborazione di un Point Cloud

Attori Principali: Client.

Precondizioni: Il *client* richiede l'elaborazione di un Point Cloud al *server*.

Descrizione: Il *server* elabora il Point Cloud ricevuto e notifica l'esito al *client*.

Postcondizioni: Il *client* riceve l'esito dell'elaborazione dal *server*.

4.2.2 UC2.1 Salvataggio dei dati ricevuti su file PCD

Attori Principali: Client.

Precondizioni: Il Client invia tramite protocollo HTTP dati in formato PCD.

Descrizione: I dati ricevuti vengono salvati sul *server* in un file in formato PCD.

Postcondizioni: I dati sono stati correttamente salvati su file.

4.2.3 UC2.2 Filtraggio del Point Cloud

Attori Principali: Client.

Precondizioni: Il *client* ha richiesto l'elaborazione di un Point Cloud.

Descrizione: Il Point Cloud viene caricato da file e filtrato.

Postcondizioni: Il Point Cloud è stato filtrato.

UC2.2.1 Rimozione punti isolati

Attori Principali: Client.

Precondizioni: Viene richiesto di eliminare i punti isolati della nuvola.

Descrizione: Il Point Cloud viene filtrato eliminando i punti isolati.

Postcondizioni: I punti isolati sono stati rimossi dal Point Cloud.

UC2.2.2 Rimozione punti esterni

Attori Principali: Client.

Precondizioni: Viene richiesto di eliminare i punti esterni della nuvola.

Descrizione: Il Point Cloud viene filtrato eliminando i punti più esterni, che non appartengono all'oggetto scansionato.

Postcondizioni: I punti esterni sono stati rimossi dal Point Cloud.

UC2.2.3 Rimozione pavimento

Attori Principali: Client.

Precondizioni: Viene richiesto di eliminare i punti del pavimento dalla nuvola.

Descrizione: Il Point Cloud viene filtrato eliminando i punti planari che corrispondono al pavimento.

Postcondizioni: I punti del pavimento sono stati rimossi dal Point Cloud.

UC2.2.4 Rimozione punti troppo vicini

Attori Principali: Client.

Precondizioni: Viene richiesto di eliminare i punti ravvicinati della nuvola.

Descrizione: Il Point Cloud viene filtrato eliminando i punti eccessivamente ravvicinati, che possono essere approssimati in un unico punto.

Postcondizioni: I punti ravvicinati sono stati rimossi dal Point Cloud.

4.2.4 UC2.3 Estrazione dell'oggetto

Attori Principali: Client.

Precondizioni: Viene richiesto di estrarre dalla nuvola i soli punti appartenenti all'oggetto scansionato.

Descrizione: La parte di Point Cloud che rappresenta l'oggetto scansionato viene isolata dal resto dei punti.

Postcondizioni: I punti del solo oggetto scansionato sono stati estratti dal Point Cloud.

4.2.5 UC2.4 Generazione mesh

Attori Principali: Client.

Precondizioni: Il *client* richiede di generare una mesh dell'oggetto ispezionato.

Descrizione: Viene effettuato il meshing del Point Cloud del solo oggetto scansionato.

Postcondizioni: É stata generata correttamente una mesh dell'oggetto ispezionato.

4.2.6 UC2.5 Calcolo del volume

Attori Principali: Client.

Precondizioni: il *client* richiede di calcolare il volume dell'oggetto scansionato.

Descrizione: Viene calcolato il volume della mesh generata a partire dal Point Cloud.

Postcondizioni: Il volume dell'oggetto è stato calcolato.

4.2.7 UC3 Caricamento di una mesh

In figura 4.4 sono riportati i casi d'uso per il caricamento di una mesh.

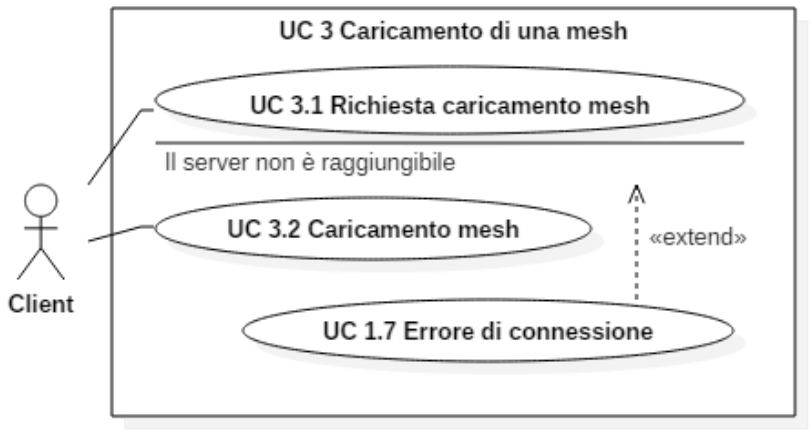


figura 4.4: Use Case - UC3: Caricamento di una mesh

UC3: Caricamento di una mesh

Attori Principali: Client.

Precondizioni: Il *client* richiede al *server* l'invio di una specifica mesh.

Descrizione: Il *server* elabora la richiesta ricevuta ed invia la mesh desiderata al *client*.

Postcondizioni: Il *client* riceve la mesh richiesta dal *server*.

4.2.8 UC3.1 Richiesta caricamento Mesh

Attori Principali: Client.

Precondizioni: Il *client* richiede il caricamento di una mesh inviando tramite protocollo HTTP il nome della mesh richiesta.

Descrizione: Il *server* elabora la richiesta e si appresta ad inviare il file richiesto.

Postcondizioni: Il *server* ha ricevuto la richiesta e si appresta ad inviare il file richiesto.

Extension Points:

- **Il server non è raggiungibile**: c'è stato un errore nel tentativo di connettersi al *server*. Viene visualizzato un messaggio d'errore (UC 1.7).

4.2.9 UC3.2 Caricamento Mesh

Attori Principali: Client.

Precondizioni: Il *client* ha inviato il nome della mesh da caricare .

Descrizione: Il *server* legge il file OBJ della mesh e lo invia al *client*.

Postcondizioni: Il *client* riceve la mesh richiesta.

4.2.10 UC4 Caricamento informazioni delle mesh salvate

In figura 4.5 sono riportati i casi d'uso per il caricamento delle informazioni relative alle mesh salvate su *server*.

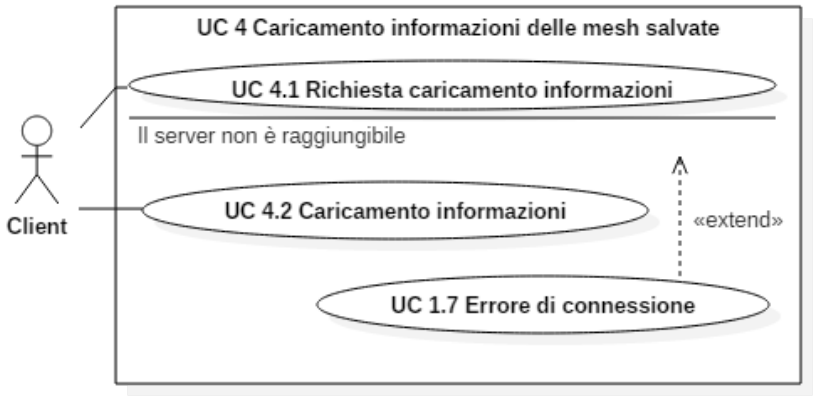


figura 4.5: Use Case - UC4: Caricamento informazioni delle mesh salvate

UC4: Caricamento informazioni delle mesh salvate

Attori Principali: Client.

Precondizioni: Il *client* richiede una lista di informazioni quali nome, data di creazione e volume delle mesh salvate su *server*.

Descrizione: Il *server* elabora la richiesta ricevuta ed invia le informazioni al *client*.

Postcondizioni: Il *client* riceve le informazioni richieste dal *server*.

4.2.11 UC4.1 Richiesta caricamento informazioni

Attori Principali: Client.

Precondizioni: Il *client* richiede informazioni sulle mesh salvate nel *server* inviando una richiesta tramite protocollo HTTP.

Descrizione: Il *server* elabora la richiesta e si appresta ad inviare le informazioni richieste.

Postcondizioni: Il *server* ha ricevuto la richiesta e si appresta a recuperare ed inviare le informazioni.

Extension Points:

* **Il server non è raggiungibile**: c'è stato un errore nel tentativo di connettersi al *server*. Viene visualizzato un messaggio d'errore (UC 1.7).

4.2.12 UC4.2 Caricamento informazioni

Attori Principali: Client.

Precondizioni: Il *client* ha richiesto l'invio di informazioni sulle mesh salvate nel *server*.

Descrizione: Il *server* recupera le informazioni richieste dalle mesh disponibili e le invia al *client*.

Postcondizioni: Il *client* riceve le informazioni richieste.

4.3 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)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

O = obbligatorio

D = desiderabile

Z = opzionale

Nelle tabelle 4.1, 4.2, 4.3 e 4.4 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

4.3.1 Requisiti Funzionali

Requisito	Descrizione	Use Case
RFO-1	Il sistema deve permettere di interagire con le mesh salvate su tablet	UC1
RFO-1.1	L'utente può visualizzare una lista delle mesh salvate localmente	UC1.1
RFO-1.2	Il sistema permette di caricare una mesh dal server	UC1.2
RFO-1.2.1	L'utente può aggiornare la lista di mesh con le mesh presenti sul server	UC1.2
RFO-1.3	L'utente può visualizzare una singola mesh come oggetto tridimensionale	UC1.3
RFO-1.3.1	L'utente può ruotare la mesh visualizzata	UC1.3.1
RFO-1.3.2	L'utente può ingrandire/rimpicciolire la mesh visualizzata	UC1.3.2
RFO-1.3.3	L'utente può visualizzare il volume calcolato della mesh caricata	UC1.3.3
RFD-1.3.4	L'utente può visualizzare il solo wireframe della mesh	UC1.3.4
RFD-1.3.5	L'utente può visualizzare la mesh alla quale sono state applicate delle textures di default	UC1.3.5
RFO-1.3.6	L'utente può abbandonare la visualizzazione di una mesh e tornare alla lista di mesh	UC1.3.6
RFZ-1.3.7	L'utente può scegliere e applicare delle textures alla mesh visualizzata	UC1.3.5
RFO-1.4	L'utente può eliminare una mesh dalla lista delle mesh	UC1.4
RFO-1.4.1	L'utente può eliminare una mesh salvata localmente	UC1.4
RFO-1.5	L'utente può tornare all'Activity precedente	UC1.5
RFD-1.6	Se non vi sono mesh salvate localmente, l'utente viene informato con un messaggio	UC1.6
RFD-1.7	Se la connessione con il server fallisce, l'utente viene informato con un messaggio d'errore	UC1.7
RFO-2	Il sistema permette l'elaborazione di un Point Cloud	UC2
RFO-2.1	Il sistema permette di ricevere Point Cloud dal client	UC2.1
RFO-2.1.1	Il sistema permette di salvare i Point Cloud ricevuti su file PCD	UC2.1
RFO-2.2	Il sistema permette di filtrare un Point Cloud	UC2.2
RFO-2.2.1	Il sistema permette di rimuovere i punti isolati di un Point Cloud	UC2.2, UC2.2.1
RFD-2.2.2	Il sistema permette di rimuovere i punti esterni di un Point Cloud	UC2.2, UC2.2.2
RFO-2.2.3	Il sistema permette di rimuovere i punti del pavimento di un Point Cloud	UC2.2, UC2.2.3

RFO-2.2.4	Il sistema permette di rimuovere i punti troppo vicini di un Point Cloud	UC2.2, UC2.2.4
RFO-2.3	Il sistema permette di estrarre da un Point Cloud i punti del solo oggetto scansionato	UC2.3
RFO-2.4	Il sistema permette di generare una mesh tridimensionale a partire dal Point Cloud filtrato	UC2.4
RFO-2.4.1	La mesh tridimensionale generata viene salvata in formato OBJ	UC2.4
RFD-2.4.2	La mesh tridimensionale generata viene salvata in formato VTK	UC2.4
RFO-2.5	Il sistema permette di calcolare il volume di una mesh tridimensionale	UC2.5
RFO-2.5.1	Il volume calcolato viene salvato su un file di testo nella stessa cartella della mesh	UC2.5
RFZ-2.5.2	Il volume calcolato viene salvato direttamente nel file della mesh	UC2.5
RFO-3	Il sistema permette di inviare al client una mesh	UC3
RFO-3.1	Il client può richiedere al server l'invio di una specifica mesh	UC3.1
RFO-3.2	Il sistema permette di leggere da file ed inviare al client una mesh	UC3.2
RFD-4	Il sistema permette di inviare al client informazioni riguardo le mesh generate	UC4
RFD-4.1	Il client può richiedere al server informazioni sulle mesh prodotte	UC4.1
RFD-4.2	Il sistema permette di recuperare una serie di informazioni delle mesh salvate su server ed inviare al client i dati richiesti	UC4.2

tabella 4.1: Tabella del tracciamento dei requisti funzionali

4.3.2 Requisiti Qualitativi

Requisito	**Descrizione**	**Fonti**
RQO-1	Lo sviluppo lato client deve rispettare la separazione tra *business logic* ed interfaccia grafica	Obbiettivo qualitativo interno
RQO-2	Lo sviluppo lato server deve separare la ricezione delle richieste HTTP dalla loro elaborazione	Obbiettivo qualitativo interno
RQD-3	Il prodotto deve superare tutti i test di sistema	Obbiettivo qualitativo interno

tabella 4.2: Tabella del tracciamento dei requisti qualitativi

4.3.3 Requisiti di Vincolo

Requisito	Descrizione	Fonti
RVO-1	L'applicativo lato client dev'essere pienamente compatibile con il sistema *Android* e i *device Tango*	Decisioni interne
RVO-2	L'applicativo lato client dev'essere disponibile in lingua inglese	Decisioni interne
RVO-3	L'applicativo lato server deve risiedere ed essere eseguibile su server *Apache*	Decisioni interne

tabella 4.3: Tabella del tracciamento dei requisti di vincolo

4.3.4 Requisiti Prestazionali

Requisito	Descrizione	Fonti
RPO-1	Le operazioni di filtraggio del Point Cloud devono essere svolte efficientemente	Richiesta committente
RPO-2	Le mesh generate devono essere sufficientemente leggere da poter essere inviate via web al tablet	UC 3.2
RPD-3	Il rendering delle mesh su tablet dev'essere veloce	UC 1.3
RPD-3.1	La visualizzazione delle mesh su tablet dev'essere fluida	UC 1.3
RPD-4	Il refresh della lista di mesh non deve bloccare l'applicazione	UC 1.2
RPD-4.1	Il refresh della lista di mesh dev'essere svolto efficientemente	UC 1.2
RPD-5	Il caricamento della lista di mesh locali dev'essere veloce	UC 1.1
RPD-6	In caso di mancata connessione col server, l'errore di connessione dev'essere comunicato prima possibile	UC 1.7
RPD-6.1	In caso di mancata connessione col server, l'applicativo lato client deve aspettare una risposta per il minor tempo possibile	UC 1.7

tabella 4.4: Tabella del tracciamento dei requisti prestazionali

Capitolo 5

Progettazione e sviluppo

In questo capitolo vengono presentate le tecnologie e gli strumenti adottati e vengono spiegate le scelte progettuali adottate

5.1 Tecnologie e strumenti

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

5.1.1 Codice

Segue la lista dei linguaggi 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*, ed in particolare *PCL* sono disponibili in questo linguaggio.
* **PHP**[3]: Il linguaggio usato per ricevere ed inviare le richieste *HTTP* necessarie alla comunicazione tra *Server* e dispositivo.
* **Python**[4]: Il linguaggio usato in combinazione con *PHP* per elaborare le richieste ed implementare la logica del lato *Server*. Essendo un linguaggio particolarmente adatto alla computazione numerica è stato usato inoltre per il calcolo del volume di una mesh.

[1] https://www.java.com
[2] www.cplusplus.com
[3] https://secure.php.net
[4] https://www.python.org

5.1.2 IDE ed editor

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

* **Android Studio**[5]: L'*IDE* ufficiale per le applicazioni *Android*.
* **QT**[6]: L'*IDE* scelto per lo sviluppo del codice *C++*.
* **Geany**[7]: L'*editor* di testo usato per scrivere gli *script PHP* e *Python*.

5.1.3 Framework

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

* **PCL, Point Cloud Library**[8]: Una delle librerie di maggior rilievo nel campo della *Computer Vision*, *Open Source*, scritta in C++, di algoritmi per l'elaborazione di *Point Cloud* tridimensionali. Contiene algoritmi per il filtraggio, la segmentazione, la *registration* e il *meshing* di Point Cloud, per citarne alcuni.
  La libreria è ampiamente utilizzata da qualsiasi applicativo debba trattare nuvole di punti, ed oltre ad essere utile ed efficiente è anche ben documentata, con molti esempi d'utilizzo reperibili online.
  Le notevoli potenzialità della libreria sono state utilizzate intensivamente nell'applicativo con lo scopo di isolare i punti appartenenti all'oggetto scansionato dal resto del Point Cloud, ed effettuarne quindi il meshing.
* **Tango API**[9]: Le *API* ufficiali, disponibili in linguaggio Java, C++ e C#, per lo sviluppo di applicazioni *Tango*.
* **Gradle**[10]: È stato usato come *tool* di *build* per tutta l'applicazione *Android*. É inoltre il *build-tool* di default dell'ambiente *Android Studio*.
* **OkHttp**[11]: È stata usato come *framework* di riferimento per le richieste *http*, come indicato nella *Android Best Practices*.
* **TangoUx**[12]: *Famework* messo a disposizione da *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* permette di richiamare metodi 'nativi' (scritti in *C/C++*) dal codice *Java*. É stato usato nell'ultimo prototipo dell'applicazione, per esplorare le funzionalità della libreria (scritta in C++) GOICP.
* **Firebase**[14]: *Framework* associato all'omonimo *web service* offerto da *Google* per lo scambio di messaggi tra applicazioni *Android* e un *server*.
* **GOICP**[15]: libreria che implementa una variante dell'algoritmo ICP per la *Registration* di set di *Point Cloud*.

[5] https://developer.android.com/studio/index.html
[6] https://www.qt.io/ide
[7] https://www.geany.org
[8] http://pointclouds.org
[9] https://developers.google.com/tango/apis/overview
[10] https://gradle.org
[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
[14] https://firebase.google.com
[15] http://iitlab.bit.edu.cn/mcislab/ yangjiaolong/go-icp/

5.1.4 Server

L'applicativo lato *server* è stato sviluppato e testato su un web *server Apache*[16], preferito per la sua facilità di configurazione, modularità ed affidabilità. Il *server Apache* è stato installato sulla macchina del tirocinante.

5.2 Progettazione

Sulla base dei requisiti analizzati nel capitolo *Analisi dei requisiti*(cap. 4) si è svolta la progettazione dell'applicativo. Prima di tutto si è reso necessario suddividere la progettazione lato *client* dal lato *server*; quanto svolto nell'applicativo lato *client* andava infatti inserito ed integrato in un prototipo già esistente, mentre il lato *server* doveva essere progettato da zero. Vediamo ora le scelte progettuali adottate.

5.2.1 Architettura lato client

Al lato *client* è stato necessario prendere coscienza dell'architettura e della logica dell'applicazione prototipale già realizzata, per poter sviluppare un nuovo prototipo che inserisca nuove funzionalità integrandole correttamente nel sistema. Il prototipo sviluppato, denominato *VIC-Tango*, partendo dallo stadio prototipale precedente dell'applicazione (*Samba*), introduce le seguenti migliorie e funzionalità:

* Possibilità di caricare in una lista le *mesh* risultato dell'elaborazione lato *server* di un *Point Cloud*, e poter visualizzare il singolo render di una mesh.
* Un servizio di notifica che riceve messaggi sull'esito dell'elaborazione di un *Point Cloud* da parte del *server*.
* Miglioramento della Camera Preview dell'applicazione.
* Introdotti i framework JNI e GOICP per testare funzionalità di Point Cloud Registration.

Vediamo le scelte progettuali adottate per implementare tali funzionalità.

Lista di mesh

Per implementare una lista di mesh e la possibilità di visualizzare il render di una singola mesh, tutte le classi e le funzionalità associate sono state inserite in un nuovo package `rendering`, come da figura 5.1. Il package contiene le seguenti classi:

* `MeshListActivity`: L'activity principale della funzionalità di visualizzazione della lista di mesh.
* `MeshListManager`: Manager responsabile di gestire la business logic di MeshListActivity, separando così la presentazine dalla logica.

Ed i seguenti sub-package:

* `receivers`: Contiene le classi necessarie a ricevere dati dal *server*, sotto forma di oggetti JSON. Il package, come da figura 5.2, contiene le seguenti classi:

[16] **https://httpd.apache.org/**.

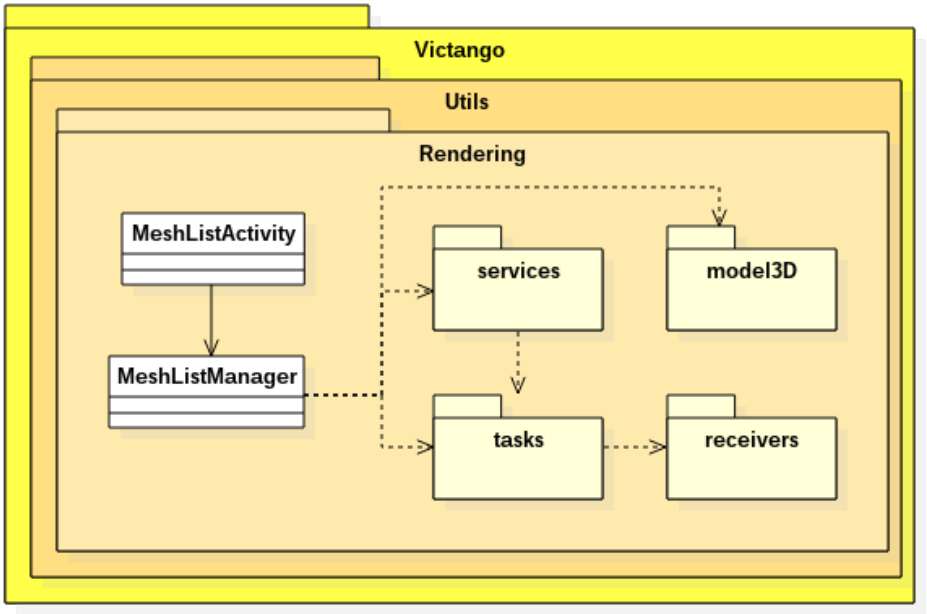


figura 5.1: Diagramma UML del package `rendering`

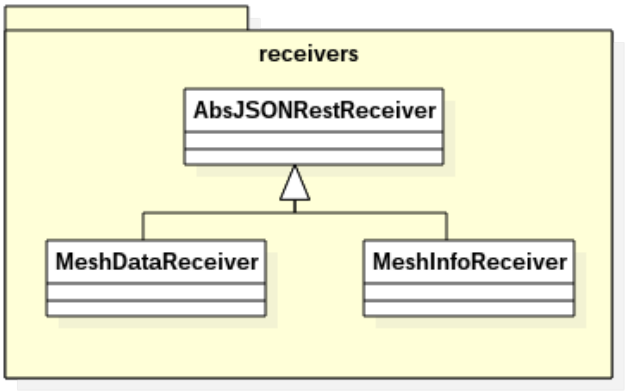


figura 5.2: Diagramma UML del package `receivers`

- `AbsJSONRestReceiver`: Classe astratta che rappresenta un interfaccia REST per la richiesta e ricezione di dati in formato JSON.
- `MeshDataReceiver`: Implementazione di `AbsJSONRestReceiver` che richiede e riceve file OBJ dal *server* come oggetti JSON.
- `MeshInfoReceiver`: Implementazione di `AbsJSONRestReceiver` che richiede e riceve informazioni sulle mesh salvate sul *server* come array di oggetti JSON.

* `tasks`: Contiene classi che eseguono asincronamente compiti temporalmente dispendiosi, quali il caricamento e salvataggio delle mesh e il caricamento da *server*. Il package, come da figura 5.3, contiene le seguenti classi:

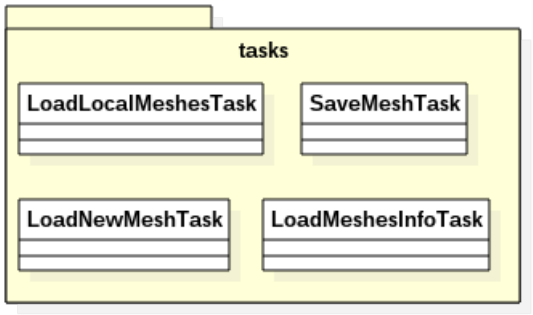


figura 5.3: Diagramma UML del package `tasks`

  - `LoadLocaMeshesTask`: Classe per il caricamento asincrono delle mesh salvate localmente sul *device*.
  - `LoadMeshesInfoTask`: Classe per il caricamento asincrono delle informazioni sulle mesh salvate nel *server*.
  - `LoadNewMeshTask`: Classe per il caricamento asincrono di una mesh dal *server*.
  - `SaveMeshTask`: Classe per il salvataggio asincrono di una mesh ricevuta sulla memoria locale del *device*. La mesh viene salvata in un file OBJ.

* `services`: Contiene classi che implementano servizi che necessitano di essere per lungo tempo in background. Il package, come da figura 5.4, contiene le seguenti classi:

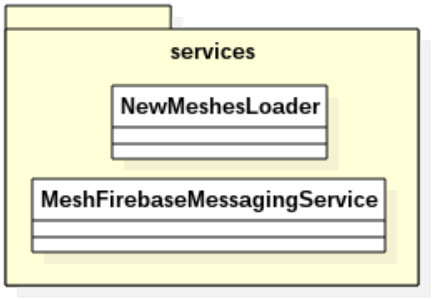


figura 5.4: Diagramma UML del package `services`

  - `NewMeshesLoader`: Servizio che effettua il refresh della lista di mesh locali caricando dal *server* le nuove mesh disponibili.

  - `MeshFirebaseMessagingService`: Servizio che, in background, aspetta di ricevere messaggi dal *server* attraverso il web-service Firebase, ed alla ricezione di un messaggio genera una notifica di sistema *Android* per informare l'utente.

* `model3D`: Contiene classi per il rendering e la visualizzazione di mesh (file OBJ). Il package corrisponde all'applicazione open source *3D Model Viewer Open Source*[17] che in quanto non implementata dal tirocinante, non verrà qui documentata.

Camera Preview

La *Camera Preview*, cioè un piccolo riquardo nell'applicazione che mostra quanto ripreso dalla fotocamera, è stata reimplementata, abbandonando la vecchia implementazione `VideoRenderer`, che soffriva di problemi di stabilità e prestazioni. É stata creata quindi nel package `camera` (vedi fig. 5.5) una classe `RGBBoxManager` che sfrutta funzionalità delle *Tango API* per il controllo della fotocamera RGB-IR dei *device Tango*, ottenendo così una preview più affidabile e fluida.

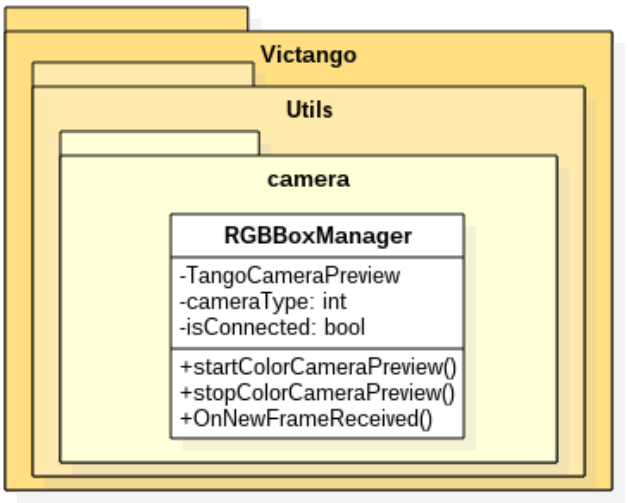


figura 5.5: Diagramma UML del package `camera`

JNI e Point Cloud Registration

In *VIC-Tango* è stata aggiunta una nuova strategia di ricostruzione del Point Cloud, rispettando l'architettura già presente e sfruttando il pattern *strategy*, una nuova classe `GOICPPCloudReconstruction` nel package `Reconstruction`, che sfrutta le funzionalità della libreria esterna GOICP, come da figura 5.6. Inoltre per integrare la libreria GOICP, scritta nativamente in C++, è stato necessario importare nel progetto il framework JNI, che permette di richiamare codice nativo da codice Java.

[17] **https://play.google.com/store/apps/details?id=org.andresoviedo.dddmodel&hl=it**.

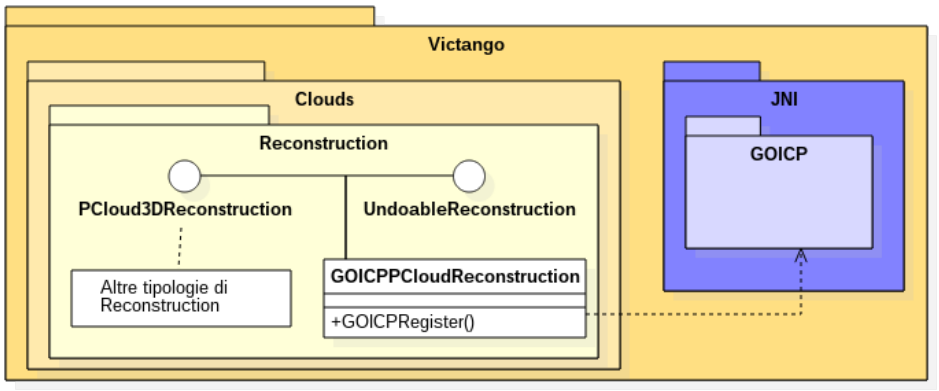


figura 5.6: Diagramma UML del package `Reconstruction`

5.2.2 Architettura lato server

Il lato *server* dell'applicazione è stato progettato da zero, basandosi quindi sui requisiti rilevati in fase d'analisi e sulle necessità del committente. L'architettura *server* espone una serie di servizi al *client*, quali la ricezione di file PCD, o l'invio di mesh ed info sulle mesh salvate, oltre ovviamente all'elaborazione dei *Point Cloud ricevuti*. Come

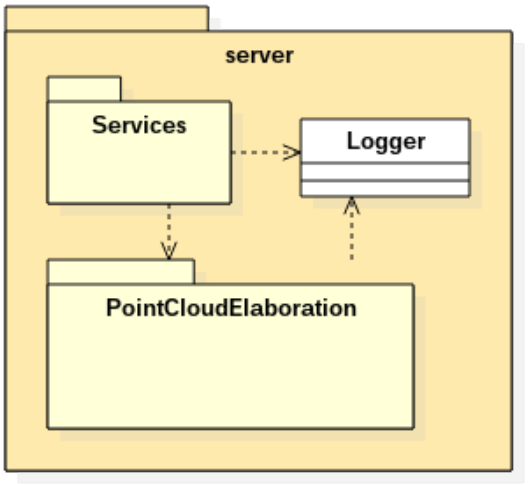


figura 5.7: Diagramma UML del package `server`

evidenziato in figura 5.7 il lato server è così suddiviso:

* `Services`: Package che espone le funzionalità del *server* al *client*.
* `PointCloudElaboration`: Package che contiene tutte le funzionalità necessarie all'elaborazione di un *Point Cloud*.

* `Logging`: Singleton che espone funzionalità di logging per i servizi del *server* e per il processo di elaborazione dei *Point Cloud*.

Vediamo in maggior dettaglio le componenti dell'architettura.

Services

Il package services racchiude le funzionalità del *server*, rappresentate in figura 5.8 dalle seguenti classi:

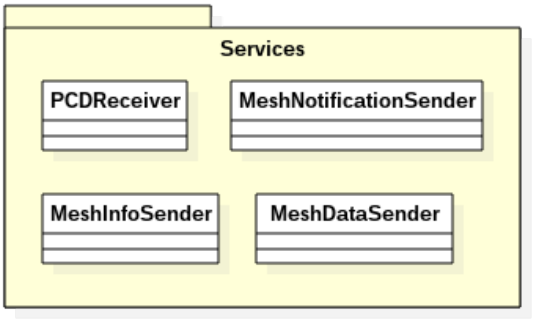


figura 5.8: Diagramma UML del package `Services`

* `PCDReceiver`: Classe responsabile di ricevere dal *client* i *Point Cloud* acquisiti e di salvarli su un file PCD locale. Una volta salvato il *Point Cloud* su file PCD ne viene avviata l'elaborazione.

* `MeshDataSender`: Classe responsabile di inviare una specifica mesh via protocollo HTTP al *client*.

* `MeshInfoSender`: Classe responsabile di inviare una serie di informazioni quali nome, data di creazione e volume delle mesh salvate, via protocollo HTTP, al *client*.

* `MeshNotificationSender`: Classe responsabile di inviare un messaggio riguardante l'esito dell'elaborazione di un Point Cloud al web-service Firebase. Firebase provvederà poi autonomamente a recapitare il messaggio al *device Tango*.

PointCloudElaboration

Il package racchiude tutte le funzionalità di manipolazione, filtraggio, meshing e calcolo del volume di un Point Cloud. Come da figura 5.9 il package è formato dalle seguenti classi:

* `VolumeEstimator`: Classe che espone le funzionalità per il calcolo del volume di una mesh. Il volume viene calcolato leggendo il file VTK prodotto da `PointCloudMesher` dopo l'elaborazione di un Point Cloud.

* `PointCloudMesher`: Classe che espone le funzionalità per il meshing di un PointCloud, può salvare la mesh prodotta in formato OBJ e VTK.

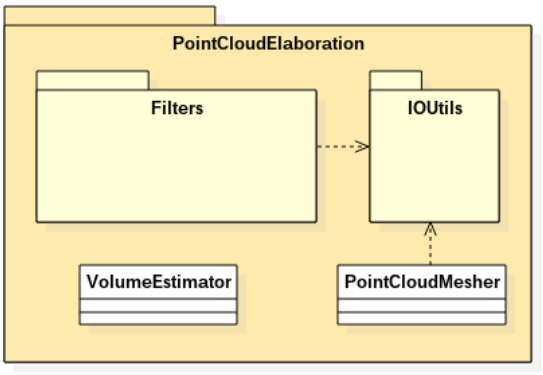


figura 5.9: Diagramma UML del package `PointCloudElaboration`

E dai seguenti sub-package:

* `Filters`: Il package racchude tutte le funzionalità per il filtraggio e l'eliminazione di punti da un Point Cloud. É formato dalle seguenti classi (vedi fig. 5.10):

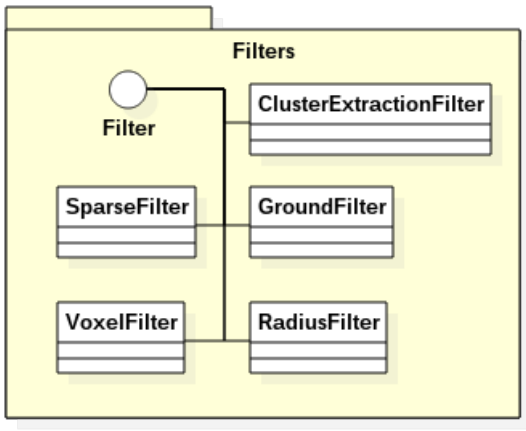


figura 5.10: Diagramma UML del package `Filters`

  - `Filter`: Interfaccia che definisce le operazioni di base per il filtraggio di Point CLoud
  - `SparseFilter`: Classe che implementa `Filter` per il filtraggio dei punti isolati di un Point Cloud.
  - `RadiusFilter`: Classe che implementa `Filter` per il filtraggio dei punti esterni (non appartenenti all'oggetto scansionato) di un Point Cloud.
  - `VoxelFilter`: Classe che implementa `Filter` per il filtraggio dei punti doppi o eccessivamente ravvicinati di un Point Cloud.
  - `GroundFilter`: Classe che implementa `Filter` per il filtraggio dei punti che rappresentano il piano del pavimento di un Point Cloud.

  - `ClusterExtractionFilter`: Classe che implementa `Filter` per l'estrazione da un Point Cloud della sola nuvola di punti rappresentante l'oggetto scansionato.

* `IOUtils`: Il package racchiude tutte le funzionalità per il salvataggio e caricamento di file inerenti all'applicativo, quindi i formati PCD, OBJ e VTK. É formato dalle classi (vedi fig. 5.11):

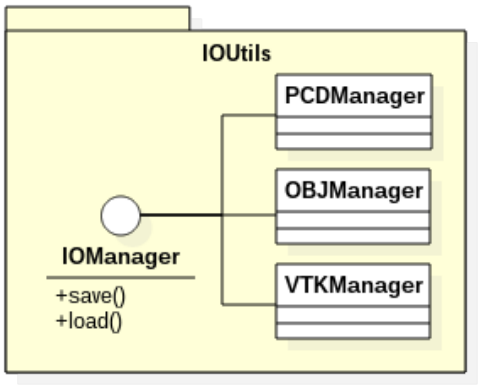


figura 5.11: Diagramma UML del package `IOUtils`

  - `IOManager`: Interfaccia che definisce le operazioni di base per un manager di Input/Output.
  - `PCDManager`: Classe che implementa `IOManager` per il salvataggio e caricamento di file PCD.
  - `OBJManager`: Classe che implementa `IOManager` per il salvataggio e caricamento di file OBJ.
  - `VTKManager`: Classe che implementa `IOManager` per il salvataggio e caricamento di file VTK.

5.2.3 Design Pattern utilizzati

Nella progettazione dell'architettura, al fine di aumentarne la robustezza e favorirne l'estensione, sono stati usati i seguenti Design Pattern.

Strategy

Il Design Pattern Strategy è stato usato per separare la dichiarazione di alcuni algoritmi dalla loro implementazione. L'utilizzo di questo pattern risulta perticolarmente utile nel caso in cui si voglia lasciare un parte del sistema aperta all'aggiunta di migliorie e funzionalità non inizialmente preventivate. Per questo è stato utilizzato nel lato *server* per definire le strategie di filtraggio e di I/O da file (fig. 5.12), mentre nel lato *client* è stata sfruttato uno *Strategy* già implementato per aggiungere trasparentemente una nuova tecnica di ricostruzione del *Point Cloud* (`GOICPPCloudReconstruction`, vedi fig. 5.6).

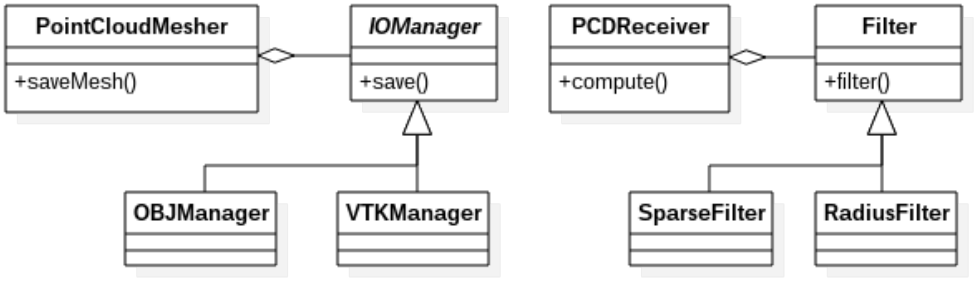


figura 5.12: Diagramma UML del Design Pattern `Strategy`

Model View Presenter

È stato scelto il Design Pattern Model View Presenter per separare la logica dell'applicazione lato *client* dalla sua rappresentazione e per seguire le Android Best Practices. Il pattern è stato utilizzato nell'implementazione della "lista di mesh" dell'applicativo, come da figura 5.13.

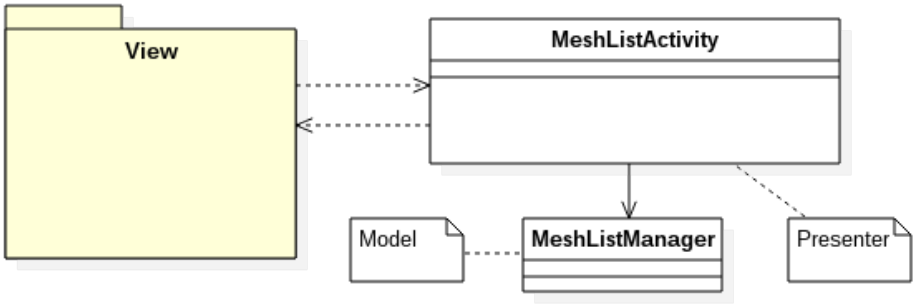


figura 5.13: Diagramma UML del Design Pattern `MVP`

Nel sistema Android la parte *View* è rappresentata da soli file XML, quindi è totalmente passiva, mentre tutta la logica viene accentrata nel *Presenter*, che fa da tramite tra la *View* e il *Model*.

Singleton

Il Design Pattern è stato utilizzato per controllare gli accessi alle classi che utilizzano il Logger per registrare le proprie eleborazioni. Un suo esempio d'utilizzo nell'applicativo è in figura 5.14.

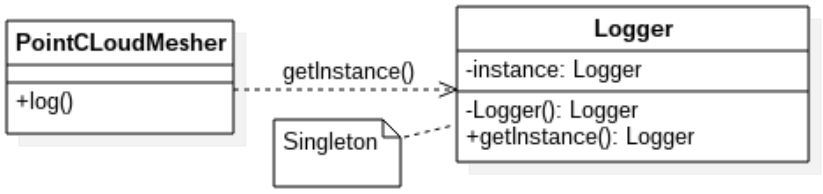


figura 5.14: Diagramma UML del Design Pattern `Singleton`

5.3 Sviluppo

5.3.1 Lato client

VIC-Tango è una versione rifinita e migliorata del prototipo di partenza (*Samba*), del quale sono stati risolti svariati bug, immancabili allo stadio prototipale. Introduce inoltre alcune nuove funzionalità che migliorano la User experience e lasciano la porta aperta a sviluppi futuri.

Firebase

Firebase è un *framework* associato all'omonimo *web service* offerto da *Google* per lo scambio di messaggi tra applicazioni *Android* e un *server*. In particolare è stato utlizzato il servizio *Firebase* di cloud messaging per la gestione e invio affidabile di messaggi a più dispositivi differenti.
Il suo utilizzo in VIC-Tango ha permesso di implementare un servizio di notifica da parte del *server*, che può così inviare messaggi asincroni al device Tango, il quale li riceverà come notifica di sistema Android, informandolo ad esempio che il Point Cloud inviato è stato elaborato e una nuova mesh è stata generata. L'utente può così continuare l'ispezione e acquisire nuovi Point Cloud e, alla ricezione della notifica, caricare e visualizzare automaticamente le mesh quando disponibili. In figura 5.15 vediamo un esempio di notifica ricevuta da Firebase durante il normale utilizzo dell'applicativo.

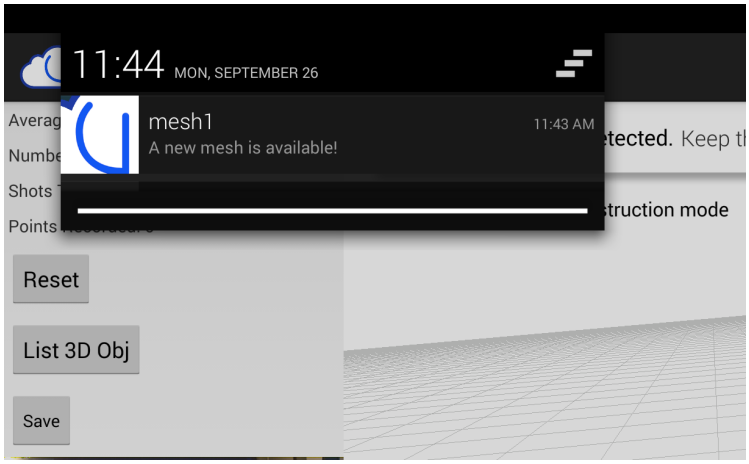


figura 5.15: Una notifica di sistema sull'esito del meshing di un Point Cloud

Camera preview

Fin dal prototipo precedente era stata introdotta una piccola preview di quanto ripreso dalla fotocamera RGB-IR. Tuttavia la funzionalità era stata implementata utilizzando una soluzione general-purpose applicabile a più dispositivi Android, che causava però problemi prestazionali e crash occasionali dell'applicativo.
La preview è stata quindi reimplementata utilizzando le funzionalità esposte dalle Tango API, soluzione inizialmente scartata perchè quest'ultime erano mal documentate, ottenendo un miglioramento delle prestazioni e la correzione dei bug rilevati.

Visualizzazione di mesh

Nel prototipo è stata introdotta la possibilità di caricare e visualizzare le *mesh* risultato dell'elaborazione lato *server* di un *Point Cloud*. Una *mesh* poligonale è una collezione di vertici, spigoli e facce che definiscono la forma di un oggetto poliedrico; in *Samba* è ottenuta a partire dalla nuvola di punti elaborata. Tale *mesh* viene salvata dal *server* in formato *OBJ*, comunemente usato nelle applicazioni 3D. Dall'applicazione viene quindi richiesto di caricare e salvare nella memoria locale del *tablet* le mesh disponibili sul *server*, delle quali viene visualizzata su *tablet* una lista, come in figura 5.16.

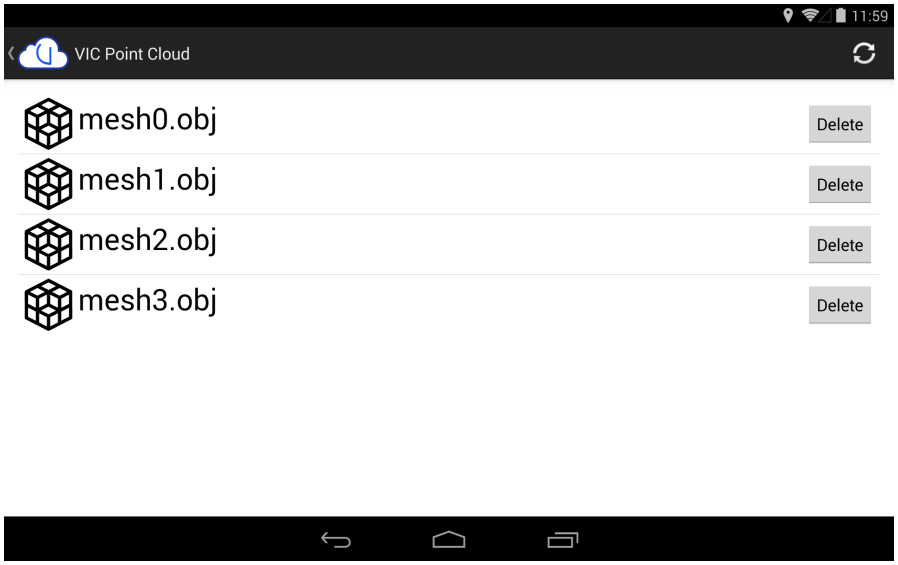


figura 5.16: La lista di *mesh* visualizzata su *tablet*

Selezionando poi una singola mesh dalla lista è possibile visualizzarne il rendering tridimensionale, come in figura 5.17. Utilizzando i comandi in alto a destra è possibile attivare/disattivare la modalità *texture*, in cui al wireframe dell'oggetto vengono applicate delle textures di default, e visualizzare il volume calcolato dell'oggetto.

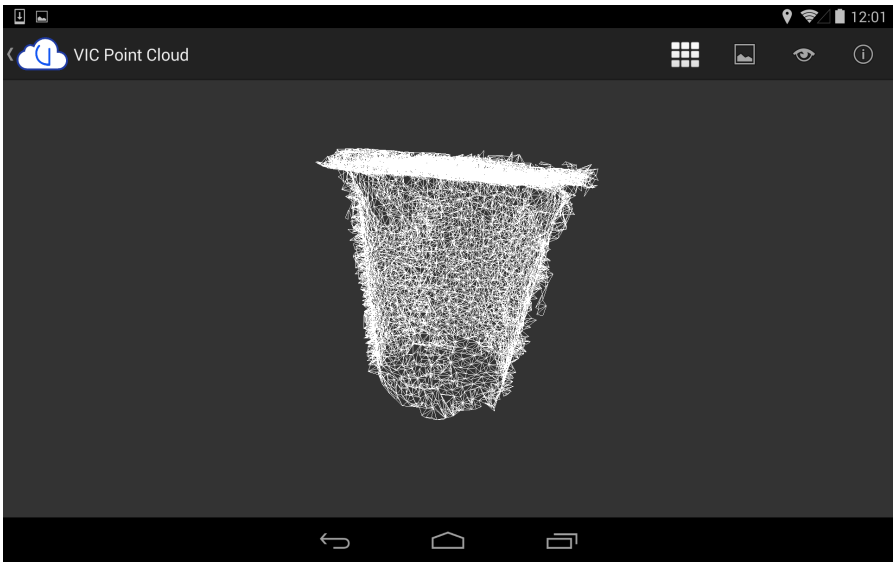


figura 5.17: Rendering su *tablet* di una *mesh* di un cestino

JNI e Point Cloud Registration

La Java Native Interface o JNI è un framework del linguaggio Java che consente al codice Java di richiamare codice cosiddetto "nativo", ovvero specifico di un determinato sistema operativo scritto in altri linguaggi di programmazione, in particolare C/C++. L'integrazione della JNI con lo sviluppo Android, in particolare il suo utilizzo nell'IDE Android Studio, è un compito tutt'altro che semplice, dato il supporto ancora scarso all'integrazione del framework con l'IDE e il build-tool Gradle.
Dopo svariati tentativi iniziati già i primi giorni di stage è stato infine possibile usufruire della JNI utilizzando versioni sperimentali di Android Studio e Gradle.
Attraverso la JNI si è provato ad importare la libreria *PCL*, così da poterne utilizzare le potenti funzionalità direttamente dal *tablet*. Purtroppo l'integrazione con il build-tool Gradle richiede un makefile apposito, compito improponibile per una libreria notevolmente complessa come *PCL*, le cui potenzialità sono quindi rimaste nel lato *server*.
Si è riuscito invece ad importare, date le ridotte dimensioni, una libreria per la *Registration* di Point Cloud. Col termine *Registration* si indica il processo che trasforma in coordinate assolute e allinea due set di Point Cloud in un unico set che minimizza le distanze tra punti corrispondenti (ad es. fig. 5.18).
La libreria in questione, GOICP, implementa una versione ottimizzata dell'algoritmo ICP (Iterative Closest Point), spesso utilizzato in applicazioni nelle quali sia necessario ricostruire una superficie tridimensionale a partire da più scansioni, che punta a minimizzare la distanza tra i punti delle due Point Cloud.
L'algoritmo ICP tenta di minimizzare l'errore quadratico medio (in inglese Mean Squared Error, MSE) cioè la discrepanza quadratica media fra i valori dei punti della coppia di Point Cloud, definita come:

$$MSE = \frac{\sum_{i=1}^{n}(x_i - x_i')^2}{n}$$

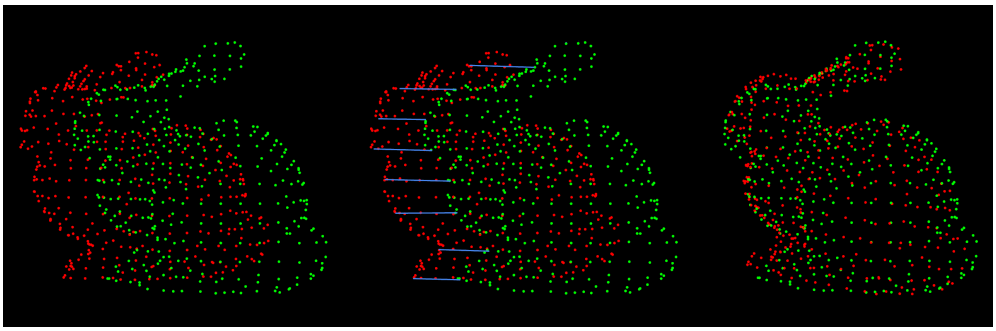


figura 5.18: *Point Cloud Registration* tramite l'algoritmo *ICP*

Si è testato GOICP sulle coppie di Point Cloud acquisite con VIC-Tango, per ovviare al problemi già presentati del *drifting* (fig. 1.4a) e del *ghosting* (fig. 1.4b); si sono ottenuti però risultati insoddisfacenti. Sono sorti infatti i seguenti problemi:

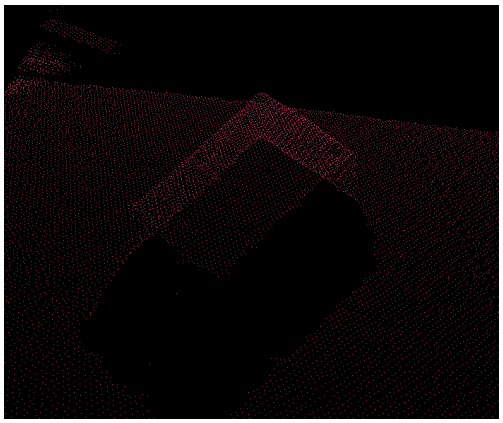
* Il processo è eccessivamente dispendioso da effettuare su *tablet*, con tempi di elaborazione tra i 10 e i 60 secondi.
* L'algoritmo ICP, basandosi sul MSE come valore di bontà, converge spesso lentamente e a soluzioni errate, per colpa della grande quantità di punti planari sul pavimento, che rendono l'errore quadratico medio un valore non attendibile su cui basare il successo di una *Registration*

Come si vede in figura 5.19c l'algoritmo, pur ottenendo un basso valore di MSE, non effettua la *Registration* desiderata, dato che vengono allineati perlopiù i punti planari.

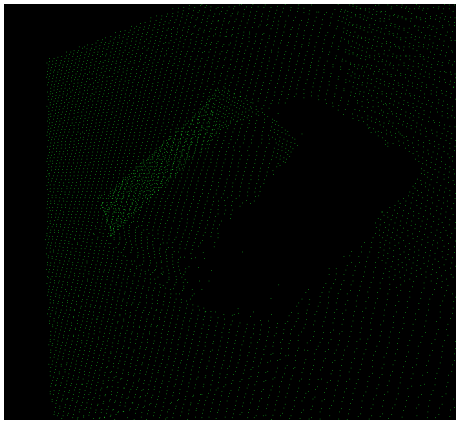
Per questi motivi la *registration* tramite GOICP è rimasta in uno stadio prototipale, che necessita di almeno una delle seguenti correzioni:

* Trovare una misura più adatta del MSE per valutare la convergenza dell'algoritmo ICP
* Alternativamente, eliminare dal Point Cloud i punti planari appartenenti al pavimento così da poter utilizzare l'errore quadratico medio come affidabile valore di convergenza
* Ottimizzare GOICP o implementare un algoritmo ICP ad hoc per ridurre drasticamente i tempi di elaborazione

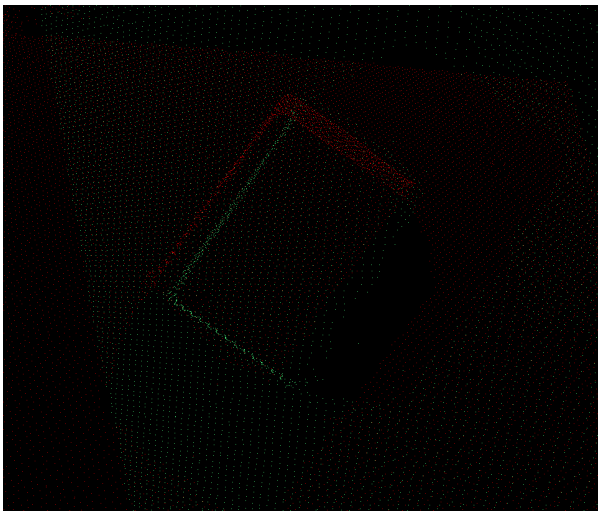
Tali correzioni trattano problemi non triviali, che l'esiguo tempo di tirocinio rimasto non mi ha permesso di affrontare adeguatamente. La Point Cloud Registration rimane quindi una funzionalità necessaria ad assicurare un'affidabile e corretta ricostruzione degli oggetti scansionati, aperta a sviluppi futuri.



(a) Prima scansione



(b) Seconda scansione



(c) Risultato di GOIPC

figura 5.19: Errore nella *Registration* con *GOICP*

5.3.2 Lato server

Il lato *server* dell'applicazione si occupa di ricevere dai *Tango device* i Point Cloud acquisiti, elaborarli utilizzando la libreria *PCL* per estrarre i punti rappresentanti il solo oggetto scansionato, produrre una mesh dell'oggetto così ottenuto e calcolarne il volume. Invia poi quando necessario i risultati elaborati ai dispositivi che li richiedono. Quando un Point Cloud viene ricevuto dal *server* tramite una richiesta HTTP POST, inserito in un oggetto JSON, viene salvato in un file PCD. Il formato PCD (Point Cloud Data) usato nella libreria *PCL* è molto semplice: è formato da un header contenente alcune informazioni utili come il numero totale di punti e il tipo e numero di valori associati ad ogni punto; segue poi l'effettivo Point Cloud codificato riga per riga, dove ogni riga contiene una successione di valori che rappresenta un punto.
Il file salvato viene poi elaborato sfruttando la libreria *PCL*.

5.3.3 Elaborazione di un Point Cloud

La nuvola di punti salvata in un file PCD viene elaborata da un applicativo scritto in C++ che sfrutta le numerose funzionalità della *PCL*. L'elaborazione di un Point Cloud è composta di più passi sequenziali:

1. Viene caricato il nuovo Point Cloud da file PCD salvato localmente
2. Vengono rimossi i punti isolati della nuvola attraverso la libreria Sparse Filtering
3. Vengono rimossi i punti troppo esterni della nuvola attraverso la libreria Radius Filtering
4. Vengono rimossi i punti che corrispondono al piano del pavimento attraverso la libreria Ground Filtering
5. Viene effettuato il *downsample* del *dataset* attraverso la libreria Voxel Filtering
6. Viene estratto il cluster maggiore dai punti rimanenti attraverso la libreria Cluster Extraction
7. Viene effettuato il meshing del cluster estratto, e la mesh viene salvata in due file OBJ (per la visualizzazione lato *client*) e VTK (per il calcolo del volume)
8. Viene calcolato e salvato il volume della mesh prodotta

Per ogni passo di filtraggio la nuvola di punti viene salvata su un distinto file PCD, per poter analizzare visivamente la bontà dell'elaborazione. Di seguito vengono descritti in dettaglio filtri applicati e il processo di meshing.

Sparse filtering

Questo primo passo di filtraggio si occupa di rimuovere dal Point Cloud i punti isolati. Per ogni punto viene effettuata una analisi statistica della distanza media dai suoi vicini. I punti la cui distanza dai vicini supera un certo valore sono quindi isolati e possono essere eliminati dalla nuvola. Un esempio di tale tecnica, sul quale si basa l'implementazione di questo filtro, è spiegato in [12].
In figura 5.20 vediamo i risultati di tale filtro, con una rimozione di circa 10.000 punti.

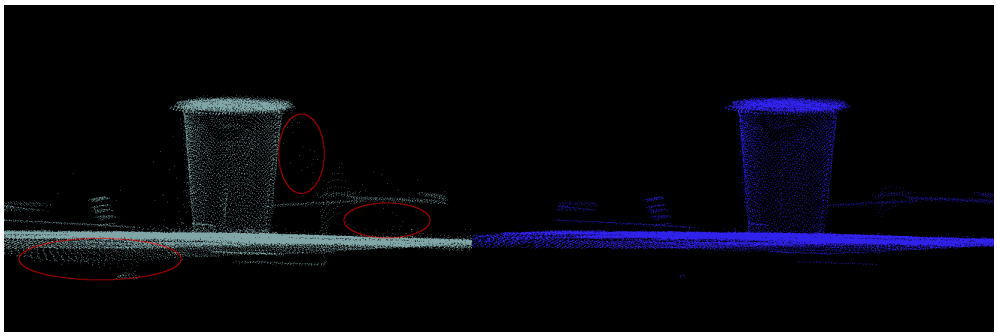


figura 5.20: *Sparse filtering*

Radius filtering

Questo passo di filtraggio si occupa di eliminare i punti troppo distanti dal centro, che solitamente appartengono alle pareti circostanti o comunque ad oggetti diversi da quello scansionato, che sarà sempre verso il centro della nuvola.
Partendo da questo presupposto, il filtro calcola il punto centrale del Point Cloud, media del valore di tutti i punti, e procede con l'eliminare i punti che superano una certa distanza (*radius*), attraverso l'algoritmo *PCL* spiegato in [7], fino a che non è stata eliminata una definita percentuale del point set iniziale.
In figura 5.21 vediamo i risultati di tale filtro, con una rimozione di circa 24.000 punti.

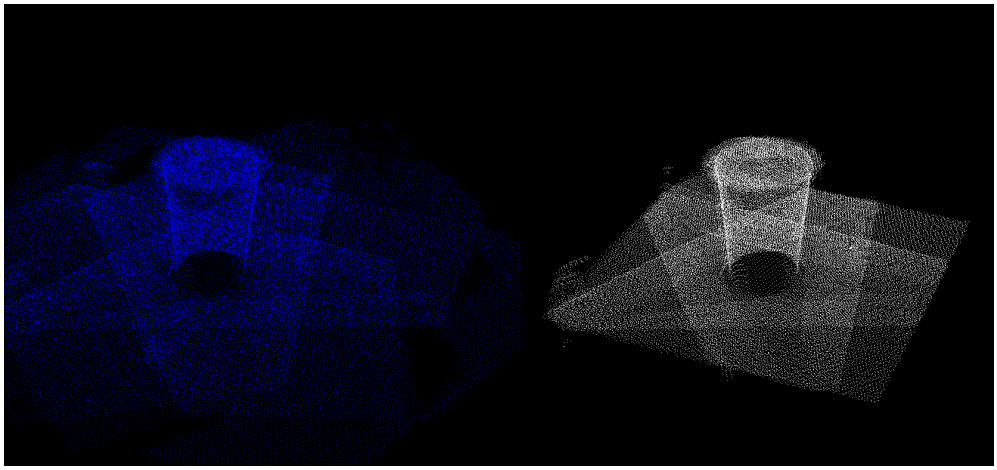


figura 5.21: *Radius filtering*

Ground filtering

Questo passo di filtraggio si occupa di eliminare i punti planari appartenenti al pavimento, problema che si è rilevato essere uno dei più difficili da trattare. In primo luogo non sempre c'è un unico componente planare che rappresenta il pavimento, ma solitamente ve n'è più d'uno, a causa delle imprecisioni nell'acquisizione dei Point Cloud lato *client*. Per l'implementazione del filtro si è fatto uso degli algoritmi di *segmentation* di *PCL* (ad es. come spiegato in [10]), che consentono di estrarre i

punti della nuvola che giacciono sullo stesso piano. Senza ulteriori aggiustamenti però l'algoritmo elimina anche piani utili, come il lato superiore di una scatola o di un tavolo. Si è reso quindi necessario applicare i seguenti passi:

1. Calcolare l'altezza del Point Cloud e dividere la parte superiore da quella inferiore, che deve contenere il pavimento
2. Applicare l'algoritmo di *planar segmentation* alla sotto-nuvola estratta fino a che non è stato eliminato un numero adeguato di punti
3. Ricongiungere la parte inferiore filtrata alla parte superiore del Point Cloud

Il filtro così impostato ottiene ottimi risultati nella maggior parte dei casi. Tuttavia se il Point Cloud originale presenta troppo *drifting*, e i piani del pavimento sono molto distanti tra loro, diventa impossibile eliminarli senza togliere anche punti importanti dell'oggetto scansionato.
In figura 5.22 vediamo i risultati di tale filtro, applicato ad un Point Cloud già sottoposto ai filtri precedenti, con una rimozione di circa 50.000 punti.

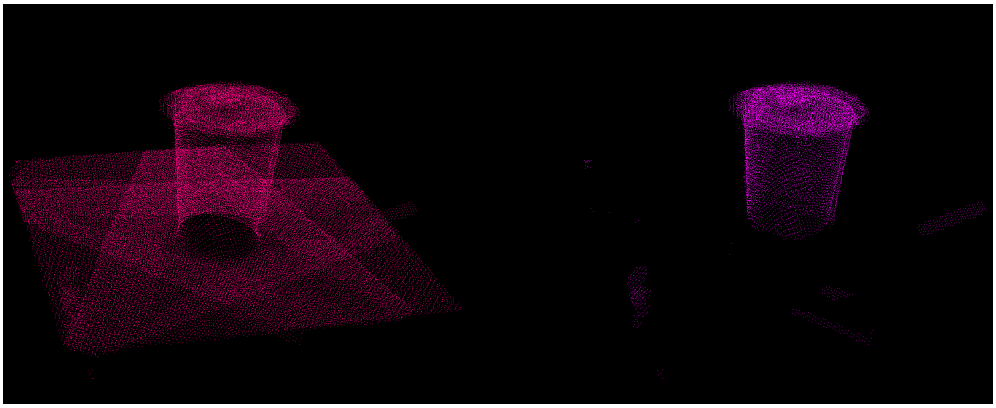


figura 5.22: *Ground filtering*

Voxel filtering

Questo passo di filtraggio si occupa di eliminare i punti doppi e ridurre le dimensioni del Point Cloud per il futuro meshing. Il filtro utilizza la stessa tecnica già implementata nel lato *client*, cioè suddividere lo spazio in tanti *voxel* cubici e calcolare il punto medio di ogni voxel come approssimazione dei punti che ricadono al suo interno. L'implemetazione prende spunto dall'esempio di *PCL* riportato in [4]. Per i risultati del *voxel filtering* si rimanda alle figure 1.5a e 1.5b.

Cluster extraction

L'ultimo passaggio prima del meshing è il *Cluster extraction*, cioè il completo isolamento della nuvola di punti rappresentante oggetto in ispezione. L'algoritmo *PCL* documentato in [5] su cui si base quest'ultimo step di filtraggio suddivide lo spazio del Point Cloud in più parti ed analizzandone statisticamente la densità per ogni parte, riesce a determinare quali sono gli aggregati di punti nella nuvola, distinguibili gli uni dagli altri. L'algoritmo suddivide quindi la nuvola in più cluster, e ne estrae quello

di maggiori dimensioni. Questo filtraggio finale indispensabile al passo successivo di meshing, perchè elimina dal Point Cloud tutti gli elementi estranei all'oggetto.
In figura 5.23 vediamo i risultati di tale filtro, applicato ad un Point Cloud già sottoposto ai filtri precedenti, con una rimozione di circa 4.000 punti. Sono evidenziati in figura alcuni piccoli gruppi di punti che è necessario eliminare, che rappresentano degli artefatti inesistenti nel mondo reale, ma che sono immancabilmente presenti nelle scansioni di un oggetto. L'origine di tali anomalie non è ancora chiara, probabilmente sono causate da riflessi del pavimento.

figura 5.23: *Cluster extraction*

Meshing

Una volta ottenuti i punti della nuvola rappresentanti il solo oggetto scansionato, si può procedere col generarne una mesh tridimensionale. Il processo sfrutta un algoritmo greedy di *PCL*, di cui è riportato un esempio in [6], che, connettendo ogni punto ai propri vicini, genera un mesh composta da migliaia di triangoli che approssima la superficie reale dell'oggetto. In figura 5.24 vediamo i risultati del processo di meshing applicato ad un Point Cloud già sottoposto ai filtri precedenti.

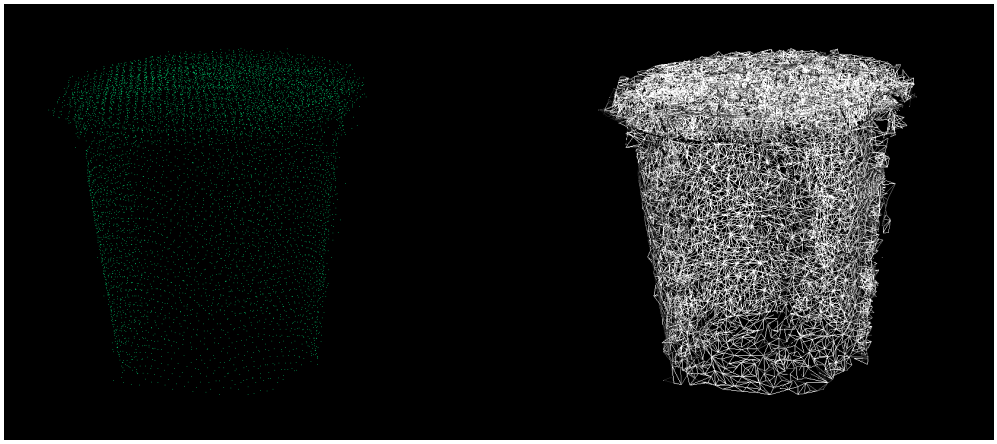


figura 5.24: *Meshing* di un *Point Cloud*

La mesh così ottenuta non è una superficie chiusa, spesso infatti le scansioni non riescono a catturare tutti i punti di un oggetto, e manca completamente il piano di base dell'oggetto scansionato, che non è possibile rilevare. Ciò non influisce particolarmente sul successivo calcolo del volume.

5.3.4 Calcolo del volume

Una volta generata la mesh è possibile calcolarne il volume sfruttando il risultato pubblicato da Cha Zhang e Tsuhan Chen nel loro paper [15] .Viene calcolato, 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*.
Il calcolo del volume, nella sua semplicità e velocità d'elaborazione, pone però delle condizioni sulla qualità della mesh:

* La superficie non necessita di essere chiusa, ma il calcolo ne beneficierebbe
* Il punto scelto come quarto vertice del tetraedro deve appartenere alla mesh
* La mesh non deve contenere triangoli che si sovrappongono o si intersecano

La bontà del calcolo dipende quindi molto dalla qualità del Point Cloud prima del meshing, che dev'essere quanto più assente da fenomeni di *ghosting*, *drifting* (risolvibili in buona parte con algoritmi di *Registration* come spiegato nella sezione 5.3.1) e in generale da artefatti e punti sparsi che non riflettono la vera forma dell'oggetto ispezionato.

Capitolo 6

Test

Vista la natura prototipale del progetto è stato necessario creare un'adeguata suite di Test di Sistema per valutare i risultati dello sviluppo e capire i punti critici dell'applicativo.
Segue la tabella (6.1) di tutti i Test di Sistema con il relativo esito.

Test	**Descrizione**	**Esito**
TS1.0	L'app deve permettere di visualizzare una lista di mesh	Success.
TS1.1	L'app deve caricare correttamente le mesh salvate localmente in una lista	Success.
TS1.2	Selezionando una mesh dalla lista dev'essere possibile visualizzarne il rendering	Success.
TS1.3	Una mesh deve poter essere ruotata ed ingrandita a piacere	Success.
TS1.4	Dev'essere possibile cancellare correttamente una mesh salvata localmente	Success.
TS2.0	L'app deve poter visualizzare messaggi di notifica provenienti da un web-server Firebase	Success.
TS2.1	L'app deve poter ricevere notifiche dal server riguardo l'esito della computazione di una mesh	Success.
TS2.2	L'apertura di una notifica sull'esito di una mesh deve automaticamente caricare localmente la nuova mesh e reindirizzare l'utente alla lista di mesh	Fallito
TS3.0	La *preview* della fotocamera deve mostrare correttamente quello che inquadra.	Success.
TS3.1	La preview della fotocamera non deve cessare di funzionare in caso di riavvio dell'activity principale	Success.
TS4.0	L'applicativo server deve poter ricevere Point Cloud dal client	Success.
TS4.1	L'applicativo server deve poter salvare i Point Cloud ricevuti su file PCD	Success.

TS4.2	L'applicativo server deve poter leggere un file PCD	Success.
TS5.0	L'applicativo server deve poter elaborare un Point Cloud	Success.
TS5.1	L'applicativo server deve poter filtrare i punti isolati di un Point Cloud	Success.
TS5.2	L'applicativo server deve poter filtrare i punti esterni di un Point Cloud	Success.
TS5.3	L'applicativo server deve poter filtrare i punti del pavimento di un Point Cloud	Success.
TS5.4	L'applicativo server deve poter filtrare i punti doppi o ravvicinati di un Point Cloud	Success.
TS5.5	L'applicativo server deve poter estrarre dal Point Cloud una nuvola di punti rappresentante il solo oggetto scansionato	Success.
TS6.0	L'applicativo server deve poter effettuare il meshing di un Point Cloud	Success.
TS6.1	L'applicativo server deve poter salvare una mesh generata in formato OBJ	Success.
TS6.2	L'applicativo server deve poter salvare una mesh generata in formato VTK	Success.
TS7.0	L'applicativo server deve poter calcolare il volume di una mesh	Success.
TS7.1	Il volume calcolato deve essere salvato su file per essere letto in futuro	Success.
TS7.2	Il volume calcolato deve essere salvato nel file stesso della mesh valutata	Fallito
TS8.0	L'applicativo server deve poter inviare una mesh al client	Success.
TS8.1	L'applicativo server deve poter inviare informazioni sulle mesh salvate localmente al client	Success.
TS8.2	L'applicativo server deve poter inviare una notifica sull'esito dell'elaborazione di un Point Cloud al client	Success.

tabella 6.1: Test di Sistema

Su un totale di **28** Test effettuati gli esiti sono stati così suddvisi:

* **26** Test passati con successo
* **2** Test falliti

Con una percentuale di successo del 93%.

Capitolo 7

Conclusioni

Lo scopo principale dello stage era valutare se i prototipi precedentemente realizzati, in collaborazione con un processo d'elaborazione lato *server* che sfruttasse le potenzialità della libreria *PCL*, potessero essere migliorati fino ad ottenere un applicativo, inseribile nel contesto produttivo aziendale, da utilizzare del campo ispettivo. I risultati ottenuti sono stati piuttosto soddisfacenti, e quanto realizzato, anche se definito ancora allo stadio prototipale, una volta applicate le necessarie migliorie e raffinamenti, potrà diventare un'applicazione completa di grande utilità nel contesto sopracitato.

7.1 Prove pratiche di calcolo del volume

Uno degli obiettivi di primaria importanza del progetto, passo finale del processo di elaborazione e meshing di un Point Cloud, era il calcolo del volume dell'oggetto scansionato, che è uno dei parametri di valutazione del processo ispettivo. Per rendere l'applicazione utilizzabile nel contesto produttivo è necessario che il calcolo del volume si discosti il meno possibile dal volume reale dell'oggetto. Vediamo ora nelle tabelle 7.1 e 7.2 i risultati ottenuti con l'applicativo nella sua ultima versione, per poi discuterne la bontà ed analizzare meglio il problema.

7.1.1 Calcolo del volume di un cestino

* **Oggetto**: Cestino cilindrico
* **Volume reale**: 0.5652 m^3

Test	**Volume calcolato** (m^3)	**Errore** (%)
Test1	0.424100625	24.96%
Test2	0.4515751603	20.10%
Test3	0.5139998176	9.06%
Test4	0.4822320579	14.68%
Test5	0.3294179155	41.72%
Test6	0.375825924	33.51%
Test7	0.5680688195	0.51%
Test8	0.487294459	13.78%
Test9	0.432407363	23.49%

Test10	0.4659140104	17.57%

tabella 7.1: Risultati del calcolo del volume di un cestino

Volume medio: 0.453 m^3 **Errore medio**: 19.94%

I risultati del primo set di test, effettuati scansionando un cestino di forma (approssimativamente) cilindrica non sono particolarmente buoni. L'errore medio del calcolo si assesta infatti attorno al 20%, con picchi di errore del 42%. Tali valori sono inaccettabili per un utilizzo produttivo dell'applicativo. La causa principale di tale divergenza è da ricercarsi nella scarsa qualità dei Point Cloud ricostruiti, che soffrivano di evidenti problemi di *drifting*, complice anche la lucidità della superficie metallica del cestino, causa di disturbo al sensore di profondità infrarosso.

7.1.2 Calcolo del volume di una scatola

* **Oggetto**: Scatola
* **Volume reale**: 0.42 m^3

Test	**Volume calcolato** (m^3)	**Errore (%)**
Test1	0.446018735	6.19%
Test2	0.41666931	0.79%
Test3	0.42045269	0.11%
Test4	0.440469797	4.87%
Test5	0.46461499	10.62%
Test6	0.4279854711	1.90%
Test7	0.430752392	2.56%
Test8	0.388152041	7.58%
Test9	0.776337723	84.84%
Test10	0.302679318	27.93%

tabella 7.2: Risultati del calcolo del volume di una scatola

Volume medio: 0.451 m^3 **Errore medio**: 14.74%

I risultati del secondo set di test, effettuati scansionando una scatola di forma parallelepipoidale, sono molto migliori di quanto osservato nei primi test. L'errore medio del calcolo si assesta infatti attorno al 14.7%, anche se la bontà della media è ampiamente sfalsata da un paio di risultati particolarmente erronei, mentre i restanti 8 test riportano un errore inferiore al 10%. I Point Cloud ricostruiti della scatola erano molto più accurati di quelli del cestino, di qui i risultati migliori, anche se le ricostruzioni soffrivano ancora parzialmente di problemi di *ghosting* e *drifting*.

7.1.3 Conclusioni sul calcolo del volume

Analizzando i dati raccolti risulta evidente che l'applicazione non è ancora pronta per un uso affidabile sul campo. L'esito del calcolo è strettamente correlato alla qualità della mesh da valutare, quindi dal Point Cloud del quale si effettua il meshing e,

risalendo alla radice, alla qualità del Point Cloud inizialmente ricostruito.
Per essere utilizzabile in un contesto produttivo l'applicativo deve generare stime il cui errore dev'essere sufficientemente basso (ad es. $\leq 5\%$), e per ottenere tale risultato è necessario migliorare prima di tutto la ricostruzione iniziale dell'oggetto, come già tentato e discusso nella sezione "*JNI e Point Cloud Registration*" (sez. 5.3.1).

7.2 Problemi irrisolti e sviluppi futuri

Il progetto è nato in tempi molto recenti, dopo circa tre mesi di sviluppo sono stati prodotti più di un prototipo soddisfacente, l'ultimo dei quali realizzato dal tirocinante. La natura sperimentale del progetto ha fatto emergere durante la realizzazione una serie di problemi spesso non banali, alcuni dei quali irrisolti, ma anche una serie di idee e funzionalità per migliorare l'applicativo. Riporto di seguito alcuni di entrambe le categorie.

7.2.1 Problemi irrisolti

Artefatti nel Point Cloud

Durante l'acquisizione dei Point Cloud necessari a ricostruire l'oggetto scansionato, capita spesso che l'acquisizione sia disturbata da alcuni "artefatti", cioè gruppi di punti, tipicamente porzioni di piano, che non hanno corrispettivo nel mondo reale, e degradano la qualità della ricostruzione. Ne vediamo un esempio in figura 7.1.

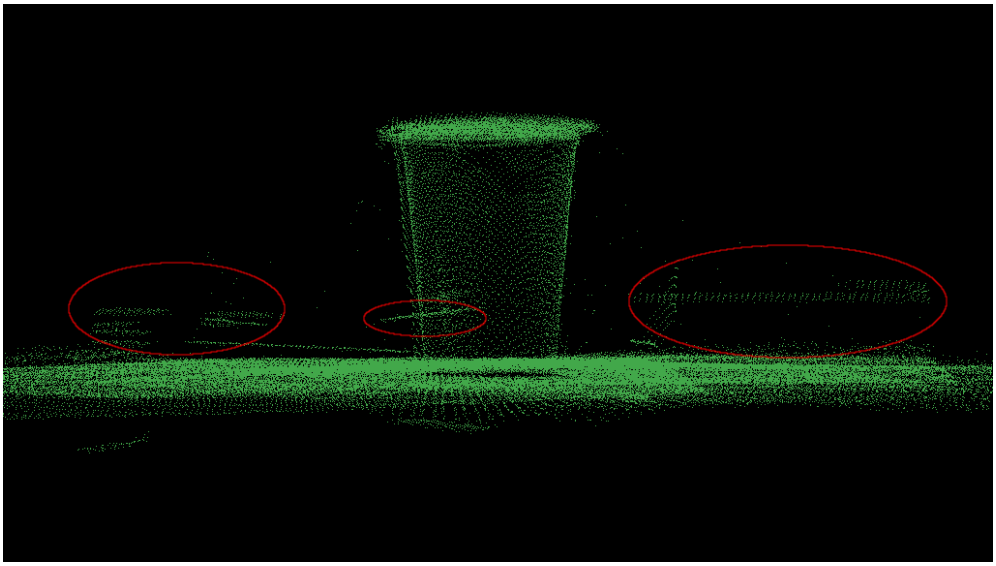


figura 7.1: Esempio di artefatti in un Point Cloud

L'origine di tali artefatti non è ancora chiara, tra le possibili cause ipotizzate:

* Un bug nascosto nell'acquisizione dei punti
* La presenza di riflessi sulle superfici scansionate, che causano problemi al sensore di profondità

* Problemi di esposizione della fotocamera RGB

Solitamente gli artefatti possono essere eliminati con successo attraverso il filtraggio *Cluster extraction*, ma capita talvolta che un artefatto vada ad intersecare i punti dell'oggetto scansionato, rendendo così impossibile estrarre una nuvola di punti che rappresenti precisamente l'oggetto.

Surriscaldamento e consumo della batteria

Il *device Tango* è sottoposto ad un notevole sforzo computazionale durante l'acquisizione dei Point Cloud, a causa dell'utilizzo di più sensori e di pesanti elaborazioni parallele, ed è facilmente predisposto al surriscaldamento (che ne peggiora le prestazioni) e ad un consumo eccessivo della batteria. Non era negli scopi dello stage affrontare tali problematiche, ma in un futuro potrebbe essere necessario risolverle rendendo più performante il processo di aquisizione.

Limiti del device

Durante l'acquisizione dei Point Cloud sono emersi alcuni problemi dovuti ai limiti fisici del *device Tango*, dovuti in particolare all'utilizzo dei sensori di profondità a luce infrarossa, che non funzionano (quindi non è possible acquisire dei punti) nei seguenti casi:

* Superfici riflettenti o molto lucide.
* Superfici di colore molto scuro, tendente al nero

In questi casi il segnale infrarosso viene completamente assorbito o deviato, e non si ottiene quindi nessun feedback. Possiamo vedere gli effetti di tale problema in figura 7.2, in cui si nota come alle parti metalliche della sedia corrisponda l'assenza di punti.

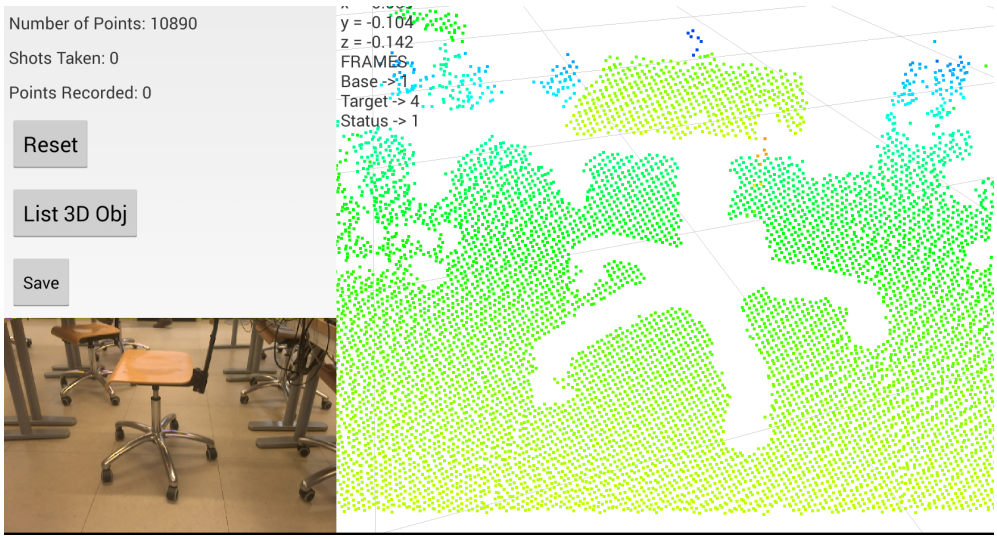


figura 7.2: Esempio di problemi nell'acquisizione di superfici lucide

Non potendo ovviamente agire sulle capacità hardware del dispositivo, l'unica strada possibile per arginare il problema è di implementare un sistema software capace di ricostruire i punti mancanti, compito che esula dagli scopi dello stage.

7.2.2 Sviluppi futuri

Point Cloud Registration

Come già citato nelle sezioni 5.3.1 e 5.3.4, la *Registration* tramite algoritmo ICP di Point Cloud è una funzionalità indispensabile a migliorare la precisione del calcolo del volume, e quindi allo sviluppo futuro dell'applicazione.
Vi sono due strade percorribili per attuare tale miglioria:

* **ICP su *tablet***: opzione preferita, la cui attuazione con GOICP non ha dato i risultati sperati, ma che vale la pena esplorare ulteriormente, testando altri algoritmi ICP, adattando GOICP o implementando un algoritmo ICP ad hoc in Java. Resta da valutare attentamente però l'effettiva capacità del *tablet* di eseguire in tempi accettabili le computazioni richieste

* **ICP lato server**: la libreria *PCL* offre anche un algoritmo ICP, che è stato testato con successo, ma che non è stato possibile importare su *tablet* a causa della complessità dello stesso. Per effettuare ICP su *server* è quindi necessario che il *client* invii non più il Point Cloud ricostruito, ma le singole scansioni una ad una. Tale approccio non è praticabile nel contesto d'uso nel quale dovrà inserirsi l'applicativo, che non prevede una connessione internet stabile, mancanza che renderebbe inaccettabilmente lungo l'intero processo di acquisizione ed elaborazione.

L'implementazione di ICP su *tablet* sembra quindi essere l'alternativa migliore, e senza dubbio è una funzionalità che migliorerebbe significativamente le potenzialità dell'applicativo.

Comparazione con modello 3D esatto

Una funzionalità di indubbia utilità nel campo ispettivo sarebbe la possibilità di confrontare le mesh generate con modelli tridimensionali esatti degli oggetti scansionati, in modo da valutarne la conformità ed evidenziarne eventualì deformità e difetti.
Si pensi ad esempio all'utilizzo in una catena di montaggio per valutare la conformità dei beni prodotti con i modelli di partenza, o al controllo dei danni per beni che siano stati sottoposti a trasporto. L'implementazione di tale funzionalità richiederà prima di tutto una maggiore precisione delle mesh generate, che dovranno anche essere completamente "chiuse", dato che al momento le mesh mancano della parte inferiore, che dev'essere adeguatamente ricostruita.

Integrazione con l'applicazione VIC

L'azienda fornisce ai suoi dipendenti un'applicazione che permette di automatizzare le mansioni ispettive, con l'inserimento nel database aziendale di ogni nuova ispezione o *job*, permettendo all'utente di compilare diversi *form* per definirne scopi e risultati, per fornire così in tempo reale un rapporto dettagliato e standardizzato del lavoro svolto. In questo ambito un integrazione con i risultati ottenuti tramite i *device Tango* potrebbe quindi fornire *report* più dettagliati e velocizzare le operazioni d'ispezione.

Meshing con textures

Il prodotto fornisce delle buone ricostruzioni tridimensionali per quanto riguarda la forma e le dimensioni dell'oggetto; ai fini ispettivi del contesto aziendale, di fatto,

non c'è bisogno d'altro. Ciononostante le ricostruzioni visualizzate su *tablet*, essendo formate da soli triangoli privi di textures, sono spesso di difficile comprensione da parte dell'utenza.
Un primo passo per migliorare tale visualizzazione è stato di permettere l'applicazione di textures standard alla mesh, che perlomeno aiutano a definire meglio la volumetria dell'oggetto. Il passo successivo da implementare per rendere più immediato il riconoscimento dell'oggetto da parte dell'utente sarà di poter applicare textures liberamente scelte, o meglio ancora, di applicare texture che corrispondano ai colori e materiali dell'oggetto scansionato. Questo sviluppo, oltre a migliorare grandemente l'aspetto grafico del sistema, lo renderebbe anche più vendibile.

7.3 Consuntivo finale

Il periodo di stage ha avuto inizio il 12 Luglio 2016 ed è terminato l'8 Settembre 2016, per una durata totale di 320 ore. Seguono in tabella 7.3 le ore preventivate per le attività dello satge, confrontate con le ore realmente impiegate.

Attività	**Preventivo**	**Consuntivo**	**Diff**	**Diff %**
Studio preliminare	40	50	+10	+ 25%
Analisi dei requisiti e progettazione	40	55	+15	+37.5%
Calcolo del poligono e del volume	120	85	- 35	- 29%
Sviluppo dell'applicazione	80	85	+5	+ 6%
Testing e valutazione dei risultati	40	45	+5	+12.5%
Totale	320	320	+0	+0%

tabella 7.3: Distribuzione ore preventivo e consuntivo

Lo ore totali sono state rispettate senza subire variazioni.
La ripartizione interna invece ha subito variazioni consistenti nelle prime fasi del progetto, quali lo studio preliminare e l'analisi dei requisiti, le cui ore sono state sottostimate. L'esplorazione intensiva della documentazione *PCL* si è rivelata invece più dispendiosa di quanto calcolato, ed ah portato ad un maggior numero di ore rivolto all'estensione dello studio di fattibilità e dei requisiti utente.
L'accurato studio iniziale ha però permesso di svolgere con profitto e in tempi minori l'attività di "Calcolo del poligono e del volume", le cui ore preventivate sono state largamente sovrastimate. Le risorse risparmiate in quest'ultima fase sono state però ripartite parzialmente nelle fasi successive, ed hanno permesso di effettuare valutazioni più numerose ed accurate durante la fase di testing.

7.4 Raggiungimento degli obiettivi

Gli obbiettivi riportati inizialmente nel piano di lavoro erano i seguenti:

* **Obbligatori**
  - *ob01*: studio dello stato attuale del progetto;
  - *ob02*: ideazione di una soluzione per il calcolo del poligono rappresentante un oggetto e per il calcolo del relativo volume;
  - *ob03*: implementazione prototipo in grado di calcolare un poligono rappresentante un oggetto e il relativo volume;
  - *ob04*: implementazione app come indicato nella sezione "Struttura applicazione" del documento "Relazione app per Project Tango" fornito dall'azienda;
  - *ob05*: Testing dell'applicazione e valutazione della precisione dei risultati ottenuti.
* **Desiderabili**
  - *de01*: perfezionare calcolo del poligono (cavità nell'oggetto, eliminazione dati di sfondo/rumore);
  - *de02*: perfezionare calcolo del volume (cavità nell'oggetto);
* **Opzionali**
  - *op01*: ottimizzazione della comunicazione con il *server* nel caso risultasse necessaria.

Gli obbiettivi obbligatori definiti sono stati tutti pienamente raggiunti. Per quanto riguarda gli obbiettivi desiderabili invece:

* *de01*: É stato parzialmente soddisfatto, con l'eliminazione degli elementi di disturbo dal Point Cloud. Non è stata implementata invece una tecnica efficiente per il calcolo delle cavità del poligono.
* *de02*: Non raggiunto

L'obbiettivo opzionale *op1* è stato raggiunto con l'implementazione di meccanismi di fallback lato *client* in caso di mancata connessione.

7.4.1 Requisiti

Dagli obiettivi posti nel piano di lavoro e dai casi d'uso prodotti sono stati ricavati i requisiti esposti nel capitolo 4. Seguono le tabelle (7.4, 7.5, 7.6, 7.7) di soddisfacimento di questi ultimi divisi per categorie.

Requisiti Funzionali

Requisito	**Esito**
RFO-1	Soddisfatto
RFO-1.1	Soddisfatto
RFO-1.2	Soddisfatto
RFO-1.2.1	Soddisfatto
RFO-1.3	Soddisfatto
RFO-1.3.1	Soddisfatto
RFO-1.3.2	Soddisfatto
RFO-1.3.3	Soddisfatto
RFD-1.3.4	Soddisfatto
RFD-1.3.5	Soddisfatto
RFO-1.3.6	Soddisfatto
RFZ-1.3.7	Non soddisfatto
RFO-1.4	Soddisfatto
RFO-1.4.1	Soddisfatto
RFO-1.5	Soddisfatto
RFD-1.6	Soddisfatto
RFD-1.7	Soddisfatto
RFO-2	Soddisfatto
RFO-2.1	Soddisfatto
RFO-2.1.1	Soddisfatto
RFO-2.2	Soddisfatto
RFO-2.2.1	Soddisfatto
RFD-2.2.2	Soddisfatto
RFO-2.2.3	Soddisfatto
RFO-2.2.4	Soddisfatto
RFO-2.3	Soddisfatto
RFO-2.4	Soddisfatto
RFO-2.4.1	Soddisfatto
RFD-2.4.2	Soddisfatto
RFO-2.5	Soddisfatto
RFO-2.5.1	Soddisfatto
RFZ-2.5.2	Non soddisfatto
RFO-3	Soddisfatto
RFO-3.1	Soddisfatto
RFO-3.2	Soddisfatto
RFD-4	Soddisfatto
RFD-4.1	Soddisfatto
RFD-4.2	Soddisfatto

tabella 7.4: Soddisfacimento dei requisti funzionali

Requisiti Qualitativi

Requisito	Esito
RQO-1	Soddisfatto
RQO-2	Soddisfatto
RQD-3	Non soddisfatto

tabella 7.5: Soddisfacimento dei requisti qualitativi

Requisiti di Vincolo

Requisito	Esito
RVO-1	Soddisfatto
RVO-2	Soddisfatto
RVO-3	Soddisfatto

tabella 7.6: Soddisfacimento dei requisti di vincolo

Requisiti Prestazionali

Requisito	Esito
RPO-1	Soddisfatto
RPO-2	Soddisfatto
RPD-3	Non soddisfatto
RPD-3.1	Soddisfatto
RPD-4	Soddisfatto
RPD-4.1	Soddisfatto
RPD-5	Soddisfatto
RPD-6	Soddisfatto
RPD-6.1	Soddisfatto

tabella 7.7: Soddisfacimento dei requisti prestazionali

Esito finale

In tabella 7.8 possiamo vedere i dati riassuntivi del soddisfacimento di tutti i requisiti.

Requisiti funzionali			
Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	27	27	100%
Desiderabili	9	9	100%
Opzionali	2	0	0%
Requisiti qualitativi			
Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	2	2	100%
Desiderabili	1	0	0%
Requisiti di vincolo			

Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	3	3	100%
Requisiti prestazionali			
Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	2	2	100%
Desiderabili	7	6	85%
Totale			
Tipologia	**Totali**	**Soddisfatti**	**Percentuale**
Obbligatori	34	34	100%
Desiderabili	17	15	89%
Opzionali	2	0	0%

tabella 7.8: Soddisfacimento totale dei requisti

7.5 Conoscenze acquisite

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

La richiesta di applicazioni in ambito *mobile* è in continuo aumento negli ultimi anni. Per questo l'apprendimento della progettazione e sviluppo delle applicazioni mobili *Android* è certamente una conscenza preziosa ed immancabile per un inserimento nel mondo dell *IT*.
L'approccio ad una tecnologia sperimentale ed ancora di nicchia come *Project Tango* è un notevole vantaggio in ambito occupazionale in quanto gli sviluppatori nell'ambito sono ancora scarsi. 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 capaci di sfruttare le potenzialità che questo tipo di *device* ha da offrire.
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 *IntelliJ*, 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. Il suo utilizzo nello stage è stato possibile solo nell'ultima settimana, grazie al miglior supporto del framework introdotto nelle ultime versioni sperimentali di *Android Studio*. 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.
L'apprendimento delle potenzialità della libreria *PCL* e la gestione dei *Point Cloud* è un'altra competenza non banale, che può dare i suoi frutti in molti campi della *Computer grafica*. Lo studio di *PCL* è stata inoltre un'occasione formativa importante, dalla quale ho imparato ad approcciare in autonomia una libreria vasta e complicata, ed estrapolarne le funzionalità richieste, le metodologie d'utilizzo e le *best practices*.

Il tirocinio è stato quindi un'esperienza formativa interessante e stimolante, dalla quale esco profondamente arricchito nelle conoscenze e nelle capacità produttive.

Glossario

API Con *API* o *Application Programming Interface* si intende ogni insieme di metodi e procedure resi disponibili al programmatore. Di solito raggruppati a formare un set di strumenti specifici per l'esecuzione di un determinato compito. 2, 61

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. 12, 61

Design Pattern Un *Design Pattern* è una soluzione progettuale di comprovata bontà ad un problema ricorrente in un certo contesto. 36, 61

Drifting Il *drifting* è l'errore di calcolo nella posizione di un dispositivo che utilizza il *Motion Tracking*, che ne causa una scorretta stima della posizione.. 4, 61

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*. 4, 61

Fotocamera Fisheye Una Fotocamera *Fish-eye* dispone di un obiettivo grandangolare che abbraccia un angolo di campo di circa 180 gradi, in particolare quella in dotazione nel dispositivo usato è in bianco e nero. 2, 61

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 1.4a. 5, 61

Mesh Una *mesh* o *mesh* poligonale è una collezione di vertici, spigoli e facce che definiscono la forma di un oggetto tridimensionale. 2, 61

PCL Una delle librerie di maggior rilievo nel campo della *Computer Vision*, *Open Source*, scritta in C++, di algoritmi per l'elaborazione di *Point Cloud* tridimensionali. Contiene algoritmi per il filtraggio, la segmentazione, la *registration* e il *meshing* di Point Cloud, per citarne alcuni.
La libreria è ampiamente utilizzata da qualsiasi applicativo debba trattare nuvole di punti, ed oltre ad essere utile ed efficiente è anche ben documentata, con molti

esempi d'utilizzo reperibili online.
Le notevoli potenzialità della libreria sono state utilizzate intensivamente nell'applicativo con lo scopo di isolare i punti appartenenti all'oggetto scansionato dal resto del Point Cloud, ed effettuarne quindi il meshing.. 61

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. 3, 62

Project Tango 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. 2, 62

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 63

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. 6, 62

Acronimi

PCL Point Cloud Library. 9

UML Unified Modeling Language. 13, 62

Bibliografia

[1] *3D Model Viewer Open Source*
URL: https://play.google.com/store/apps/details?id=org.andresoviedo.dddmodel&hl=it (cit. a p. 10).

[2] *3D viewer*
URL: https://play.google.com/store/apps/details?id=com.pcvirt.Viewer3D&hl=it (cit. a p. 10).

[3] *Apache HTTP*
URL: https://developer.android.com/reference/org/apache/http/package-summary.html (cit. a p. 10).

[4] *Downsampling a PointCloud using a VoxelGrid filter*
URL: http://pointclouds.org/documentation/tutorials/voxel_grid.php/voxelgrid (cit. alle pp. 10, 45).

[5] *Euclidean Cluster Extraction*
URL: http://pointclouds.org/documentation/tutorials/cluster_extraction.php/cluster-extraction (cit. alle pp. 10, 45).

[6] *Fast triangulation of unordered point clouds*
URL: http://pointclouds.org/documentation/tutorials/greedy_projection.php/greedy-triangulation (cit. alle pp. 10, 46).

[7] *Filtering a PointCloud using a PassThrough filter*
URL: http://pointclouds.org/documentation/tutorials/passthrough.php/passthrough (cit. alle pp. 10, 44).

[8] *Fitting trimmed B-splines to unordered point clouds*
URL: http://pointclouds.org/documentation/tutorials/bspline_fitting.php/bspline-fitting (cit. a p. 10).

[9] *OkHTTP*
URL: http://square.github.io/okhttp/ (cit. a p. 10).

[10] *Plane model segmentation*
URL: http://pointclouds.org/documentation/tutorials/planar_segmentation.php/planar-segmentation (cit. alle pp. 10, 44).

[11] *Reading Point Cloud data from PCD files*
URL: http://pointclouds.org/documentation/tutorials/reading_pcd.php/reading-pcd (cit. a p. 9).

[12] *Removing outliers using a StatisticalOutlierRemoval filter*
URL: http://pointclouds.org/documentation/tutorials/statistical_outlier.php/statistical-outlier-removal (cit. alle pp. 10, 43).

[13] *The PCD file format*
URL: http://pointclouds.org/documentation/tutorials/pcd_file_format.php/pcd-file-format (cit. a p. 9).

[14] *Writing Point Cloud data to PCD files*
URL: http://pointclouds.org/documentation/tutorials/writing_pcd.php/writing-pcd (cit. a p. 10).

[15] Cha Zhang e Tsuhan Chen
«EFFICIENT FEATURE EXTRACTION FOR 2D/3D OBJECTS IN MESH REPRESENTATION»
in: (2001)
URL: http://research.microsoft.com/en-us/um/people/chazhang/publications/icip01_ChaZhang.pdf (cit. alle pp. 10, 47).