ISSN/0393-3016

M. GUARNERI, S. DI FRISCHIA, M. NUVOLI

Dipartimento Fusione e Tecnologie per la Sicurezza Nucleare
Divisione Tecnologie Fisiche per la sicurezza e la salute
Laboratorio di Diagnostica e Metrologia
Centro Ricerche Frascati, Roma

COBRAKIN: SVILUPPO DI UN SENSORE PER IL MONITORAGGIO ALL'INTERNO DEI MUSEI

RT/2017/31/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

M. GUARNERI, S. DI FRISCHIA, M. NUVOLI

Dipartimento Fusione e Tecnologie per la Sicurezza Nucleare
Divisione Tecnologie Fisiche per la sicurezza e la salute
Laboratorio di Diagnostica e Metrologia
Centro Ricerche Frascati, Roma

COBRAKIN: SVILUPPO DI UN SENSORE PER IL MONITORAGGIO ALL'INTERNO DEI MUSEI

RT/2017/31/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

COBRAKIN: SVILUPPO DI UN SENSORE PER IL MONITORAGGIO ALL'INTERNO DEI MUSEI

M. Guarneri, S. Di Frischia, M. Nuvoli

Riassunto

Questo documento descrive la progettazione e la realizzazione di COBRAKIN, un sensore sviluppato all'interno del Progetto di Ricerca COBRA, che ha lo scopo di acquisire in tempo reale una visione 3D a 360°. Questa caratteristica può essere sfruttata per il monitoraggio e la sorveglianza di ambienti interni (nel nostro caso spazi museali) in maniera innovativa rispetto agli attuali dispositivi di sicurezza.
COBRAKIN è formato da 4 Microsoft Kinect disposte ad angolo retto l'una con l'altra e collegate a due mini-PC (2 Kinect ad un PC-SERVER e 2 Kinect ad un PC-CLIENT) che comunicano tra loro per mezzo di una connessione di rete e di un protocollo sviluppato ad hoc.
I sensori infrarossi contenuti nelle Kinect, abbinati alle telecamere HD, permettono di scansionare in tempo reale l'ambiente circostante in tre dimensioni grazie al processo di misurazione della profondità detto "a luce strutturata".
Il software utilizzato per lo sviluppo dell'interfaccia utente e degli algoritmi di computer vision si basa sul linguaggio di programmazione Processing e su ulteriori tool e librerie open-source. Sono stati implementati diversi algoritmi che introducono caratteristiche utili ad un dispositivo per la video-sorveglianza. Tra questi il Depth Threshold per impostare una distanza di sicurezza rispetto ad un target, il Frame Compare per rilevare esclusivamente figure in movimento nella scena, e il Background Subtraction per determinare cambiamenti rispetto alla scena iniziale.

Parole chiave: sicurezza, video-sorveglianza, visione 3D, luce strutturata, Kinect, beni culturali, computer vision, software, grafica, algoritmi

Abstract

This document describes the design and development of COBRAKIN, a device conceived within the COBRA Research Project, whose purpose is acquiring in real time a full 3D vision in 360°. This feature can be exploited for the monitoring and the surveillance of indoor environments (in our case museum spaces) in an innovative way compared to the actual security devices.
COBRAKIN is designed with 4 Microsoft Kinect set at 90 degree one with the other, and connected to two mini-PC (2 Kinects connected to a PC-SERVER and 2 Kinects to a PC-CLIENT) that communicate through a network connection and a network protocol developed for this aim.
The infrared devices inside the Kinects, coupled to the HD cameras, allow to scan in real time the surrounding environment in three dimensions, thanks to a process of depth measure called "structured light".
The software used for the development of the user interface application and for the computer vision algorithms is based on the Processing programming language, and on further open-source tools and libraries. Several algorithms have been implemented to introduce useful features for the video surveillance. For example, the Depth Threshold to set a security distance against a target, the Frame Compare to detect only the subjects in movement inside the scene, and the Background Subtraction to detect changes compared to the initial scene.

Keywords: *security, video surveillance, 3D vision, structured light, Kinect, cultural heritage, computer vision, software, graphics, algorithms*

INDICE

1 Introduzione

1.1 Progetto COBRA

CO.B.RA è un Progetto di Ricerca di responsabilità ENEA e finanziato dalla Regione Lazio, il cui scopo principale è lo Sviluppo e la diffusione di metodi, tecnologie e strumenti avanzati per la **COnservazione dei Beni culturali, basati sull'applicazione di Radiazioni e di tecnologie Abilitanti**.

Per l'ENEA, in particolare, le Divisioni coinvolte sono: **FSN-TECFIS** presso il Centro Ricerche di Frascati; **SSPT-USER-SITEC** presso il Centro Ricerche di Casaccia; **DTE-ICT** presso i Centri Ricerche di Frascati e Casaccia; **STUDI** presso la sede centrale di Roma.

Figura 1 - Logo del Progetto COBRA

Le finalità generali del progetto sono la diffusione ed il trasferimento alle PMI delle competenze ENEA e di strumenti avanzati di diagnostica e di indagine, atti alla qualificazione dei materiali e all'identificazione dei trattamenti innovativi, per proteggere e conservare il Patrimonio Culturale. Il trasferimento tecnologico viene agevolato garantendo l'accesso delle imprese ai laboratori ENEA, rendendo così disponibili le infrastrutture tecniche di eccellenza presenti nei laboratori dei centri di Ricerca di Frascati e Casaccia, e gli spazi dedicati ai dimostratori.

Le attività del progetto COBRA si articolano in 4 Work-Package così definiti:

- WP1 – Realizzazione del sistema informativo
- WP2 – Potenziamento degli assets di Ricerca e Sviluppo
- WP3 – Diffusione e trasferimento dei risultati
- WP4 – Project Management

Nel dettaglio, l'attività di ricerca e sviluppo che è oggetto di questo rapporto tecnico è stata compiuta all'interno del WP2, nel gruppo di Attività n.°5 ovvero: Sviluppo di Dimostratori di sistemi diagnostici.

Avendo finalità di trasferimento tecnologico infatti, il progetto prevede la realizzazione di una serie di Dimostratori tecnologici per dare evidenza, agli utenti finali e in particolare alle PMI, della capacità innovativa, della applicabilità e dell'usabilità di una serie di strumenti: i cosiddetti Dimostratori. I Dimostratori di COBRA vengono presentati anche come prototipi o ambienti di sperimentazione a supporto delle PMI in specifici interventi.

1.2 Perché COBRAKIN?

Il sensore COBRAKIN, nascendo dunque all'interno del Progetto COBRA, è stato progettato come un oggetto low-cost applicabile al campo dei Beni Culturali con la specifica di essere facilmente fruibile da eventuali imprese interessate al suo utilizzo. Ricordiamo infatti che il trasferimento tecnologico è uno degli obiettivi principali di questo Progetto di Ricerca.

Lo scopo di COBRAKIN è quello di acquisire in tempo reale una visione 3D a 360°, che può essere sfruttata per il monitoraggio e la sorveglianza di ambienti interni (nel nostro caso, ad esempio, spazi museali) in maniera innovativa rispetto agli attuali dispositivi di sicurezza presenti nella maggior parte dei musei.

Inoltre, se l'ambito d'uso principale di COBRAKIN resta focalizzato sulla sicurezza e la video-sorveglianza, la sua versatilità e la semplicità di trasporto e utilizzo possono farne anche un'alternativa low-cost allo scanning 3D di ambienti particolarmente scomodi o difficili da raggiungere, come ad esempio sotterranei o catacombe.

Il nome COBRAKIN è formato dalle parole COBRA, il nome del Progetto di Ricerca, e KIN che sta per Kinect, il sensore di proprietà della Microsoft commercializzato originariamente per offrire una nuova esperienza di gioco nei videogame tramite il riconoscimento dei movimenti del corpo umano. Il sistema COBRAKIN è infatti composto da 4 Kinect poste a 90° l'una dall'altra, che, attraverso la combinazione di sorgenti e detector infrarossi e una fotocamera HD, sono in grado di ricostruire nuvole di punti 3D colorate, generando in questo modo video real-time in 3 dimensioni con una panoramica a 360° dell'ambiente circostante. La descrizione dettagliata della strumentazione e del suo funzionamento è trattata nei paragrafi seguenti.

Come detto sopra, l'applicazione del sensore COBRAKIN è rivolta principalmente a sostituire la tecnologia attuale di sorveglianza all'interno di ambienti museali, basata sull'impiego di videocamere 2D. La sola informazione bidimensionale raccolta dai sistemi di sorveglianza presenta infatti delle criticità e delle debolezze di cui potrebbero approfittare eventuali malintenzionati. Facendo un esempio concreto, pensiamo ad uno spazio museale sorvegliato da normali videocamere 2D poste alcuni metri in alto rispetto alla

superficie fruibile dai visitatori. In questo caso, ad un malfattore che volesse danneggiare un'opera esposta, basterebbe indossare un cappello o un indumento che nasconda parte del volto per eludere la sua identificazione da parte dei sistemi di sorveglianza.

Figura 2- Esempi di sorveglianza 2D in musei durante attacchi terroristici.
Jewish Museum, Bruxelles, maggio 2014 (sinistra) e Museo del Bardo, Tunisi, marzo 2015 (destra)

Restando in tema sicurezza, in aggiunta ai sistemi video di sorveglianza, è diffuso quasi ovunque nei musei l'utilizzo di sensori di prossimità collocati ad una certa distanza dall'opera da proteggere, pronti a far scattare un allarme nel caso in cui un visitatore oltrepassi la soglia di sicurezza (Perimeter Detection).

L'installazione di COBRAKIN in uno spazio di questo tipo, sarebbe in grado di sopperire contemporaneamente sia alla potenziale debolezza dei sistemi di videocamera a due dimensioni, sia all'installazione dei sensori di prossimità a protezione delle opere esposte.

Per quanto riguarda i dispositivi di allarme a soglia infatti, i sensori ad infrarosso sfruttati da COBRAKIN possono calcolare la distanza in tempo reale di un soggetto rispetto a un target specifico. Impostando a piacere una soglia di sicurezza, il sistema è in grado di riconoscere immediatamente un soggetto che oltrepassi anche solo parzialmente la soglia di sicurezza, avvisando prontamente l'operatore della security.

Ancora più importante, con il miglioramento nell'acquisizione di informazioni da immagini 2D a una visione 3D in real-time fornita da COBRAKIN, possiamo introdurre alcune nuove caratteristiche come il riconoscimento e l'analisi di dati morfologici (ad esempio il viso o la corporatura di una persona) in modo tale da aggiungere nuovi gradi di libertà per il rilevamento automatico o semiautomatico di soggetti all'interno di ambienti chiusi. Tornando all'esempio precedente, nascondere il proprio volto da un cappello o da un altro indumento, non impedirà al sistema di acquisire (ed eventualmente salvare o confrontare) le informazioni morfologiche di un potenziale malintenzionato. Anche se apparentemente nascosto in direzione del sensore COBRAKIN, gli angoli di proiezione orizzontale e verticale offerti dai proiettori infrarossi della Kinect permettono infatti l'acquisizione del soggetto di interesse.

Va ricordato che esistono già sul mercato videocamere e sistemi di sorveglianza 3D che applicano algoritmi di analisi morfologica e riconoscimento facciale, tuttavia COBRAKIN si pone come una valida alternativa sia in termini di costo contenuto, sia in termini di portabilità, versatilità e personalizzazione da parte dell'utente.

2 Materiali

2.1 Sensore Kinect

Come accennato nel paragrafo precedente, COBRAKIN sfrutta la visione in tre dimensioni di 4 Kinect che raccolgono immagini in tempo reale. La scelta di utilizzare questo tipo di sensore è legata a diversi fattori che vanno dalle caratteristiche ottiche del device, alla risposta in tempo reale e, non ultimo, all'ottimo rapporto qualità-prezzo che lo rende adatto ad un eventuale trasferimento tecnologico al mondo delle imprese e dei beni culturali.

Microsoft Kinect (dal greco kinēsis = movimento) è una linea di accessori dedicata al Motion Detection sviluppata da Microsoft per le console videoludiche Xbox, e successivamente adattata anche ai PC Windows. Basata su una periferica in stile webcam, la Kinect consente agli utenti di controllare e interagire con la propria console/computer senza la necessità di un controller di gioco, attraverso un'interfaccia utente naturale usando gesti e comandi vocali. La versione utilizzata in COBRAKIN è la Kinect V2, rilasciata originariamente nel novembre 2013, che contiene significativi miglioramenti rispetto alla versione precedente. Ogni Kinect è dotata di 4 sensori principali:

- una telecamera RGB in alta definizione
- un array di microfoni
- un emettitore laser infrarosso
- un ricevitore(telecamera) laser infrarosso

Figura 3 - Microsoft Kinect V2 per Xbox One

Una lista completa dei requisiti della Kinect V2, confrontati con la versione precedente, può essere esaminata nella figura seguente:

Feature	Kinect for Windows 1	Kinect for Windows 2
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	~4.5 M	~4.5 M
Min Depth Distance	40 cm in near mode	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	yes	no
Skeleton Joints Defined	20 joints	26 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Supported OS	Win 7, Win 8	Win 8

Figura 4 - Elenco delle caratteristiche della Kinect v1 e v2

Come si nota dalla tabella, la versione V2 della Kinect ha una risoluzione di tre volte maggiore della precedente, un campo visivo più largo del 60% e può monitorare fino a 6 "scheletri" contemporaneamente. Può inoltre rilevare la frequenza cardiaca, l'espressione del viso, la posizione e l'orientamento di 26 punti (joints) individuali del corpo, il peso messo su ogni arto, la velocità dei movimenti e i gesti degli utenti.

La tecnologia di visione 3D della Kinect sfrutta un processo di misurazione della profondità con una telecamera chiamato a **luce strutturata**. Questo metodo ricade in un ampio gruppo di tecnologie che si basano sulla triangolazione, ovvero sul processo usato dai sistemi di visione stereo che non fanno altro che riprodurre il meccanismo della visione umana, misurando la disparità nel "tempo di volo" della luce da un oggetto ad una coppia di sensori. Non avendo due telecamere per eseguire una triangolazione classica, la Kinect esegue una triangolazione tra una sorgente laser nel vicino infrarosso e la corrispondente telecamera a infrarossi.

Il metodo di scansione 3D della luce strutturata usa la proiezione di un pattern noto (tipicamente righe orizzontali o verticali) sulla scena da acquisire; il modo in cui l'immagine proiettata si deforma colpendo un oggetto, permette al sistema di visione di calcolare la profondità dell'oggetto colpito. Nella Kinect questo calcolo è eseguito dalla telecamera ad infrarossi che lavora a tempo-di-volo (time-of-flight camera), misurando cioè il tempo che il segnale laser impiega per colpire ogni punto della scena e tornare indietro. Nel nostro caso, il pattern usato è un insieme di punti pseudo-random proiettati dalla sorgente, e

successivamente registrati dal detector ad infrarossi e confrontati con il pattern noto. Ogni discrepanza è riconosciuta come una variazione della superficie, in modo tale che la distanza è calcolata come più vicina o più lontana in base al tempo di volo. La time-of-flight camera della Kinect è progettata per leggere 2 Gigabit di dati al secondo.

Questo approccio non è esente da problematiche, che possono essere riassunte in tre punti fondamentali:

- la lunghezza d'onda di emissione dev'essere costante
- la luce con cui è illuminata la scena può causare incongruenze
- la distanza è limitata dalla potenza della sorgente

Per risolvere il primo problema la Kinect è dotata di un piccolo heater/cooler che mantiene costante la temperatura del diodo laser; la lunghezza d'onda infatti può essere compromessa da variazioni di temperatura e alimentazione. Il secondo problema è stato parzialmente mitigato dall'applicazione di un filtro IR a 830nm, ma va riconosciuto che la luce del sole in ambienti all'aperto rischia di compromettere l'acquisizione di una scena. E' per questo che useremo COBRAKIN in ambienti chiusi e non all'esterno. Per quanto riguarda la terza questione, la potenza del laser è limitata dalla necessità di essere eye-safe. Attualmente la Kinect V2 può arrivare ad acquisire correttamente un oggetto a circa 4.5 metri di distanza.

Riassumendo, combinando la tecnologia della time-of-flight camera e il metodo della luce strutturata è possibile acquisire l'intera scena con una singola sorgente laser fissa, in opposizione ad esempio ad uno scanning puntuale utilizzato dai sistemi LIDAR. A tale acquisizione, codificata in una struttura dati che riproduce una nuvola di punti 3D (consultare il paragrafo sulle strutture dati) viene aggiunta l'informazione fondamentale del colore rilevata dalla fotocamera RGB. Grazie alla sovrapposizione delle immagini di profondità e colore operate dalla Kinect, riusciamo così ad avere in tempo reale un video a colori in 3 dimensioni della scena scansionata, con la possibilità di identificare e tracciare i soggetti presenti all'interno di essa.

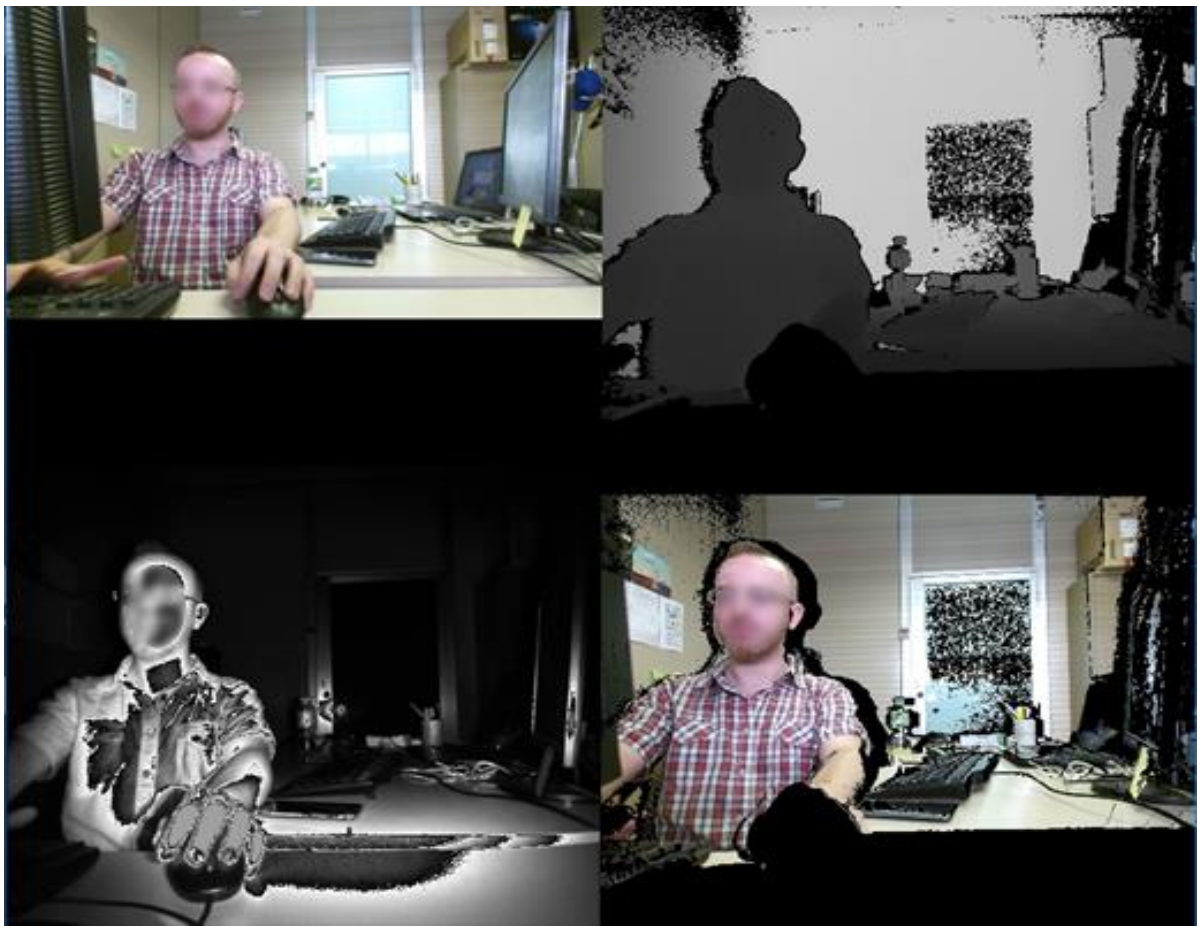

Figura 5 - Esempio di immagini acquisite dai sensori Kinect. Camera RGB HD (in alto a sinistra), Depth Camera (in alto a destra), Depth Camera con Colore (in basso a destra), Camera a Infrarossi (in basso a sinistra)

2.2 Hardware e strumenti software

Le 4 Kinect utilizzate nel sensore COBRAKIN sono collegate a 2 PC tramite degli adattatori per PC Windows rilasciati dalla Microsoft. Ogni adattatore è fornito di un cavo che connette la Kinect con il PC tramite un'entrata USB 3.0 e di un altro che funge da alimentazione, dato che ognuna delle 4 Kinect necessita di un'alimentazione propria.

Poiché avere 4 Kinect collegate contemporaneamente allo stesso PC comporta un traffico enorme sul bus USB (ogni Kinect può arrivare a consumare 2 GB – 2.5 GB di banda al secondo), con una ripercussione negativa sulle prestazioni di calcolo e grafiche dell'acquisizione video, si è deciso di connettere 2 Kinect ad un primo PC che chiameremo PC-SERVER, e altre 2 Kinect ad un secondo PC che chiameremo PC-CLIENT. I due computer sono connessi direttamente tramite un cavo di rete LAN, affinchè il PC-CLIENT invii i dati video registrati dalle sue 2 Kinect al PC-SERVER che si occupa di eseguire un merge totale delle scene 3D di tutte e 4 le Kinect. Maggiori dettagli sul funzionamento del software saranno esposti nei paragrafi seguenti.

La scelta dei 2 PC ha dovuto naturalmente tenere conto, oltre che dei loro requisiti, anche della loro dimensione ridotta, visto che uno degli obiettivi iniziali di COBRAKIN era la facilità di uso e di trasporto in ambienti differenti.

Per il PC-SERVER è stato scelto un **ROG GR8 di ASUSTeK** con le seguenti caratteristiche principali:

- Intel® Core™ i7 4510U Processor 2GHz / 2.60 GHz
- NVIDIA® GeForce GTX750Ti 2GB
- RAM 8 GB
- Hard Disk 2.5" 256GB SATA 6Gb/s SSD
- Dimensioni 6 x 24.5 x 23.8 cm (WxDxH)

Per il PC-CLIENT la scelta è ricaduta su un **GIGABYTE GB-BXi7G3-760 di INTEL** con le seguenti caratteristiche principali:

- Intel® Core™ i7-4710HQ 2.5GHz / 3.50 GHz
- NVIDIA GeForce GTX 760
- RAM 8 GB
- Hard Disk 2.5"
- Dimensioni 5.9 x12.8x11.5 cm

Entrambi i PC girano con un sistema operativo Microsoft Windows a 64 bit. Il PC-SERVER con Windows 10 mentre il PC-CLIENT usa Windows 8.1.

Per quanto riguarda il software da utilizzare nel programma di gestione dell'acquisizione dell'ambiente 3D e dell'interfaccia utente, l'orientamento è stato quello di utilizzare strumenti e tool open-source. I vantaggi nell'uso di software libero sono da ricercare nello sviluppo costante affidato ad una vasta comunità, nella riduzione dei costi di sviluppo e produzione, e nella possibilità che il progetto sia migliorato e debuggato dalla comunità stessa.

Figura 6 - Foto (non in scala) dei due PC utilizzati in COBRAKIN. ROG di ASUSTeK (a sinistra) e GIGABYTE di INTEL (a destra)

Il ricorso al software open-source ha reso necessaria la ricerca di librerie alternative a quelle rilasciate dalla Microsoft per lo sviluppo di applicazioni basate sulla Kinect (Kinect SDK). Esistono tuttavia molte librerie open-source che offrono allo sviluppatore le medesime caratteristiche con un'interfaccia di alto livello per accedere alle funzionalità offerte dal device. Tra l'altro, una delle limitazioni dell'ambiente di sviluppo ufficiale Microsoft era l'impossibilità di sfruttare appieno l'utilizzo di più Kinect contemporaneamente, caratteristica fondamentale di COBRAKIN.

Il linguaggio di programmazione usato per lo sviluppo del software di gestione di COBRAKIN è **Processing**. Processing è un linguaggio open-source fornito di un proprio IDE (Ambiente di Sviluppo Integrato) concepito originariamente per realizzare applicazioni con un immediato riscontro grafico (giochi, animazioni, contenuti interattivi, ecc…). Processing eredità l'intera sintassi, i comandi, le librerie e il paradigma di programmazione orientata agli oggetti dal linguaggio **Java** e in più fornisce numerose funzioni di alto livello per gestire in modo semplice gli aspetti grafici e multimediali.

Un'applicazione standard scritta in Processing (chiamata *sketch*, tecnicamente una sotto-classe della classe Java *PApplet*), sarà composta dalle varie classi di oggetti che la compongono, da un'eventuale cartella *Data* in cui vengono salvate le risorse multimediali utili all'applicazione, e da due metodi obbligatori per l'esecuzione finale: il metodo *setup()* e il metodo *draw()*. Nel primo, eseguito una sola volta all'avvio del programma, vengono impostati i parametri principali dell'applicazione che rimarranno costanti durante l'esecuzione, mentre il secondo, eseguito ciclicamente, conterrà al suo interno il processo per il quale è progettata l'applicazione.

Processing inoltre è fornito di alcune librerie che consentono di interfacciarsi in maniera semplice, attraverso metodi intuitivi, con una o più Kinect collegate al PC. Per lo sviluppo di COBRAKIN la decisione è stata quella di affidarsi alla libreria open-source **libfreenect2**. I motivi sono stati la sua relativa semplicità d'uso, la gamma di operazioni e metodi che essa mette a disposizione, e l'assenza di bug nel gestire più di una Kinect contemporaneamente. La libreria libfreenect2 consente inoltre di accedere ai dati acquisiti dalla Kinect, sia sotto forma di classi *PImage* (una classe nativa di Processing concepita per memorizzare immagini), sia come array di tipo *integer* o *float*. Le strutture dati e il processamento realizzato per l'esecuzione di COBRAKIN saranno esposti nel dettaglio nei capitoli seguenti.

Va ricordato che, per un corretta elaborazione grafica dei dati inviati dalle Kinect, è necessario scaricare anche alcune librerie grafiche disponibili gratuitamente on-line. Tra di esse **OpenGL**, **TurboJPEG**, **GLFW**, e **CUDA** (o in alternativa Intel Open CL). Utili anche le librerie **OpenNI 2** e **OpenCV** che implementano caratteristiche mancanti in libfreenect2, come il face recognition o lo skeleton tracking.

Infine, per bypassare il controllo delle librerie USB standard di Windows, è obbligatorio installare le librerie **libusbK** e sostituirle nella gestione dispositivi del sistema.

Riassumendo, nella figura seguente è possibile vedere l'architettura hardware di COBRAKIN, che consiste in 4 Kinect, 4 adattatori per PC, 2 mini-PC, un cavo di rete per connettere punto-punto i 2 PC, ed eventuali altre periferiche di controllo (monitor, mouse, tastiere, ecc..).

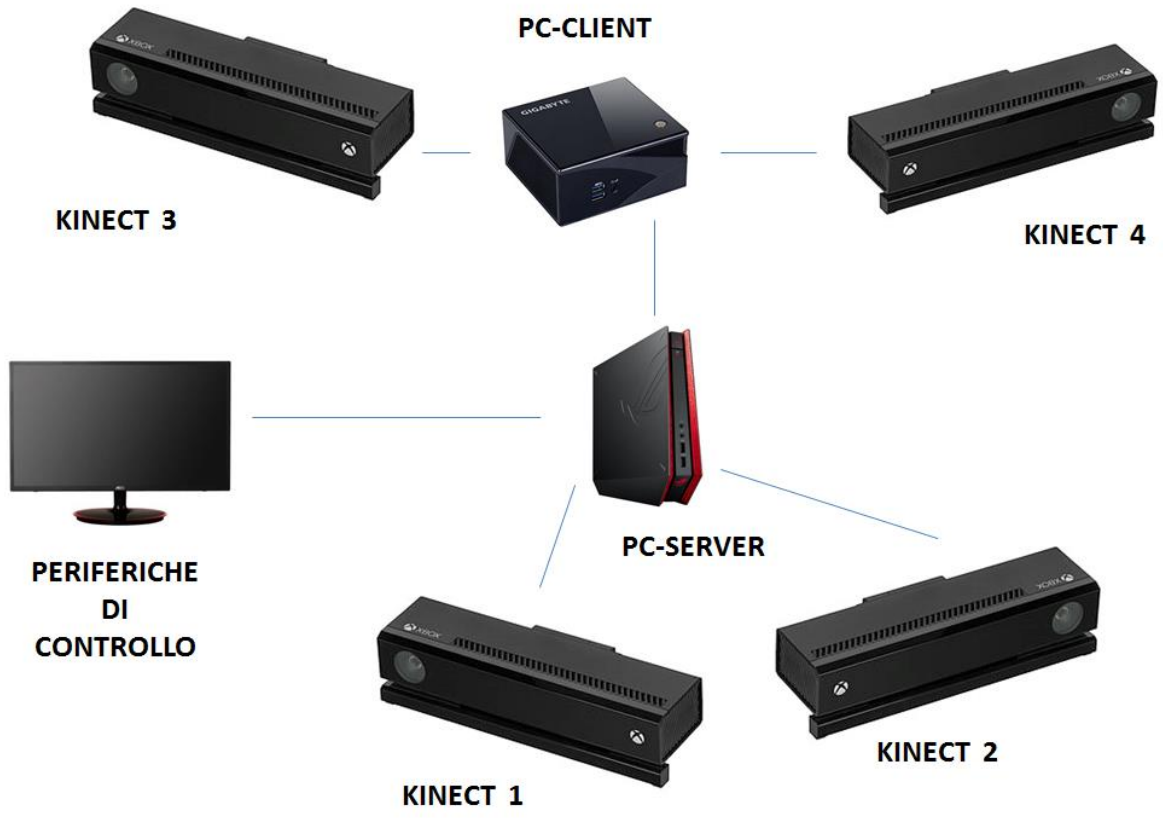

Figura 7 - Schema dei dispositivi hardware che compongono COBRAKIN

3 Metodi

3.1 Architettura software

3.1.1 Struttura delle classi

La necessità di usare 2 differenti PC, uno con la funzionalità di client e un altro di server, ha fatto sì che venissero progettate e sviluppate 2 diverse applicazioni software per COBRAKIN: una per gestire il comportamento del client, chiamata **cobrakinClient**, e un'altra per gestire il server chiamata **cobrakinServerViewer**. Il pattern software architetturale usato quindi in questo progetto è un classico sistema **Client/Server** con la presenza di alcune variazioni rispetto ai modelli canonici.

Innanzitutto, la connessione punto-punto dei due calcolatori permette una comunicazione diretta fra le due applicazioni senza transitare per una rete intermedia (che sia Internet o una LAN locale), migliorando così l'efficienza e la velocità dei dati inviati e ricevuti. Inoltre, in un'architettura classica Client/Server, il client è un sistema di limitata complessità che si occupa solo di offrire un'interfaccia verso il server, richiedendo un servizio o accedendo ad una risorsa che il server mette a disposizione. Nel nostro caso, il client non è una semplice interfaccia, ma acquisisce i dati delle 2 Kinect a lui collegate, e, dopo aver compiuto alcune operazioni di *downsampling*, invia i dati raccolti al server. Allo stesso tempo il server, oltre ad implementare i servizi principali di processamento e gestione dei dati delle 4 Kinect, offre all'utente anche l'interfaccia grafica di controllo e naturalmente la visione real-time della scansione 3D. Va ricordato infine che l'architettura client/server non impedisce di aggiungere in futuro nuovi client al sistema COBRAKIN, senza che sia necessario con ciò modificare le classi e il comportamento dell'applicazione server.

La struttura delle due applicazioni rispecchia un approccio modulare, affinché ogni classe abbia una sua ben precisa funzionalità e al fine di facilitare l'aggiunta di nuovi moduli con l'aumentare delle operazioni e dei servizi offerti da COBRAKIN. Nella programmazione con il linguaggio Processing si sono adottate alcune *best practices* piuttosto diffuse con le applicazioni sviluppate in Java, e più in generale con la programmazione orientata agli oggetti. Le due più importanti sono l'alta coesione e il basso accoppiamento (**High Cohesion – Low Coupling**).

La coesione si riferisce al grado in cui gli elementi presenti all'interno di uno stesso modulo software sono correlati. Essa misura dunque la forza della relazione tra i metodi di una certa classe. Classi con alta coesione hanno metodi che svolgono funzioni in comune ed accedono ad un insieme correlato di dati. L'alta coesione è positiva perché riduce la complessità del sistema, aumenta la sua manutenibilità e favorisce il riuso dei moduli per altri scopi.

L'accoppiamento è il grado di interdipendenza fra moduli software; è una misura di quanto siano connessi e correlati i metodi di moduli e funzioni diverse. Più basso è l'accoppiamento fra i moduli software di un

sistema, più alta sarà la sua coesione e di conseguenza migliore sarà il suo funzionamento. Un basso accoppiamento impedisce che i cambiamenti di un modulo si riflettano a cascata su altri, riduce lo sforzo nella programmazione o nella modifica di un singolo modulo e favorisce il riuso e il testing delle singole classi.

Di seguito è mostrato un diagramma UML delle due applicazioni con una breve spiegazione delle funzionalità delle varie classi software.

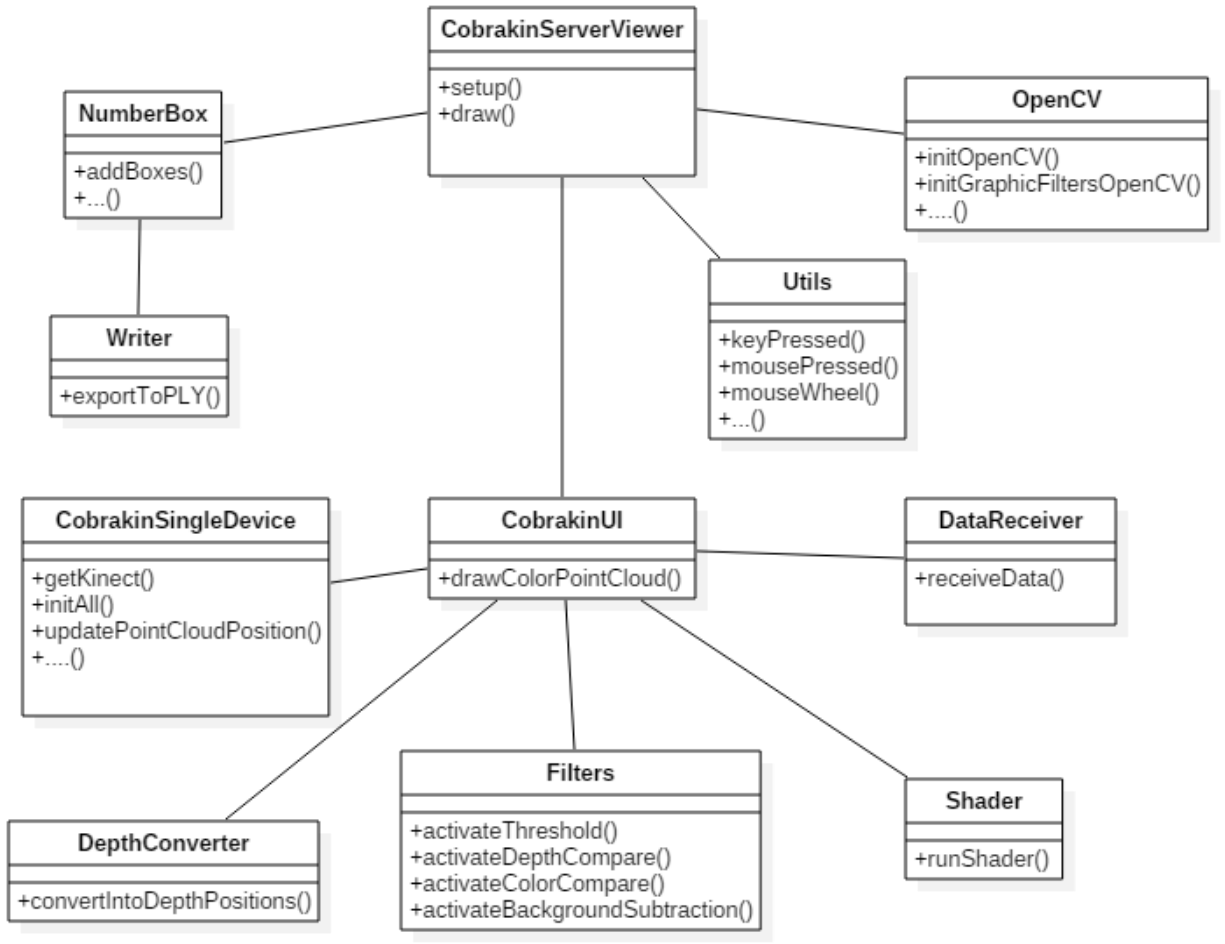

Figura 8 - Diagramma delle classi UML dell'applicazione Server

- **CobrakinServerViewer**: classe main *PApplet* responsabile del setup e dell'esecuzione dell'applicazione
- **NumberBox**: classe responsabile della creazione delle caselle di testo interattive nell'interfaccia grafica
- **Writer**: classe responsabile del salvataggio della nuvola di punti 3D in formato PLY

- **OpenCV**: classe contenente i metodi ereditati e sviluppati a partire dalla libreria OpenCV
- **Utils**: classe responsabile della risposta del sistema con l'interazione dell'utente (tasti, mouse, ecc..)
- **CobrakinUI**: classe principale contenente l'algoritmo di processamento dei dati acquisiti dalle Kinect, e responsabile della rappresentazione grafica finale
- **CobrakinSingleDevice**: classe che rappresenta il singolo *device* Kinect, con i metodi per accedere alle funzionalità base di esso
- **DataReceiver**: classe che implementa il protocollo di comunicazione fra Client e Server (in questo caso riceve i dati dal client)
- **DepthConverter**: classe che converte l'array *integer* di raw depth in array *float* di depth
- **Filters**: classe che implementa i filtri sviluppati per l'applicazione
- **Shader**: classe responsabile della resa grafica finale con l'ausilio degli shader

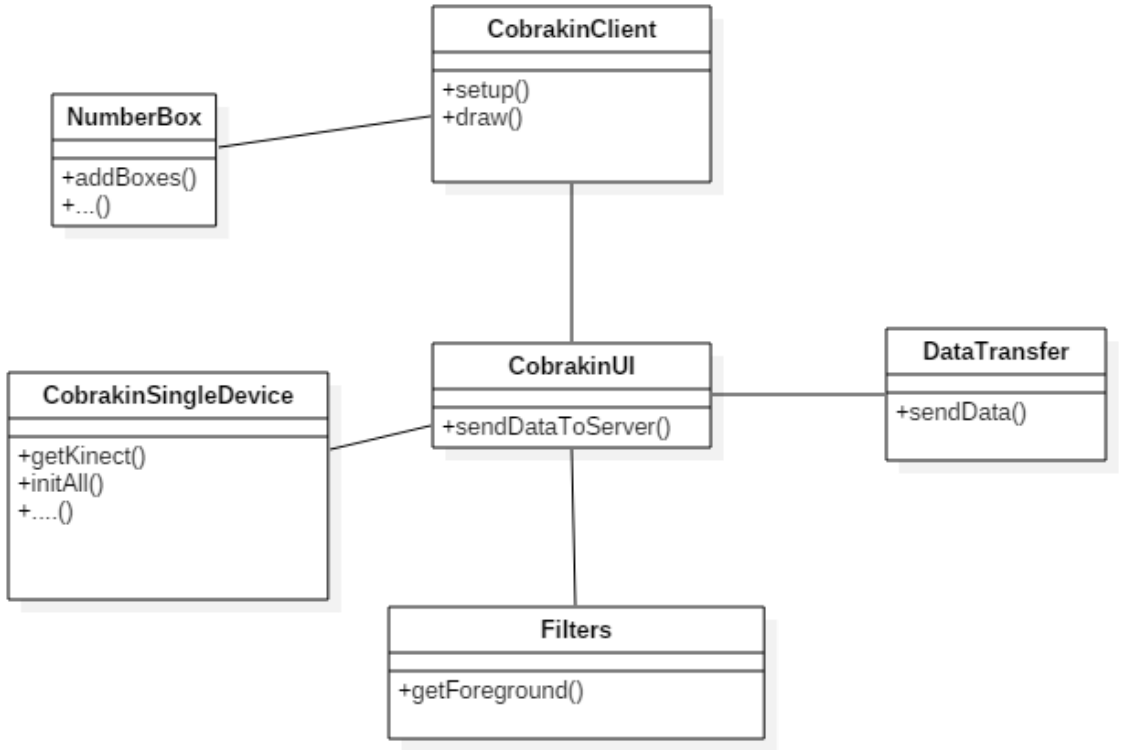

Figura 9 - Diagramma delle classi UML dell'applicazione Client

- **CobrakinClient**: classe main *PApplet* responsabile del setup e dell'esecuzione dell'applicazione
- **NumberBox**: classe responsabile della creazione di box interattivi nell'interfaccia grafica
- **CobrakinUI**: classe principale contenente l'algoritmo di processamento dei dati acquisiti dalle Kinect, e resposabile del loro invio al server

- **CobrakinSingleDevice**: classe che rappresenta il singolo *device* Kinect, con i metodi per accedere alle funzionalità base di esso
- **DataTransfer**: classe che implementa il protocollo di comunicazione fra Client e Server (in questo caso invia i dati al server)
- **Filters**: classe che implementa i filtri di riduzione (*background subtraction*) nel processamento dei dati della Kinect

3.1.2 Interfaccia Utente

COBRAKIN fornisce all'utente finale un'interfaccia semplice e intuitiva per la visualizzazione, la gestione e la modifica della scena 3D ricostruita a partire dalle 4 Kinect. Come descritto nei paragrafi precedenti, l'applicazione che si occupa di offrire l'interfaccia grafica risiede sul PC-SERVER, a cui sono collegate le varie periferiche attraverso le quali l'utente può gestire il sistema. I comandi presenti in COBRAKIN possono essere classificati in 3 categorie:

- Comandi per muovere e/o modificare la **visualizzazione** delle 4 point-cloud
- Comandi per applicare **filtri** di varia natura alla scena 3D
- Comandi di **esportazione** dati

Di seguito è visibile un esempio di screenshot in cui è possibile notare l'interfaccia utente di COBRAKIN:

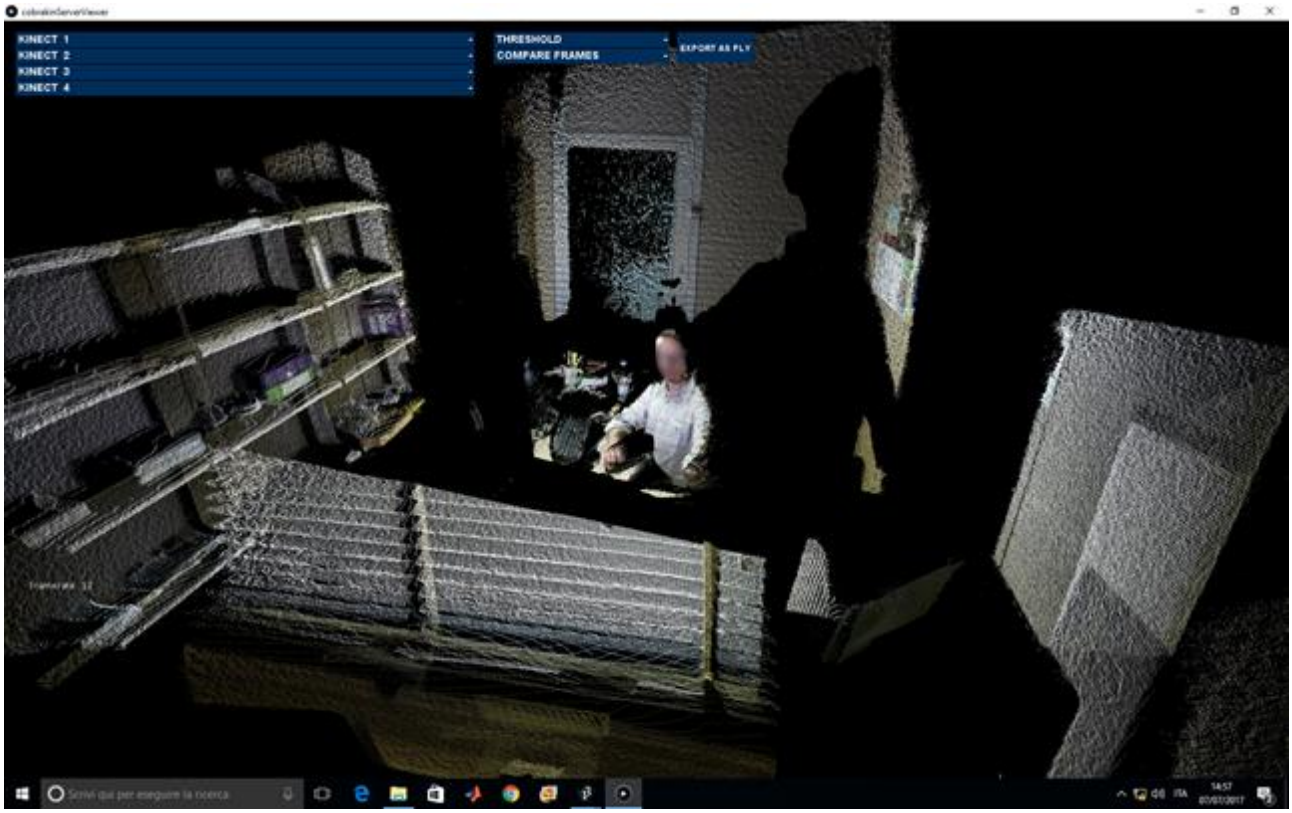

Figura 10 - Esempio interfaccia utente COBRAKIN

In alto a sinistra sono presenti 4 menu per il controllo della posizione, ognuno relativo ad una singola Kinect. Ciascun menu contiene 6 caselle di testo interattive, con le quali modificare la visualizzazione della singola point-cloud. Le prime tre controllano la traslazione lungo i 3 assi dimensionali x,y e z, e i valori da inserire (numeri decimali a 2 cifre) possono essere negativi o positivi a seconda del verso nel quale si desidera traslare la point-cloud. Le ultime tre caselle controllano invece la rotazione intorno ai 3 assi dimensionali x,y e z; i valori da inserire corrispondono al radiante dell'angolo con cui si desidera ruotare la point cloud intorno all'asse selezionato. Inoltre, sotto ogni casella, sono presenti due pulsanti "+" e "-" per incrementare o decrementare la misura di una singola unità. Questa aggiunta si è resa necessaria per facilitare eventuali piccole calibrazioni da parte dell'utente. Un esempio di questo menu è illustrato nella figura seguente.

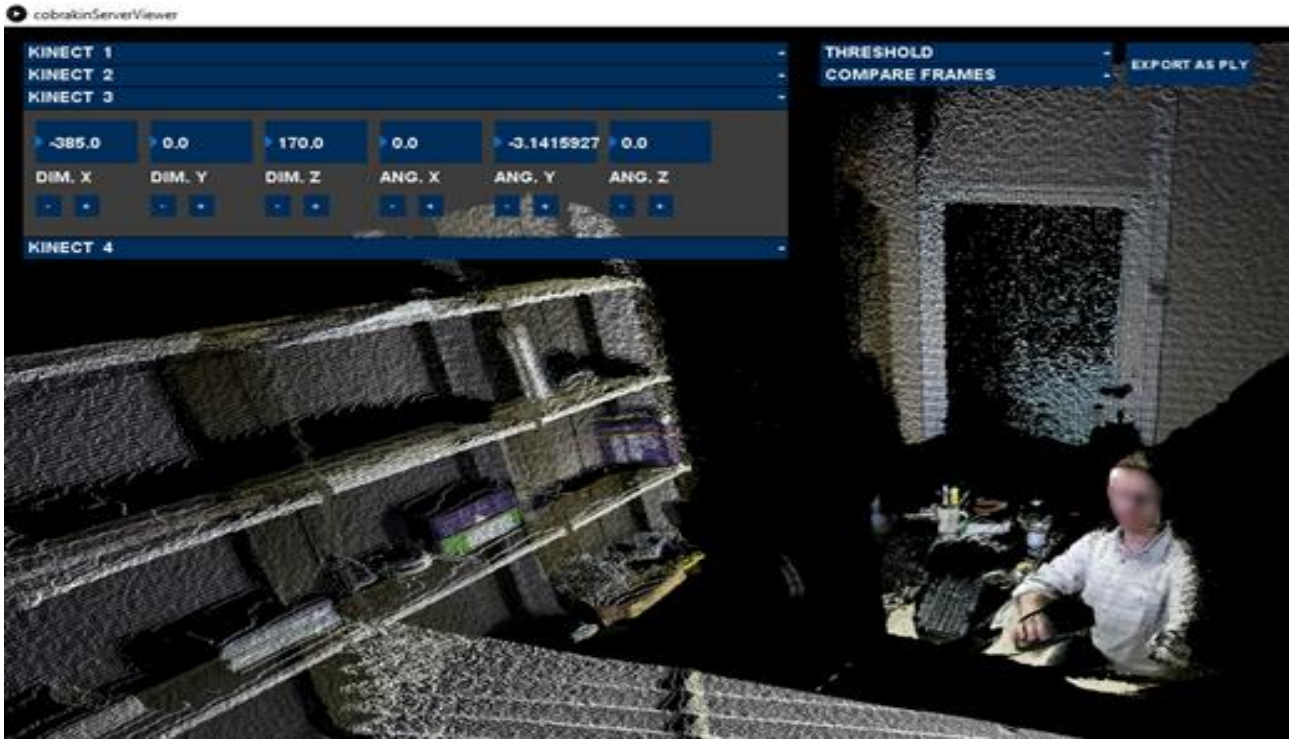

Figura 11 - Particolare del menu di posizionamento delle point-cloud

In aggiunta ai menu di configurazione del posizionamento delle point-cloud, COBRAKIN offre all'utente l'opzione di spostare, zoomare e ruotare in qualsiasi direzione la vista totale della scena 3D per mezzo del mouse o della tastiera. Di seguito è elencato l'elenco di comandi implementati:

- **Drag & Drop del mouse**: spostamento della scena 3D rispetto al punto di vista dell'utente
- **Rotella del mouse**: zoom in o out della scena 3D rispetto al punto di vista dell'utente
- Tasti '**w**' e '**s**': rotazione completa della scena rispetto all'asse orizzontale x
- Tasti '**a**' e '**d**': rotazione completa della scena rispetto all'asse verticale y
- Tasti '**q**' e '**e**': rotazione completa della scena rispetto all'asse ortogonale z

Il secondo gruppo di comandi si trova raggruppato in un menu alla destra dei comandi di posizione, e riguarda l'applicazione di filtri e algoritmi rispetto alla scena 3D. Nella versione attuale di COBRAKIN i filtri implementati sono i seguenti:

- **Depth Threshold**: impostando il valore di una soglia, il sistema esclude tutto ciò che è oltre la distanza della suddetta soglia
- **Frame Compare**: impostando un valore detto di "tolleranza", il sistema misura ogni pixel della scena confrontandolo con il pixel relativo del frame precedente. Se questo valore di differenza eccede la tolleranza, il pixel viene mostrato, altrimenti no. Questa misurazione può essere compiuta alternativamente confrontando i valori di profondità o quelli del colore.

- **Background Subtraction**: il sistema registra per qualche secondo la scena e la salva come background. Dopodiché mostra in real-time solo tutto ciò che è diverso dalla scena salvata.

I dettagli degli algoritmi e del funzionamento delle operazioni fin qui elencate saranno esposti nel capitolo seguente.

L'ultimo comando disponibile accanto ai menu già descritti riguarda la possibilità di esportare con un singolo click l'intera scena 3D, e salvarla in un file di formato PLY (Polygon File Format) per eventuali analisi e processamenti con software terzi.

Un esempio dei menu descritti è visibile nella figura seguente.

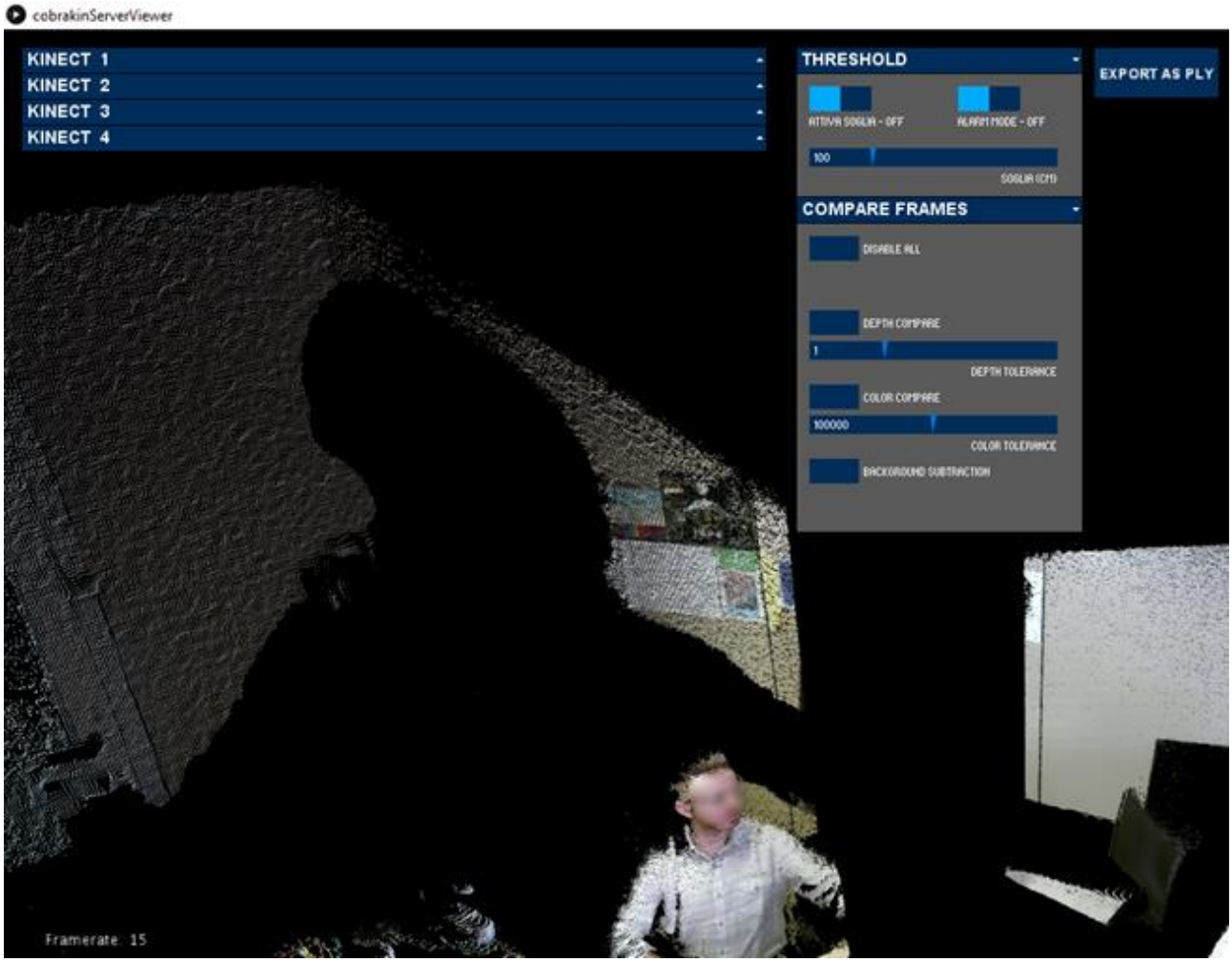

Figura 12 – Particolare del menu di applicazione filtri e algoritmi alla point-cloud

3.1.3 Protocollo di comunicazione client-server

Abbiamo descritto in precedenza l'architettura di COBRAKIN, elencando i dispositivi hardware coinvolti: 2 Kinect collegate ad un PC-CLIENT e 2 Kinect collegate ad un PC-SERVER. Le problematiche principali sorte durante lo sviluppo di questa configurazione sono state le seguenti: con quale protocollo far comunicare i 2 calcolatori? Come far sì che non vengano persi pacchetti durante la comunicazione per conservare una resa grafica ottimale? Come distinguere i dati provenienti da una Kinect piuttosto che da un'altra? Come gestire eventuali errori di comunicazione per evitare il blocco della visualizzazione 3D?

Analizzando prima il formato dei dati, possiamo notare che quelli da inviare appartengono a due categorie: gli interi che rappresentano il valore di profondità (depth) calcolato dai sensori della Kinect, e gli interi che rappresentano il colore associato ad ogni pixel (codificati in RGBA). Ognuno di questi valori sarà contenuto in un array, cioè in un vettore ordinato in cui l'indice di posizione indica il numero del pixel della scena, mentre il valore contenuto rappresenta nel nostro caso la misura della depth o del colore. Avendo una risoluzione della camera IR a 512x424 avremo due array di 217088 elementi. Nella figura seguente possiamo vedere un esempio di come vengono memorizzati i valori all'interno dell'array.

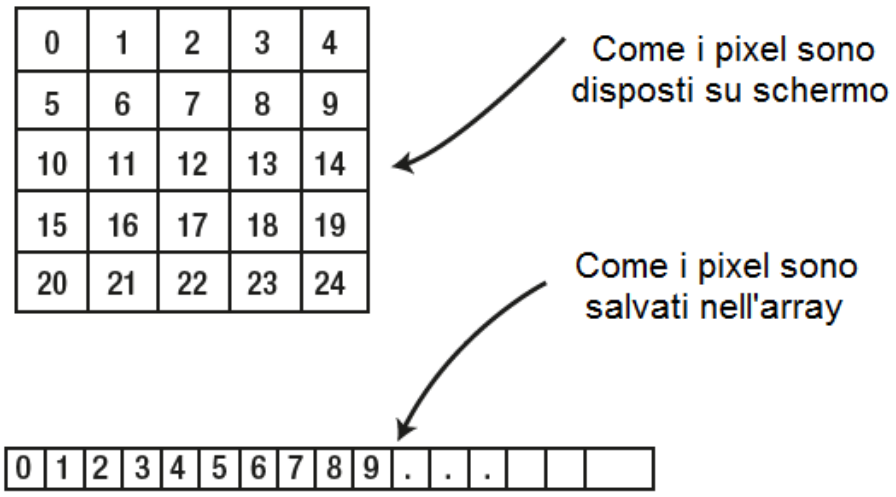

Figura 13 - Disposizione dei valori dei pixel su schermo e nell'array

Ricordando che un valore intero (int) in informatica ha dimensione 4 byte (32 bit), possiamo riassumere così i dati scambiati tra client e server:

- Un array di 217088 int che contiene i valori della depth (**depthData**)
- Un array di 217088 int che contiene i valori di colore (**colorData**)

Scendiamo ora nel dettaglio iniziando dal **PC-CLIENT**. A livello applicativo, è stato implementato un meccanismo di Background Subtraction per rendere più leggero il carico sulla rete ed aumentare il framerate finale di visualizzazione. Il concetto è quello di inviare inizialmente al Server i due array interi di depth e color, e poi (dopo circa 15 loop di esecuzione) inviare solo i pixel che sono cambiati rispetto al frame precedente, ovvero solo gli oggetti e le figure in movimento rispetto al background. In questo modo il carico di dati risulta notevolmente alleggerito, e la visualizzazione finale non risente di una perdita di informazioni. Vediamo nel dettaglio l'algoritmo:

- Se il numero di loop è minore di 15
    - Salva i due array di depth e colore in memoria come background
    - Invia al server il valore dell'indice della Kinect
    - Invia al server l'array completo di depth della relativa Kinect
    - Invia al server l'array completo di colore della relativa Kinect
- Se il numero di loop è maggiore di 15
    - Confronta il colore di ogni pixel del frame attuale (frame n) con quelli del frame precedente (frame n-1)
    - Per ogni pixel che eccede un certo valore di tolleranza:
        - Salva l'indice del pixel
        - Salva il suo valore di depth
        - Salva il suo valore di colore
    - Una volta finito il ciclo, concatena tutti i valori di pixel differenti in un unico array di "foreground"
    - Invia al server il valore dell'indice della Kinect
    - Invia al server la dimensione dell'array di foreground
    - Invia al server l'array di foreground

Spetterà quindi al server, dopo aver ricevuto inizialmente gli array completi di depth e color, processare l'array di foreground e sostituire i pixel modificati (cioè delle figure in movimento nella scena) negli array che saranno visualizzati su schermo.

A livello di trasporto, è stato scelto come protocollo di comunicazione lo **User Datagram Protocol (UDP)**. L'UDP è un protocollo di tipo *connectionless* e *stateless* (a differenza di TCP) che non gestisce il riordino dei pacchetti né la ritrasmissione di quelli persi. Tuttavia, nel nostro caso la sua scelta è stata dettata dalla sua maggiore rapidità e dal tipo di connessione punto-punto tra i due calcolatori che limita al massimo la perdita di pacchetti. UDP inoltre è usato in molti contesti nella trasmissione di informazioni audio-video. I pacchetti in UDP sono chiamati Datagrammi e, nel nostro caso, la dimensione del buffer di comunicazione per l'invio e la ricezione di datagrammi si attesta sui 16000 interi (64000 byte).E' stato perciò previsto un meccanismo di suddivisione dell'array da inviare in pacchetti da 64KB, gestendo eventuali pacchetti finali ottenuti dal resto di tale divisione.

La comunicazione fra i due calcolatori avviene dopo aver instaurato una socket tradizionale su protocollo IP, tramite l'apertura di una porta. Il tipo di socket, essendo utilizzato il protocollo di trasporto UDP, è di tipo **DatagramSocket**.

L'algoritmo a livello di trasporto è il seguente:

- Inizializza la socket impostando come parametri l'indirizzo IP e la porta del Server
- Inizializza il byte-buffer totale moltiplicando per 4 la dimensione dell'array da inviare (int = byte*4)
- Pone la dimensione del buffer del datagramma a 16000 interi. Se il dato da inviare è minore, aggiorna la dimensione con quest'ultimo valore
- Determina il numero di datagrammi da inviare dividendo la dimensione totale del dato da inviare per la dimensione del buffer del datagramma. Se la divisione contiene un resto, esso sarà il numero di byte rimanenti da inviare
- Invia al server un messaggio di stato "not-busy"
- Per ogni datagramma da inviare:
    - Recupera il blocco di byte dall'array da inviare calcolando gli indici esatti
    - Se il server non è impegnato
        - Invia al server un messaggio di stato "busy"
        - Invia al server il blocco dati
- Se il numero di byte rimanenti è maggiore di zero
    - Se il server non è impegnato
        - Invia al server un messaggio di stato "busy"
        - Invia al server il blocco di dati rimanente

Dal punto di vista del **PC-SERVER** la situazione è speculare sia dal punto di vista applicativo, sia da quello di trasporto. Nei primi 15 loop il server riceve l'intero background dal client e lo salva in memoria, in quelli successivi, ricava dall'array di foreground ricevuto gli indici e i valori dei pixel da sostituire, agendo di conseguenza. Questo è l'algoritmo nel dettaglio:

- Se il numero di loop è minore di 15
    - Riceve il valore dell'indice della Kinect
    - Riceve l'array completo di depth della relativa Kinect
    - Riceve l'array completo di colore della relativa Kinect
    - Salva gli array in memoria come "background"
- Se il numero di loop è maggiore di 15
    - Inizializza l'array da visualizzare con il background
    - Riceve il valore dell'indice della Kinect

- Riceve la dimensione dell'array di foreground
- Riceve l'array di foreground
- Esegue un ciclo sull'array di foreground determinando:
  - L'indice del pixel da sostituire
  - La depth del pixel
  - Il colore del pixel
- Sostituisce i pixel di foreground nell'array da visualizzare

Dal punto di vista del protocollo di trasporto, la socket UDP del server sarà in ascolto per ricevere i datagrammi inviati dal client. E' necessario che la socket conosca a priori la dimensione del dato che è in procinto di ricevere, in modo da determinare correttamente la grandezza del buffer e il numero dei blocchi. L'algoritmo lato server è il seguente:

- Inizializza la socket impostando come parametro la porta del Server
- Inizializza il byte-buffer totale moltiplicando per 4 la dimensione dell'array da ricevere (int = byte*4)
- Pone la dimensione del buffer del datagramma a 16000 interi. Se il dato da ricevere è minore, aggiorna la dimensione con quest'ultimo valore
- Determina il numero di datagrammi in ricezione dividendo la dimensione del dato totale da ricevere per la dimensione del buffer del datagramma. Se la divisione contiene un resto, esso sarà il numero di byte rimanenti da ricevere
- Per ogni datagramma da ricevere:
  - Crea in memoria un blocco di byte con la dimensione esatta
  - Invia al client un messaggio di stato "not-busy"
  - Riceve il datagramma
  - Invia al client un messaggio di stato "busy"
  - Determinando gli indici esatti, copia il pacchetto ricevuto nel byte-buffer totale
- Se il numero di byte rimanenti è maggiore di zero
  - Crea in memoria un blocco di byte con la dimensione esatta
  - Invia al client un messaggio di stato "not-busy"
  - Riceve il datagramma
  - Invia al client un messaggio di stato "busy"
  - Copia il pacchetto ricevuto alla fine del byte-buffer totale
- Converte il byte-buffer totale in un array di interi, pronto per essere processato dal sistema

3.2 Algoritmi e strutture dati

In questo paragrafo vengono descritti alcuni algoritmi necessari al corretto funzionamento del sistema COBRAKIN. Il primo algoritmo riguarda la conversione dell'array di valori "depth raw"(int) a "depth" (float), ovvero come ottenere da un dato di profondità "raw" e da parametri intrinseci della camera un insieme di punti (x,y,z) nello spazio 3D. I successivi riguardano filtri e algoritmi tipici della Computer Vision sviluppati per migliorare e aumentare le funzionalità e i casi d'uso di COBRAKIN.

Prima di proseguire, è necessario chiarire come verranno rappresentati i dati 3D acquisiti dalle Kinect. Quando si tratta di elaborare graficamente dati in tre dimensioni, esistono varie rappresentazioni, ognuna con i suoi vantaggi e i suoi punti deboli. Il modello nativo scelto per la Kinect, e di conseguenza usato da COBRAKIN, è quello delle Nuvole di Punti (**Point Clouds**). Una Point-Cloud è una collezione di punti 3D non connessi. Sono chiamate "nuvole" perché al momento della visualizzazione la mancanza di connettività tra i punti le fa apparire come se galleggiassero nello spazio. I tipi più semplici di Point-Cloud contengono solo le informazioni di posizione, ma in realtà ogni punto può contenere ulteriori informazioni addizionali (come il colore nel nostro caso, o l'orientamento normale).

3.2.1 Conversione dell'array depth da int a float

Abbiamo descritto nel paragrafo precedente come le due Kinect collegate al PC-CLIENT inviano i dati acquisiti incapsulandoli in due strutture dati di tipo array, uno per la profondità e uno per il colore. Una volta ricevuti correttamente dal PC-SERVER, questi due array devono essere elaborati da uno Shader per la loro rappresentazione su schermo. Lo **Shader** è un'applicazione usata per i processi grafici più complessi che possiede un alto grado di flessibilità e sfrutta direttamente la GPU (Graphics Processing Unit) del calcolatore, migliorando notevolmente i processi di resa grafica rispetto alle normali pipeline grafiche. Nel nostro caso, COBRAKIN sfrutta per la visualizzazione delle 4 Point-Cloud dei **Vertex Shader**.

Il problema è che lo Shader richiede come parametro in entrata un array di valori float (numeri in virgola mobile a 32 bit) che contenga 3 valori per ogni punto della nuvola: uno per la coordinata x, uno per y, e uno per z. L'array ricevuto dal PC-SERVER invece, contiene un solo valore int per ogni punto dello spazio. Per questo, è necessario applicare una trasformazione in un sistema di coordinate 3D che rispecchi la compatibilità con lo Shader, determinando i valori dalla profondità "raw" e dai parametri intrinseci della camera. Possiamo riassumere i passaggi necessari alla visualizzazione grafica nella figura seguente.

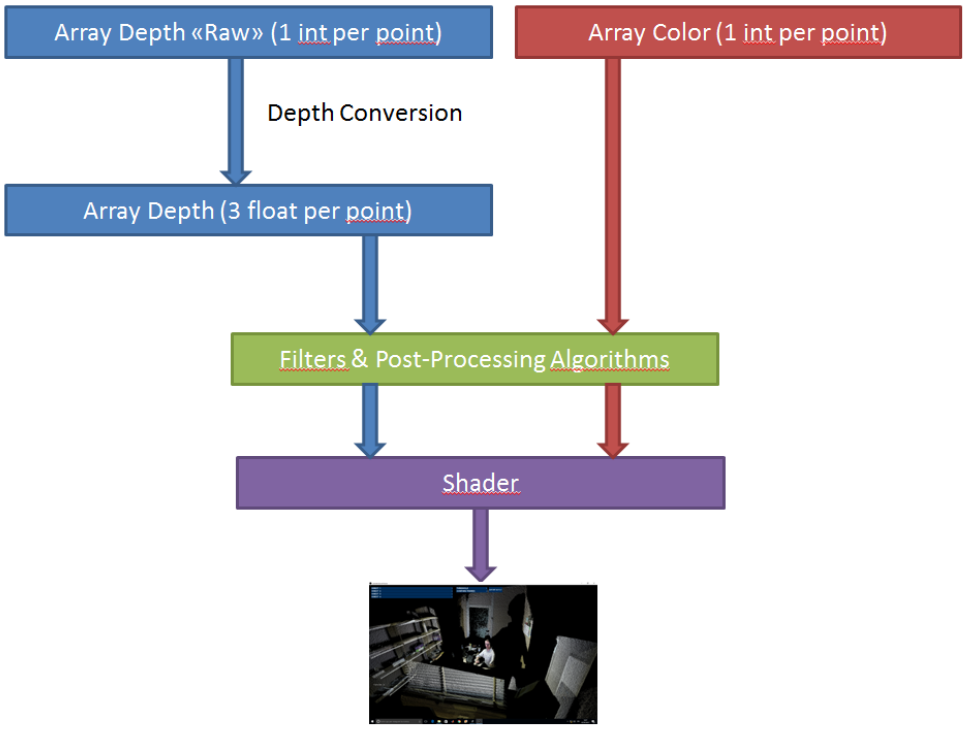

Figura 14 - Schema dell'elaborazione grafica dei due array (depth e color)

La trasformazione dell'array di profondità si basa sul concetto che il valore intero del primo array rappresenta il valore della coordinata z (asse ortogonale rispetto all'osservatore), quindi in qualche modo della profondità vera e propria. Per quanto riguarda i due restanti valori, si esegue un ciclo su una matrice che ha le dimensioni della risoluzione della camera IR della Kinect (512x424). Per ogni pixel, si estrae dalla coordinata z i valori della x e della y secondo una formula che unisce gli indici della matrice e i valori hardware del sensore. L'algoritmo è descritto nel listato seguente.

```
/*Metodo per ricavare un array depth (Float) da un array raw depth (Int)*/
  float[] convertIntoDepthPositions(int[] depthData, int depthWidth,
                                    int depthHeight){
  //Creo un buffer di float di dimensione int*3
  float[] depthDataFloat = new float[depthData.length*3];
  //Per ogni int di depthData, faccio una trasformazione per ricavare x,y,z
  int j; //indice dell'array float da riempire
  for (int x=0; x<depthWidth; x++) {
    for (int y=0; y<depthHeight; y++) {
      int i = x + y * depthWidth;
      int val = depthData[i];
      float zCoord = val;
      float xCoord = (x - CameraParams.cx)*zCoord / CameraParams.fx;
      float yCoord = (y - CameraParams.cy)*zCoord / CameraParams.fy;
      j = i*3;
      depthDataFloat[j]=xCoord;
      depthDataFloat[j+1]=yCoord;
      depthDataFloat[j+2]=zCoord;
    }
  }
```

```
  return depthDataFloat;
}

/*Camera information based on the Kinect v2 hardware*/
static class CameraParams {
  static float cx = 254.878f;
  static float cy = 205.395f;
  static float fx = 365.456f;
  static float fy = 365.456f;
  static float k1 = 0.0905474;
  static float k2 = -0.26819;
  static float k3 = 0.0950862;
  static float p1 = 0.0;
  static float p2 = 0.0;
}
```

3.2.2 Depth Threshold

L'array di profondità può essere gestito per introdurre nuove funzionalità all'interno dell'applicazione COBRAKIN. Infatti, se ogni valore contenuto nell'array indica il valore della coordinata z, l'operatore può applicare filtri in base all'ambiente in cui si sta lavorando, e interagire con i valori di profondità per raggiungere determinati task.

Una delle funzionalità fondamentali di COBRAKIN che sfrutta questo principio è l'applicazione alla scena 3D di una soglia di profondità (**Depth Threshold**). Il concetto è quello di introdurre un valore di soglia che indica la distanza dalle Kinect. Una volta impostato il valore, il sistema, agendo sull'array di profondità, escluderà dalla scena tutto ciò che si trova oltre la soglia, in modo tale da focalizzare l'attenzione solo sullo spazio di interesse. Naturalmente il filtro può essere usato anche in maniera inversa, escludendo dalla scena tutto ciò che si trova entro una certa distanza dalla Kinect, e mostrando solo gli oggetti che sono visibili oltre la soglia.

Questo filtro apparentemente semplice può essere in realtà usato per una moltitudine di applicazioni pratiche. Pensiamo ad esempio, nell'ambito della video sorveglianza, come un operatore grazie a questa funzionalità possa facilmente escludere sfondo e oggetti distanti privi d'interesse e concentrarsi invece sulla zona più vicina che necessita di essere sorvegliata, con la possibilità di modificare in tempo reale le dimensioni, e quindi la visibilità, di questo spazio di interesse. In alternativa, usando il principio della soglia in maniera inversa, l'operatore può decidere di selezionare un target di interesse ad una certa distanza escludendo tutto ciò che appare prima, magari aggiungendo una seconda soglia per meglio delimitare il perimetro di interesse, facendo sì che la distanza del target sia il valor medio tra le due soglie.

A livello di resa grafica COBRAKIN implementa due modalità differenti per la visualizzazione della soglia grafica: la prima modalità cancella dalla scena tutto ciò che eccede la soglia impostata; l'utente visualizzerà quindi solo gli oggetti che rientrano nel campo d'interesse, mentre il resto della scena non sarà visibile. La seconda modalità, chiamata "Alarm Mode", mantiene la visibilità dell'intera scena, ma colora

uniformemente tutto lo spazio che rientra al di sotto della soglia impostata. Nel caso di un perimetro da non violare ad esempio, questa modalità darà risalto a tutto ciò che supera l'ipotetica soglia di sicurezza. Le due modalità sono raffigurate negli screenshot seguenti.

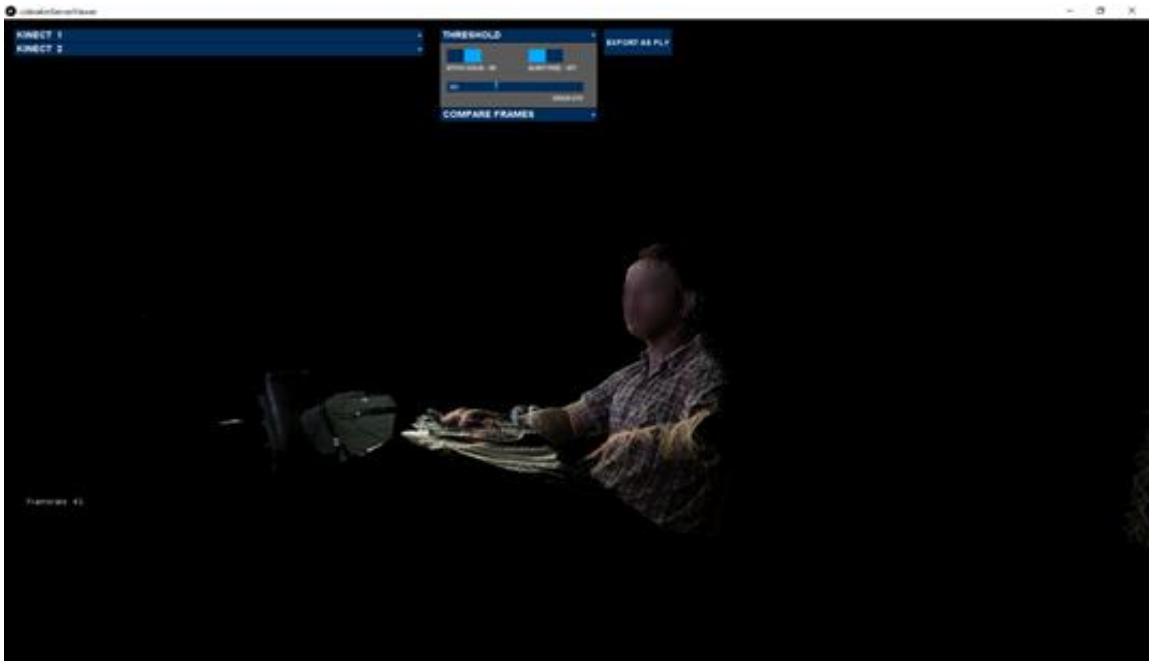

Figura 15 - Esempio d'uso di Depth Threshold (modalità standard)

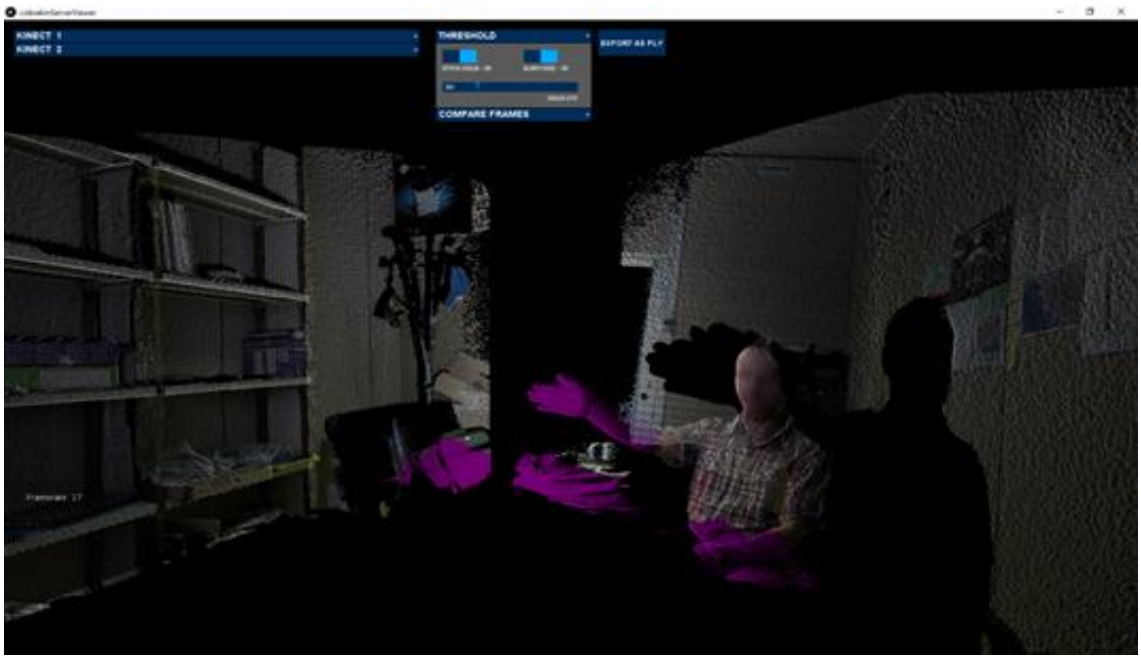

Figura 16 - Esempio d'uso di Depth Threshold (Alarm Mode)

L’algoritmo è descritto nel listato seguente.

```
/** Filtro per l'attivazione della DEPTH THRESHOLD **/
void activateThreshold(FloatBuffer depthPositions, IntBuffer regDataNew) {
  //Ciclo sull'array della depth
  for (int i = 2; i < 512 * 424 * 3; i+=3) {
    float val = depthPositions.get(i);
    //Se il valore eccede la soglia e non è attivata la modalità ALARM
    if (val<0 || val>soglia) {
      if (!activateAlarm) {
        //Cancella il pixel dalla scena
        depthPositions.put(i, -999999);
      }
    }
    //Altrimenti disegna normalmente il pixel
    else {
      //Se è attivata la modalità ALARM
      if (activateAlarm) {
        //Colora il pixel di viola
        regDataNew.put((i-2)/3,color(120,0,120));
      }
    }
  }
}
```

3.2.3 Background Subtraction

Nel paragrafo 3.1.3 è stato illustrato il protocollo di comunicazione fra PC-CLIENT e PC-SERVER spiegando come, per mantenere leggero il carico di dati sulla rete, venga utilizzato un meccanismo di **Background Subtraction** per inviare al server prima il background, e poi solo i pixel della scena che risultano in movimento.

Gli algoritmi di Background Subtraction sono piuttosto comuni nel campo dell’Image Processing. Normalmente, l’algoritmo memorizza inizialmente un singolo frame dalla videocamera e lo usa successivamente come base di confronto per i frame seguenti. L’immagine risultante mostrerà tutto ciò che è cambiato nella scena. Nel nostro caso, il PC-SERVER riceverà il frame di background all’avvio dell’applicazione, mentre il PC-CLIENT lo userà come base di confronto per tutti i frame successivi, inviando al PC-SERVER solo il foreground, che quest’ultimo unirà al background in memoria per mostrare l’immagine in tempo reale.

Nel listato seguente vediamo come è implementato sul PC-CLIENT l’algoritmo di Background Subtraction. In particolare è mostrato il metodo di acquisizione del foreground in cui viene confrontato il frame attuale con quello di background memorizzato in memoria. La ricerca delle differenze, e quindi dei cambiamenti, nell’immagine è implementata calcolando per ogni pixel la differenza nelle sue componenti di colore Red, Green e Blue (RGB). La differenza tra i colori dovrà essere maggiore di una soglia di tolleranza, ma allo stesso tempo minore di un valore standard di noise. Questo accorgimento si è reso necessario per escludere

dal cambiamento della scena quei pixel che cambiavano drasticamente di valore in seguito a problemi di rumore del sensore Kinect. Introducendo questa soglia di noise, il carico di dati in rete si è ridotto considerevolmente, migliorando il framerate e la resa grafica finale. Ricordiamo che l'array di foreground da inviare al PC-SERVER ha tre componenti per ogni pixel: la depth, il colore e l'indice di posizione rispetto alla scena.

```
/******************* BACKGROUND SUBTRACTION ***************************/

int[] getForeground(int[] depthData, int[] regData, int kinectIndex) {
    //Inizializzo un array vuoto per il foreground di capacità massima
      (caso peggiore)
    int[] newForegroundData = new int[217088*3];
    int arraySum=0;
    //Per ogni pixel della scena
    for (int k=0; k<depthData.length; k++) {
        //Salvo il colore del pixel
        color currColor = regData[k];
        color prevColor = 0;
        //Recupero il colore del pixel nel frame di background
        switch (kinectIndex) {
            case 2:
                prevColor = backgroundColorPixels[k];
                break;
            case 3:
                prevColor = backgroundColorPixels2[k];
                break;
        }
        //Recupero le componenti Red, Green e Blue dal pixel corrente
        int currR = (currColor >> 16) & 0xFF; // Like red(), but faster
        int currG = (currColor >> 8) & 0xFF;
        int currB = currColor & 0xFF;
        //Recupero le componenti Red Green e Blue dal pixel di background
        int prevR = (prevColor >> 16) & 0xFF;
        int prevG = (prevColor >> 8) & 0xFF;
        int prevB = prevColor & 0xFF;
        //Calcolo le differenze fra i valori di Red, Green e Blue
        int diffR = abs(currR - prevR);
        int diffG = abs(currG - prevG);
        int diffB = abs(currB - prevB);
        //Calcolo la somma delle singole differenze
        int movementSum = diffR + diffG + diffB;
        //Se una delle differenze eccede un valore di tolleranza e la somma è
          minore di un valore soglia
        //(Il valore max di soglia serve per impedire che vengano calcolati
          come oggetti in movimento una parte del noise della Kinect)
        if ((diffR>colorTolerance || diffG>colorTolerance ||
             diffB>colorTolerance) && movementSum<noiseValue){
            //Inserisco nell'array di foreground la depth, il colore e l'indice
              del pixel
            newForegroundData[arraySum] = depthData[k];
            newForegroundData[arraySum+1] = regData[k];
            newForegroundData[arraySum+2] = k;
            arraySum+=3;
        }
    }
```

```
    //Se la dimensione dell'array di foreground è minore della capacità massima
      inizializzata
    if (arraySum<newForegroundData.length) {
        //Elimino dall'array gli elementi vuoti
        int[] result = subset(newForegroundData,0,arraySum);
        //Ritorno l'array di foreground
        return result;
    }
    //Ritorno l'array di foreground
    else {
        return newForegroundData;
    }

}
```

4 Risultati

Il sensore COBRAKIN è stato testato con successo in laboratorio dimostrando la validità dell'architettura hardware e software e il funzionamento degli algoritmi di Computer Vision sopra elencati. Il framerate dell'immagine 3D generata si è rivelato pienamente soddisfacente nell'ottica degli obiettivi di sicurezza e sorveglianza per cui il dispositivo è stato progettato. Inoltre, anche la struttura di supporto in cui le 4 Kinect sono contenute è stata completamente progettata e realizzata all'interno del C.R. ENEA di Frascati (Figura 17 - Aspetto finale del sensore COBRAKIN).

Figura 17 - Aspetto finale del sensore COBRAKIN

Riguardo agli sviluppi futuri, il dispositivo attuale è un punto di partenza da cui implementare ed estendere funzionalità sempre più avanzate. Un esempio è nella realizzazione di tecniche avanzate di Body Tracking e di Face Recognition: nel primo caso COBRAKIN può essere programmato per individuare in tempo reale l'ingresso di una persona nella scena, seguirne i movimenti e calcolare la sua distanza da target predefiniti, facendo scattare un allarme nel caso il soggetto violi la soglia di sicurezza; nel secondo caso, l'ingresso di una nuova persona nell'area di scansione può determinare un'acquisizione istantanea dei tratti somatici del soggetto, ed un successivo confronto con un database noto per individuare immediatamente la presenza di persone indesiderate e/o potenzialmente pericolose. Nel caso in cui il soggetto non sia presente nei database, è possibile memorizzare il suo volto per analisi future, se le disposizioni in materia di privacy di questo ipotetico scenario lo permettessero.

Va ribadito che tutte le nuove funzionalità in materia di Computer Vision che in futuro potranno essere introdotte, si integrano perfettamente con relativa semplicità nell'architettura software modulare con cui

COBRAKIN è stato programmato. Anche il loro utilizzo da parte di un operatore risulterà intuitivo grazie all'aggiunta di nuovi menu e controlli all'interno dell'interfaccia grafica attuale.

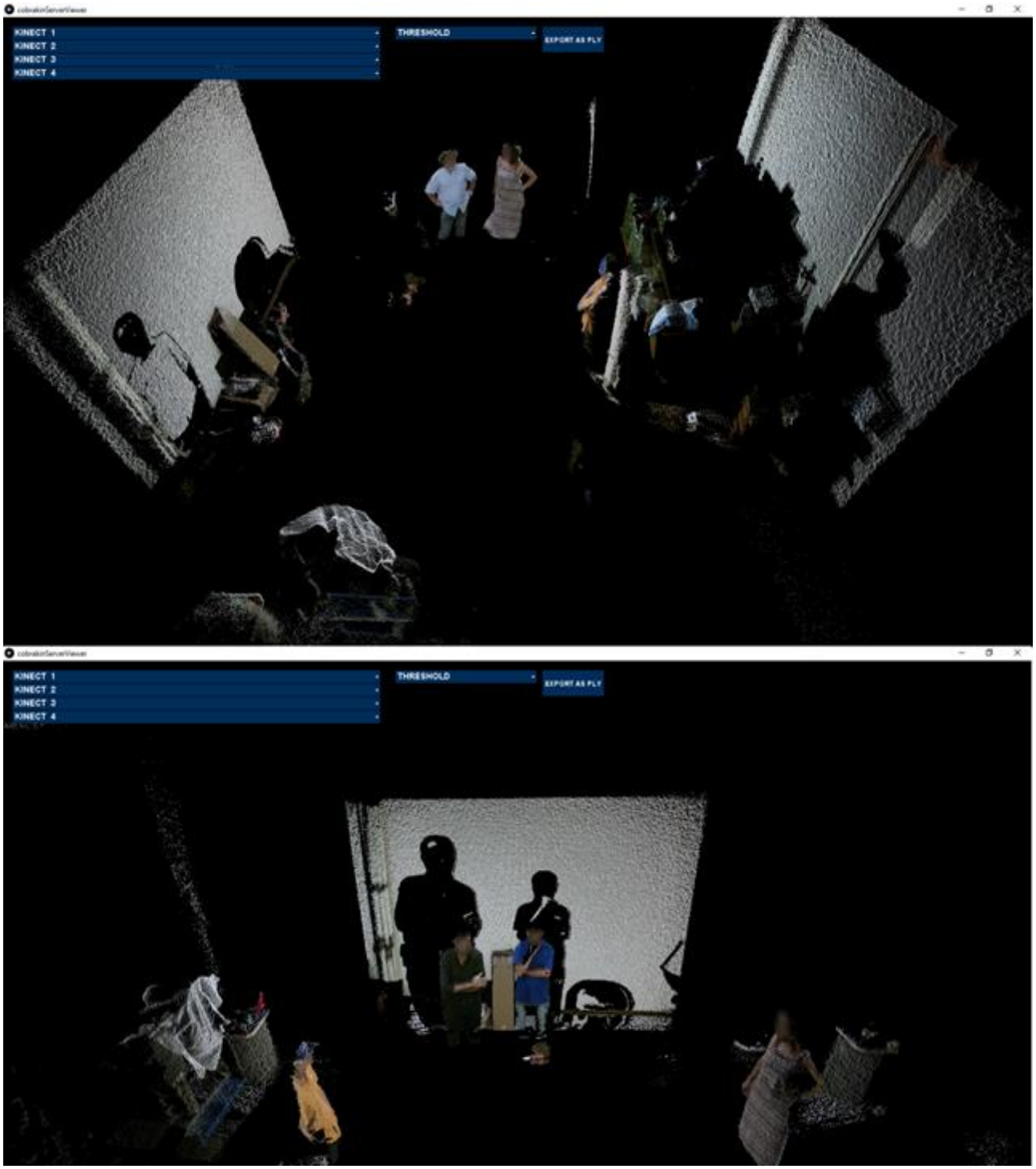

Figura 18 - Prove di COBRAKIN in laboratorio

4.1 Screenshot framerate

Di seguito vengono mostrati alcuni screenshot per analizzare il tasso di frame al secondo registrato dal sistema COBRAKIN in diverse modalità di utilizzo. Va ricordato che per framerate si intende la frequenza di cattura o di riproduzione dei fotogrammi che compongono un filmato, e l'attuale standard di riproduzione televisiva/cinematografica si attesta sui 24fps, una cifra che consente all'occhio umano di non rilevare rallentamenti e di non percepire le immagini come "artefatte".

In Figura 19 - **Screenshot Framerate 1**si può notare un framerate oltre i 40fps nel caso in cui vengano utilizzate solo le 2 Kinect collegate al PC-SERVER tramite USB.

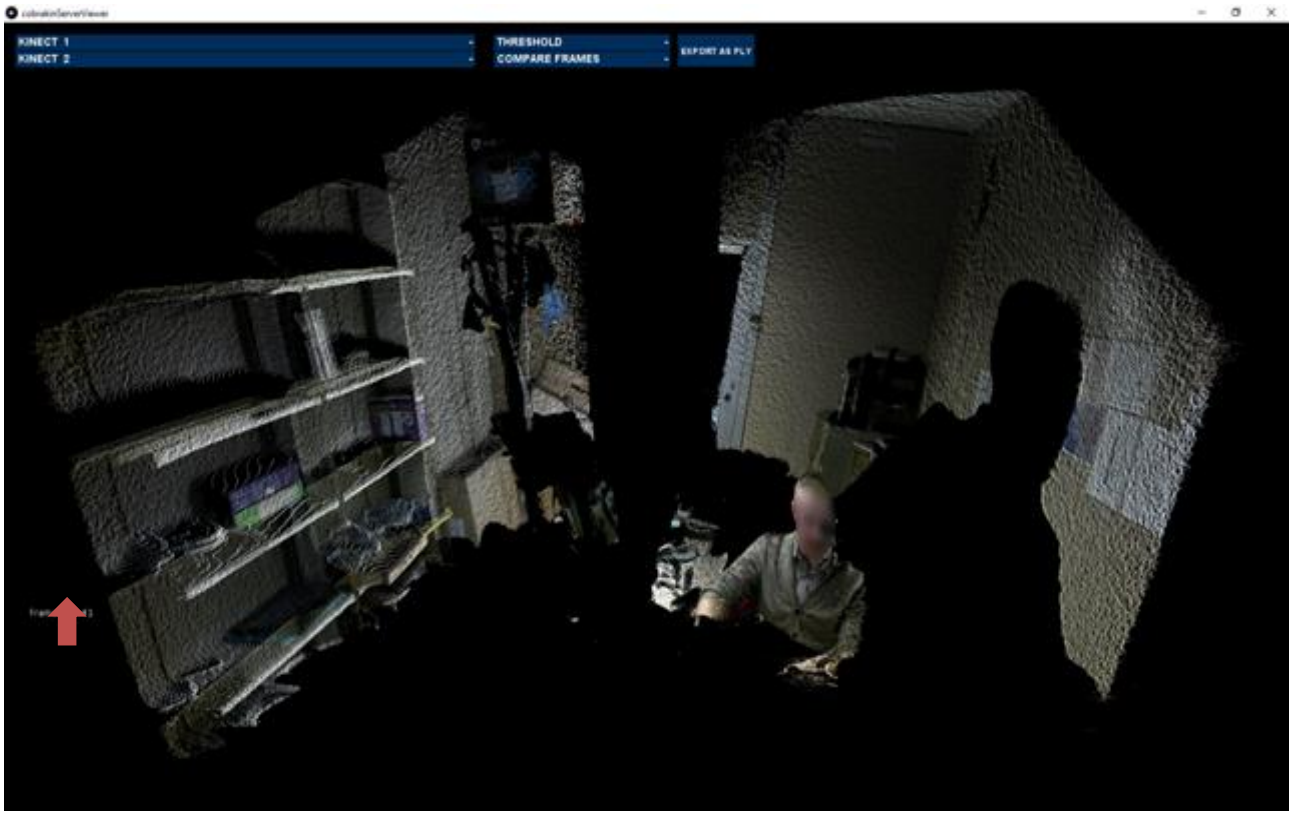

Figura 19 - Screenshot Framerate 1

Connettendo le altre 2 KINECT tramite LAN per mezzo del PC-CLIENT possiamo notare nella Figura 20 - Screenshot Framerate 2un framerate che varia tra i 15fps e i 20fps. Il valore, pur essendosi dimezzato, non inficia la corretta visualizzazione della scena 3D da parte dell'operatore, avvicinandosi infatti allo standard TV/cinema.

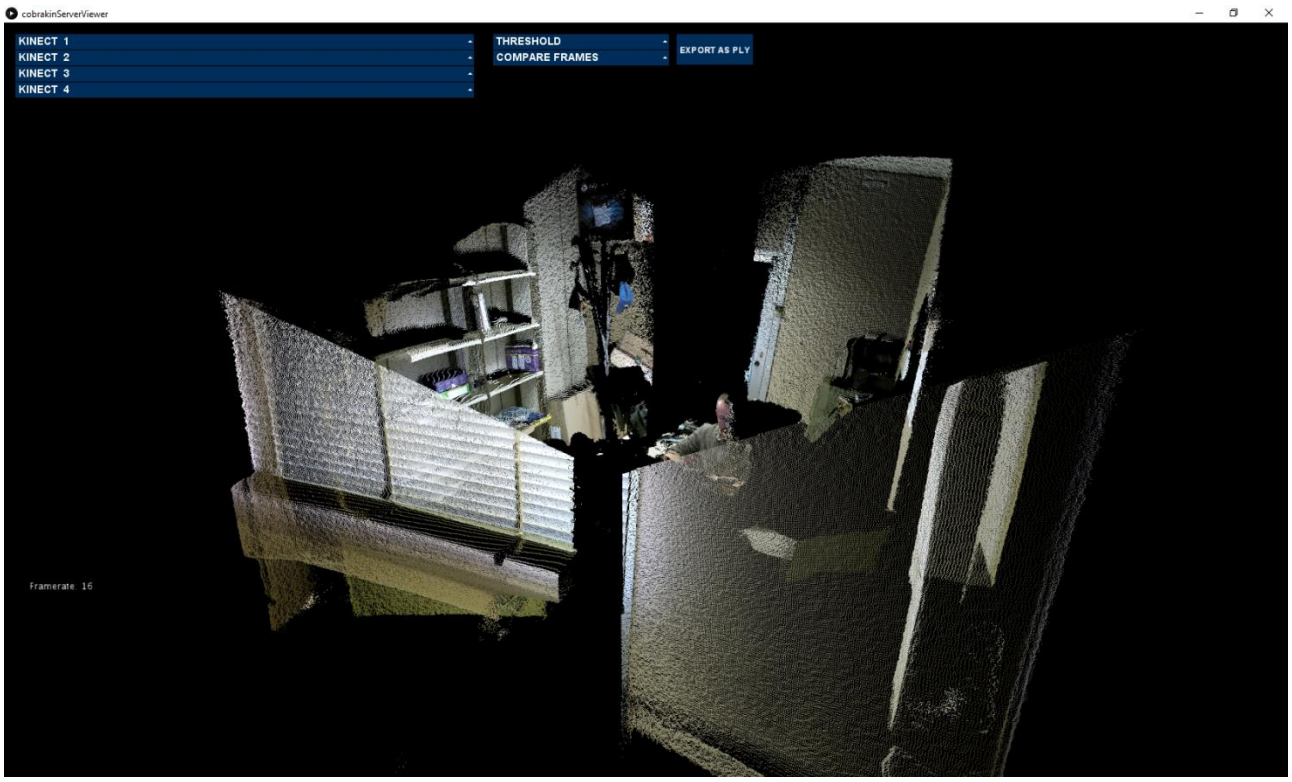

Figura 20 - Screenshot Framerate 2

Applicando un algoritmo di Depth Threshold vediamo in figura 20 un framerate che viaggia ad una velocità di 13fps. La diminuzione del valore rispetto all'esempio precedente è minima, e quindi praticamente impercettibile all'occhio umano. Va precisato che il carico computazionale su un algoritmo di questo tipo non è elevato, dato che si tratta del confronto di un vettore di valori con un numero fisso.

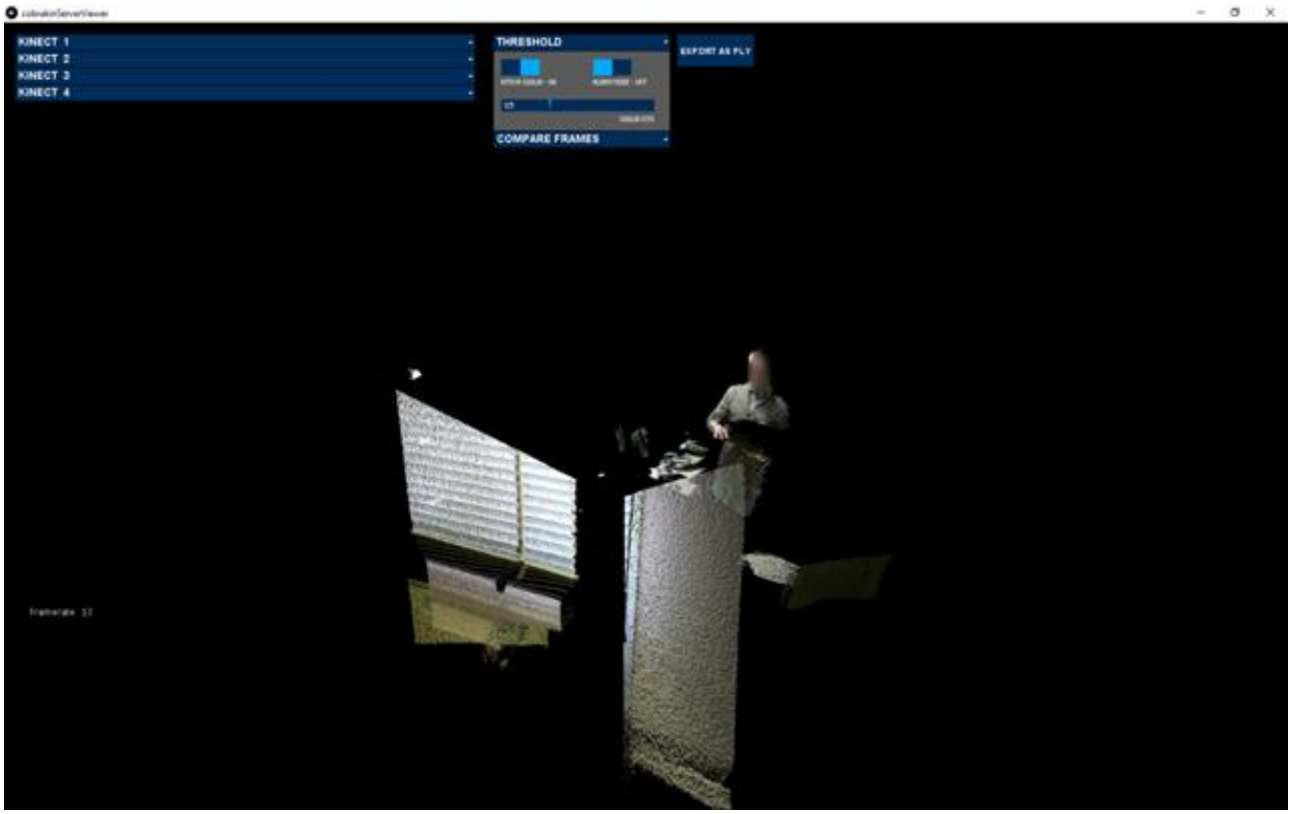

Figura 21 - Screenshot Framerate 3

Un algoritmo di maggiore complessità come può essere un algoritmo di Background Subtraction implica un'ulteriore diminuzione del framerate (come mostrato in Figura 22 - Screenshot Framerate 4) sotto i 10fps.

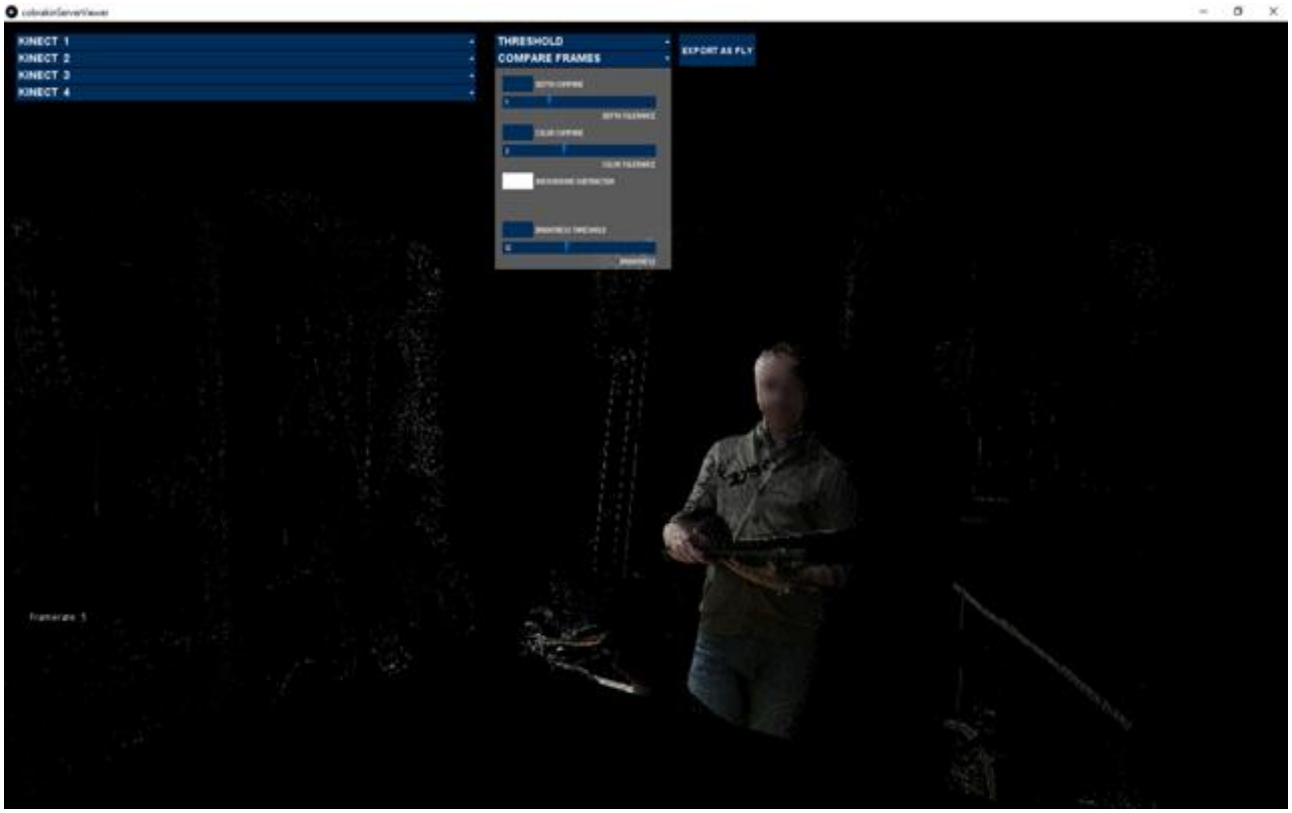

Figura 22 - Screenshot Framerate 4

Per un dispositivo di questa natura, che possiede un'architettura complessa e una serie di funzionalità avanzate di Computer Vision, il framerate rimane in ogni caso molto soddisfacente, nonché adeguato agli scopi del sensore. Ciò non toglie che nei successivi aggiornamenti il miglioramento della frequenza di riproduzione e l'eliminazione di potenziali "colli di bottiglia" sarà uno degli obiettivi principali degli sviluppatori.

ENEA
Servizio Promozione e Comunicazione
www.enea.it

Stampa: Laboratorio Tecnografico ENEA - C.R. Frascati
ottobre 2017