ALMA MATER STUDIORUM

UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Sede di Forlì

Corso di Laurea in
INGEGNERIA AEROSPAZIALE
Classe 10

ELABORATO FINALE DI LAUREA

in Costruzioni Aeronautiche

PROGETTAZIONE E REALIZZAZIONE DI UN SISTEMA DI EYE-TRACKING PER LA GENERAZIONE DI CONTENUTI IN REALTÀ AUMENTATA IN TORRE DI CONTROLLO

CANDIDATO
Simone Pelatti

RELATORE
Sara Bagassi

Anno Accademico 2018/2019

INDICE

INTRODUZIONE

Gli aeroporti sono una parte fondamentale della rete del trasporto aereo e la loro capacità è identificata come uno dei principali colli di bottiglia per la crescita del settore dell'aviazione europeo. Per affrontare il clima di crescente pressione negli aeroporti, stanno emergendo nuove tecnologie a supporto del lavoro dei controllori di torre. In particolare, una estensione ulteriore delle applicazioni e delle tecnologie sviluppate per le operazioni con torri di controllo remote, il cui utilizzo è già consolidato, può offrire soluzioni per un uso in situazioni convenzionali con un significativo aumento della sicurezza e dell'efficienza.

Un vantaggio importante si può avere da nuovi metodi per mostrare dati di aeromobili, veicoli, strutture e meteorologia in una classica torre di controllo aeroportuale, presentando ulteriori o migliori informazioni sui dispositivi già presenti o introducendone di nuovi, per esempio gli head-up display, potendo quindi fornire contenuti di realtà virtuale (VR) e realtà aumentata (AR).

Per rendere disponibile un ambiente in Realtà Aumentata in torre di controllo occorre valutare e specializzare a questa particolare situazione metodi, conoscenze e tecnologie già usati in altre applicazioni di AR e provvederne di nuovi ad-hoc.

In questo contesto si inserisce il concept RETINA (Resilient Synthetic Vision for Advanced Control Tower Air Navigation Service Provision) dell'Università di Bologna, che indaga sulle potenzialità e i metodi dell'applicazione della VR/AR ad uso del controllo del traffico aereo (Air Traffic Control, ATC). Lo scopo del progetto RETINA è contribuire significativamente alla visione a lungo termine del futuro "Single European Sky", tra i cui requisiti vi è quello di operare in sicurezza ed efficienza, tanto in condizioni di scarsa visibilità quanto in condizioni di buona visibilità.

L'approccio di RETINA si basa sulla sovraimpressione alla visione reale di overlays sintetici, che riportano informazioni aggiuntive o che consentano di gestire meglio quelle già presenti.

Figura 0-A: esempio di applicazione di Realtà Aumentata in torre di controllo tramite il concept RETINA. Università di Bologna, foto: Simone Pelatti, Sara Bagassi.

Questo può avvenire, principalmente, tramite l'utilizzo di Head-Mounted Displays o di schermi semitrasparenti fissi Head-Up Displays. Quest'ultima tecnologia ha il vantaggio di non isolare l'operatore e non richiede di indossare nessun dispositivo, in quanto tutta la strumentazione di generazione e visualizzazione dei contenuti in AR è integrata nella struttura della postazione di controllo.

Uno dei requisiti necessari dell'AR in questa configurazione è la presenza di un sistema di tracciamento del punto di vista dell'operatore (eye-tracking), per fornire contenuti virtuali coerenti (registrati) con la sua posizione reale.

Questa ricerca ha come scopo principale la creazione di un sistema di eye-tracking specifico per l'applicazione in torre di controllo e, collateralmente, fissa alcuni punti di riferimento generali per l'utilizzo di Realtà Aumentata in questo ambiente.

1. REALTA' AUMENTATA IN TORRE DI CONTROLLO: STATO ATTUALE

Le tecnologie fondamentali per lo sviluppo del concept RETINA sono Realtà Virtuale e Realtà Aumentata, strumenti informatici che sono in continuo sviluppo ed evoluzione e che, recentemente, hanno raggiunto un livello di affidabilità soddisfacente per l'utilizzo in applicazioni aeronautiche. Contemporaneamente, sono anche diventati mezzi popolari e facilmente implementabili su dispositivi personali quali smartphone e tablet per le più svariate applicazioni, cosa che garantisce un continuo abbassamento dei costi e una sempre maggiore disponibilità di professionisti in grado di sviluppare contenuti ad-hoc per queste risorse. Tutto ciò rende possibile la sperimentazione di soluzioni in Realtà Virtuale e Aumentata in ambito aeronautico anche con budget limitati e potendo contare sul supporto di una comunità di sviluppatori in costante crescita. La progressiva miniaturizzazione delle apparecchiature necessarie per fruire di queste tecnologie rende infine semplice la loro implementazione in ambienti come le torri di controllo del traffico aereo, dove non è indicato ingombrare eccessivamente lo spazio e la visuale con strumentazioni ed annessi.

1.1. Realtà Virtuale e Realtà Aumentata

Per realtà virtuale (Virtual Reality, VR) si intende una simulazione della realtà effettiva generata tramite tecnologie informatiche. L'avanzamento di queste tecnologie ha raggiunto un livello tale da rendere possibile la navigazione di ambienti virtuali dall'aspetto fotorealistico, e l'interazione, anche molto complessa, con gli elementi che popolano questi ambienti. Idealmente la VR potrebbe abbracciare tutti i cinque sensi, ma attualmente sono soprattutto la vista e l'udito a trovare le implementazioni migliori in queste tecnologie. Questo può dipendere dal fatto che nell'uomo è la vista il senso dominante, seguita dall'udito,

e per lo stesso motivo anche le tecnologie che riproducono questi stimoli sono quelle più evolute e di uso comune.

Figura 1-A: Simulatore di volo per T-38A Talon con abitacolo immerso in un ambiente ricreato tramite realtà virtuale. U.S. Air Force, photo: Javier Garcia.

La fruizione dei contenuti può avvenire sia tramite dispositivi tradizionali (schermi, casse acustiche…) sia più avanzati e che consentono una immersione maggiore nell'ambiente virtuale (soluzioni visive tridimensionali, audio surround, motion-tracking…). Anche l'interazione con i contenuti può essere classica (mouse, tastiera, joystick…) o innovativa (comandi gestuali e vocali…).

Affine alla VR è la Realtà Aumentata (Augmented Reality, AR), tramite cui è arricchita la percezione sensoriale normale con l'aggiunta di contenuti in VR, fornendo informazioni addizionali che migliorano la percezione dell'ambiente e che non sarebbero disponibili normalmente[1]. In senso lato, l'AR può anche essere usata per ridurre le informazioni, per semplificare la percezione e concentrarsi su quanto sia più importante. La modalità tipica tramite cui operano le

[1] Azuma, R. T. (1997) A survey of augmented reality.

strumentazioni di AR è l'aggiunta di livelli sintetici (overlay) alla visuale reale, che presentano le informazioni addizionali che si vogliono rendere disponibili.

Questi livelli possono avere contenuti fissi rispetto al display (ad esempio, un indicatore di velocità proiettato direttamente sul parabrezza di un'automobile) o spazialmente e geometricamente conformi rispetto all'ambiente, come un box generato attorno ad un soggetto, che lo segue nei suoi spostamenti relativi (in questo caso si parla di contenuti "registrati" rispetto alla realtà).

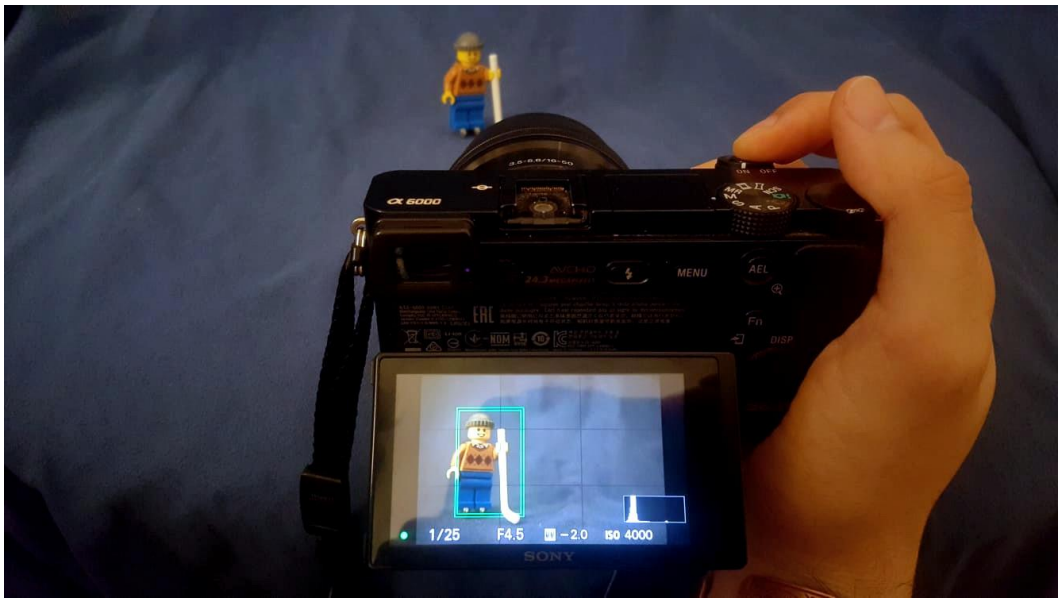


Figura 1-B: il display di una moderna fotocamera fornisce contenuti di AR. I dati numerici di esposizione sono fissi rispetto al display, mentre il riquadro verde è agganciato al soggetto da mettere a fuoco e rimane registrato rispetto ad esso. Foto: Simone Pelatti, Alice Fabbri.

Le prime e più celebri applicazioni dell'AR sono state in ambito militare (Head-Up Display, HUD, per gli aerei da combattimento) o medico, mentre oggi esistono svariate possibilità alla portata di tutti, come in ambito fotografico, tramite opportuni software per smartphone o nel cruscotto di alcuni veicoli. Il numero e la tipologia di queste applicazioni è destinato ad aumentare, specialmente in ambito educativo, dei trasporti e nell'industria manifatturiera.

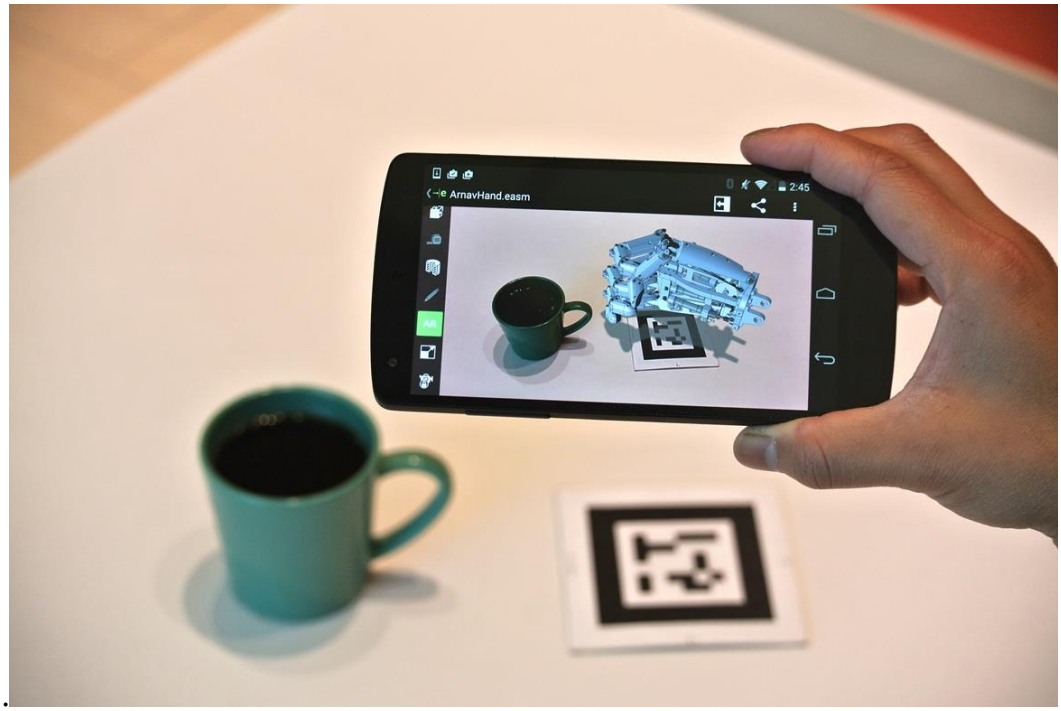


Figura 1-C: Esempio di realtà aumentata implementata tramite uno smartphone e un QR-code per far apparire un oggetto virtuale in un contesto reale. Foto: Antonio Martino, treminuti.eu.

1.2. VR e AR in torre di controllo (Air Traffic Control Tower, ATCT)

L'ausilio di AR e VR può beneficiare al sistema di controllo del traffico aereo, in particolare potenziando gli strumenti a disposizione dei controllori di torre, in quanto queste tecnologie permettono di superare o mitigare certi limiti legati alla visibilità, alla operatività e alla distanza delle strutture rispetto ai punti di interesse. Infatti le mansioni principali dei controllori si basano su informazioni visive, spesso dirette come guardare con il binocolo all'esterno[2], ed è proprio su questo tipo di stimoli che intervengono VR e AR. Alcune esperienze, come quelle con i simulatori di torre di controllo e le torri remote, hanno già evidenziato come l'utilizzo di queste tecnologie sia performante e affidabile. Inoltre, in una

[2] Ronald J. Reisman, David M. Brown (2006) Design of Augmented Reality Tools for Air Traffic Control Towers.

condizione di sempre crescente saturazione degli aeroporti, il ricorso alle tecnologie di AR e VR come supporto ai controllori può migliorare la loro efficienza e la sicurezza degli aeroporti.

Già negli anni ‘80 era stato proposto l’utilizzo di HUD in torre di controllo, che fornissero ai controllori informazioni sull’identificazione degli aerei e condizioni metereologiche[3]. Recentemente, proposte più avanzate[4] [5] [6] [7] suggeriscono che certe informazioni, come la posizione di aeromobili, veicoli terrestri e gli edifici aeroportuali, possono essere presentati tramite sistemi di AR, rendendole spazialmente conformi alla realtà utilizzando nuovi sensori di immagine, posizione GPS o radar ad alta precisione. Tali display AR potrebbero fornire ai controllori di torre una sorta di "visione a raggi X", permettendo loro di continuare le operazioni in condizioni meteorologiche che normalmente potrebbero far chiudere l'aeroporto o ridurne la capacità, a causa della limitata capacità visiva.

Infine, è emerso che il tempo di head-down in torre di controllo è uno dei principali fattori che contribuiscono ad aumentare il rischio di non percepire tempestivamente situazioni ed eventi critici[8], poiché l’operatore ha lo sguardo rivolto agli strumenti e non può vedere cosa succede nel frattempo all’esterno. Inoltre, le attività head-down richiedono la variazione della messa a fuoco dei controllori tra la visuale al di fuori della torre e gli strumenti e i dispositivi all'interno della torre, causando stanchezza e affaticando la vista degli operatori.

---

[3] Ellis S. R. (2006), “Towards Determination of Visual Requirements for Augmented Reality Displays and Virtual Environments for the Airport Tower.

[4] Redeiss, H. A. (1997) An augmented reality pilot display for airport operations under low and zero visibility conditions. AIAA Guidance, Navigation, and Control Conference, New Orleans, LA.

[5] Krozel. J., Birtcil, L., Mueller, K. T., & Azuma, R. (1999) Augmented reality system for the ATC Tower. TR 98183-01 NASA Contract NAS299024, NASA Ames Research Center, Moffett Field, CA.

[6] Ellis, S. R., Schmidt-Ott, J. R., Krozel, J., Reisman, R. J., Gips, J. (2002) Augmented reality in a simulated tower environment: effect of field of view on aircraft detection. NASA TM 2002-211853 (October, 2002).

[7] Fuerstenau, N., Rudolph, M., Schmidt, M., Bernd, W. (2004) Virtuelle Tower (Virtual Tower) DLR-Sonderheft: Wettbewerb der Visionen, Beitrag in Zeitschrift, Institut fuer Flugfuehrung, Braunschweig, Deutschland (Germany).

[8] Hilburn, B. (2004a), "Head-down Time in Aerodrome Operations: A Scope Study" Center for Human Performance Research (CHPR). The Hague, The Netherlands

Le informazioni grafiche presentate su un dispositivo AR possono eliminare questi problemi, poiché eviterebbero o ridurrebbero ai controllori la necessità di consultare gli strumenti all'interno della torre.

1.3. I precedenti delle torri di controllo remote e virtuali

Le torri di controllo remote e virtuali (Remote and Virtual Tower, RVT) sono un nuovo approccio alla fornitura di servizi di controllo del traffico aereo, in cui il controllore si trova altrove rispetto all'aerosuperficie che gestisce. Gli utenti dello spazio aereo dovrebbero disporre del livello adeguato di servizi, come se l'ATS fosse fornito localmente nell'aeroporto. Nell'aeroporto locale sarebbe collocata una torre munita di telecamere e altri strumenti, mentre la sala di controllo in cui si svolgono le operazioni potrebbe trovarsi i qualunque altro luogo, e la visualizzazione della vista dell'aeroporto locale avviene tramite schermi panoramici.

Figura 1-D: esterno di una torre di controllo remota SAAB durante una manutenzione. Newatlas.com, foto: David Szondy.

Figura 1-E: interno di una sala di controllo di una torre di controllo remota NATS. Foto: www.nats.aero.

Il concetto RVT, oltre a poter fornire servizi di torre a distanza in aeroporti di piccole e medie dimensioni, può essere sfruttato per garantire servizi di emergenza in altri aeroporti, in caso di incendio o altri eventi che potrebbero svolgersi nell'edificio della torre di controllo tradizionale.

La prima implementazione completa di una torre remota è stata realizzata in Svezia nell'Aprile 2015, con ulteriori implementazioni in altri Stati membri dell'Agenzia europea per la sicurezza aerea (European Aviation Safety Agency, EASA). Il 1° ottobre 2015 la Federal Aviation Administration statunitense, FAA, ha annunciato l'aeroporto municipale di Fort Collins-Loveland come il primo sito ufficiale di collaudo della torre di controllo del traffico aereo virtuale approvato dalla FAA negli Stati Uniti.

Il concetto base, precedentemente noto come torri virtuali, è stato introdotto da Deutsches Zentrum für Luft (DLR) nel 2002 [9] e descrive una sala di controllo remota con sorveglianza basata su sensori video invece che su una vista diretta "out of the window" da una vera torre. Le prove iniziali di ATC da remoto, per

[9] AA.VV, DLR (2004) Wettbewerb der Visionen 2001 – 2004.

aeroporti a bassa e media densità, sono state basate su sensori ottici (telecamere), fornendo agli operatori presso la RVT un'immagine della pista di alta qualità e in tempo reale, così come anche dell'area di stazionamento dell'aeroporto (APRON) e lo spazio aereo circostante. Queste immagini in tempo reale vengono visualizzate su monitor di grandi dimensioni, eventualmente disposti in modo da garantire una visualizzazione a 360 gradi.

Oltre al feed video in diretta dall'aeroporto, i controllori hanno a disposizione gli stessi sistemi di gestione del traffico aereo che avrebbero in una torre di controllo locale, sistemi di comunicazione vocale, sistemi meteorologici, sistemi di gestione dei piani di volo e sistemi di sorveglianza. A seconda della complessità dell'aeroporto, della densità del traffico e delle condizioni meteorologiche, potrebbe essere preferibile integrare le immagini con informazioni provenienti dal radar o altri strumenti.

Effettivamente, durante la sperimentazione e l'approccio alle RVT, è risultato evidente come si sarebbe potuto beneficiare di un potenziamento sintetico della visione per aumentare la consapevolezza della situazione negli aeroporti in condizioni, ad esempio, di scarsa visibilità presso l'aeroporto locale. Il fatto che la visualizzazione da parte degli operatori avviene già su schermi digitali ha reso piuttosto semplice aggiungere overlay di contenuti alle immagini provenienti dalle videocamere remote, fornendo i controllori di una visione potenziata rispetto a quella che normalmente avrebbero da una ATCT tradizionale, semplificando lo svolgimento del loro lavoro e, contemporaneamente, rendendolo più resiliente e sicuro.

Figura 1-F: interno di una sala di controllo di una torre di controllo remota NATS. Si notino gli overlay con i nominativi degli aerei in volo, le informazioni radar e meteo, lo zoom digitale nel riquadro e il bordo colorato della pista. Foto: www.nats.aero.

1.4. Caratterizzazione dell'ambiente e delle attività di torre e specifiche per AR

L'ambiente tipico di una torre di controllo del traffico aereo è quello di una stanza sopraelevata completamente finestrata, da cui si gode di una visuale ampia che domina tutta la zona aeroportuale e spazia per diversi chilometri oltre questa. L'interno è adibito, in genere, alla presenza contemporanea di diverse persone, a seconda delle dimensioni dell'aeroporto.

Figura 1-G: interno di una ATCT e dello spazio esterno circostante.

Aeroporti differenti si caratterizzano per la diversa configurazione e condivisione delle funzioni tra le posizioni di controllo in torre, ma in generale è sempre presente almeno un controllore “di cielo”, che si preoccupa dei velivoli in volo, e uno “di terra”, che gestisce gli aerei a terra e i mezzi terrestri. I controllori hanno delle proprie postazioni, ma generalmente si muovono liberamente all’interno della torre, per poter consultare gli strumenti o osservare certe aree dell’aviosuperficie. I controllori comunicano tra loro oralmente e attraverso gli strumenti, così come con gli equipaggi dei velivoli e il personale di terra. L’ambiente è in genere silenzioso per permettere di comunicare in modo preciso e senza impedimenti.

Gli strumenti a disposizione più utilizzati sono il binocolo, il radar di approccio, l’indicatore meteorologico e del vento, il radar dei movimenti a terra, la stampante e l’organizzatore delle strips.

Le attività più svolte dagli operatori sono:

- Controllo e ricerca visiva dello spazio esterno alla torre (head-up);
- Scrittura e lettura delle strips (head-down);
- Passaggio delle strips ai colleghi (head-down);

- Controllare il radar di approccio (head-down);
- Controllare il radar di movimento a terra (head-down);
- Leggere i dati metereologici (head-down);
- Contattare altri servizi dell'aeroporto e i piloti (head-down / head-up).

Studi recenti hanno analizzato come i controllori passano da una attività all'altra e che affidamento fanno sui diversi strumenti[10]. In dettaglio, emerge che i controllori di terra diano la priorità alla vista esterna rispetto ad altre fonti di informazione. Monitorare la superficie dell'aeroporto è un'attività frequente e di lunga durata. In confronto, controllare le strips e altri strumenti sono attività frequenti, ma di durata relativamente breve. Tuttavia, in corrispondenza di alti volumi di traffico o in condizioni notturne e di scarsa visibilità, i controllori mostrano una tendenza maggiore a utilizzare ed affidarsi agli strumenti piuttosto che alla vista diretta dell'aviosuperficie, che rimane comunque la principale fonte di informazioni.

Una strumentazione AR adeguata per operare in queste condizioni deve:

- Essere poco invasiva;
- Garantire la massima prestazione visiva possibile (campo visivo largo);
- Non creare interferenze con gli altri strumenti presenti;
- Mantenere una perfetta leggibilità anche in condizioni di illuminazione esterna molto differenti,
- Soddisfare un'ottima ergonomia per non affaticare l'operatore (ad esempio, tecnologie indossabili non devono essere eccessivamente pesanti);
- Integrarsi con gli altri sistemi presenti, raccogliendo e presentando le informazioni che generano;
- Garantire una elevata operabilità (autonomia delle batterie, manutenzione…).

[10] Pinska E., Tijus C. (2007) Augmented reality technology for control tower analysis of applicability based on the field study.

- Essere sicura rispetto attacchi informatici o bug del sistema informatico che la gestisce.

Soddisfatte queste condizioni, la tecnologia AR scelta dove poi presentare al controllore informazioni conformi alla mansione in svolgimento. In particolare, deve essere in grado di migliorare la fruizione dei dati specifici per la mansione particolare (ad esempio, mostrare il nome e la velocità di un aereo in avvicinamento mentre lo si sta osservando, senza bisogno di distogliere lo sguardo per controllare le strips). Inoltre, deve essere in grado di mantenere un elevata consapevolezza globale, evitando di isolare l'operatore dall'ambiente e dagli stimoli esterni.

Definiti contenuti e il modo in cui mostrarli, la soddisfazione dei requisiti di operabilità pone la necessità di scegliere quale, tra le diverse tecnologie disponibili, sia la più indicata.

1.5. Tecnologie di riferimento (classificazione)

Esistono molteplici tipologie di strumentazioni per la AR, anche molto diverse tra loro. Un modo consolidato per classificarle è in base al modo in cui sono generate le immagini che vengono visualizzate, ovvero a seconda del tipo di display o proiettore che si utilizza. Più precisamente, la classificazione può essere basata sulla posizione in cui sono posizionati i contenuti AR lungo il percorso ottico tra l'oggetto osservato e gli occhi dell'osservatore[11]. Questa classificazione comprende display collegati alla testa (cioè Head-Mounted Display, HMD), display sorretti manualmente (come gli smartphone), spatial display (fissati a una struttura e non all'osservatore) e display AR proiettati su oggetti (con proiettori fissati a testa, mano libera o altrove nello spazio). In quest'ultima tipologia, l'immagine virtuale viene proiettata direttamente su un oggetto reale.

[11] Bimber, O., Raskar, R. (2005) Spatial Augmented Reality: Merging Real and Virtual Worlds. A. K. Peters, Ltd., Natick, MA, USA.

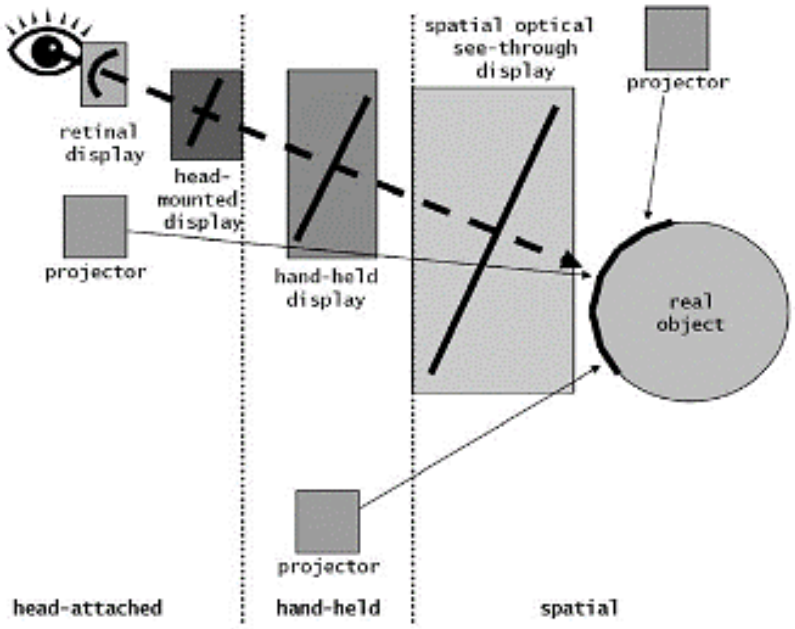


Figura 1-H: classificazione dei display AR in base alla posizione tra l'osservatore e l'oggetto. Bimber, O., Raskar, R.

Trascurando gli schermi a proiezione, i display AR possono essere ulteriormente ripartiti tra la categoria see-through, o semi-trasparenti, e la categoria video-combined o video-mixed[12]. Gli schermi semi-trasparenti AR combinano immagini reali e virtuali per mezzo di specchi, lenti, schermi trasparenti o altri componenti ottici. Questo lascia la visione del mondo reale quasi intatta, a cui è sovraimpresso il contenuto di AR. I display video-combined, invece, usano videocamere per convertire la vista reale in un feed video, che viene quindi unito all'immagine virtuale per generare la visione in AR (come avviene nelle torri remote).

Gli spatial-display sono tipicamente fissi nello spazio (ad esempio, collegati a una scrivania, fissati al pavimento o appesi al soffitto o, ancora, possono essere fatti coincidere con le finestre di un edificio). I display See-Through Head-Mounted (ST-HMD) sono indossati dall'utente, risultando in un'attrezzatura flessibile ma

[12] J. Rolland, H. Rich, and H. Fuchs (1994) "A Comparison of Optical and Video See-Through Head-Mounted Displays." In Proceedings of SPIE: Telemanipulator and Telepresence Technologies, pp. 293–307. Bellingham, WA: SPIE.

invadente. In ogni caso, affinché entrambi questi sistemi funzionino correttamente, è necessaria mantenere una prospettiva visuale allineata con la testa o, meglio, esattamente con gli occhi dell'utilizzatore. Pertanto, in qualsiasi momento, l'applicativo che produce la sovrapposizione dei contenuti AR deve sapere dove si trova il controllore e dove sta guardando. In effetti, quanto si vede sullo schermo è determinato dal punto di vista dell'osservatore, che di solito è calcolato monitorando la sua posizione e orientamento della testa o direttamente degli occhi[13]. Questo differisce da quanto avviene, ad esempio, in un HUD di un aeromobile, in cui le informazioni visualizzate sono adattate alla prospettiva dell'aeromobile piuttosto che a quella del pilota[14], in quanto il pilota stesso non ha molta liberà di movimento rispetto ad una posizione prestabilita e, inoltre, l'interazione con l'ambiente circostante (posizione rispetto all'orizzonte, azione delle armi…) è fatta dalla macchina piuttosto che dal pilota.

1.6. Head mounted display (HMD)

Gli HMD sono i dispositivi di visualizzazione che vengono attualmente maggiormente utilizzati per applicazioni di AR. Sono generalmente costituti da un semi-casco in materiale plastico, in certi casi ridotto fino alle dimensioni di un comune paio di occhiali, che racchiude display per la visione dei contenuti, unità di calcolo, sistema di alimentazione, sistema di comunicazione, sistema ottico.

Anche questi strumenti si distinguono in see-through (ST HMD), come Google Glass® e Microsoft Hololens®, e mixed-video (MV HMD) come Oculus Rift®.

[13] Masotti N., De Crescenzio F., Bagassi R. (2016) Augmented Reality in the Control Tower: a Rendering Pipeline for Multiple Head-Tracked Head-Up Displays. Department of Industrial EngineeringUniversity of Bologna.
[14] Peterson, S. (2007) Very large format stereoscopic head-up display for the airport tower. In: In Proceedings of the Virtual Images Seminar, number 16. CNRS/Renault.

Figura 1-I: esempi di HMD. Da sinistra: Google Glass, Microsoft Hololens e Oculus Rift

Questi dispositivo di AR presentano alcune carenze[15]:

- Bassa risoluzione delle grafiche, a causa della miniaturizzazione dei display che vengono utilizzati. Nei display di tipo see-through, questo limite è destinato solo all'overlay sintetico, ovviamente, in quanto l'ambiente è visualizzato direttamente in trasparenza;
- Campo visivo limitato, a causa dei limiti delle tecnologie ottiche applicate;
- Problemi di percezione visiva, a causa della profondità costante dell'immagine sintetica. In particolare, per i ST-HMD, la differenza tra la profondità dell'immagine reale e la profondità percepita dell'immagine sintetica, forza gli occhi a continui cambiamenti di messa a fuoco o a rassegnarsi a percepire uno delle due immagini come sfocata.
- Calibrazione del dispositivo spesso difficoltosa e diversa per diversi utenti, per consentire un corretto posizionamento degli overlay.
- Isolamento dell'operatore rispetto all'ambiente circostante, a causa del campo visivo limitato che genera un effetto "paraocchi".
- Dipendenza dalla durate delle batterie, col rischio di perdere l'ausilio AR in momenti critici.

Si noti che la maggior parte di queste carenze sono ereditate dai limiti generali della tecnologia di visualizzazione degli HMD che, almeno fino all'introduzione di Microsoft Hololens® a Marzo 2016 al costo di 3000$, non garantiva dispositivi di qualità adeguata ad un costo accessibile.

[15] Bimber, O., Raskar, R. (2005) Spatial Augmented Reality: Merging Real and Virtual Worlds. A. K. Peters, Ltd., Natick, MA, USA.

I vantaggi principali degli HMD sono invece:

- Elevata portabilità, che consente di disporre di questo strumento in qualsiasi luogo senza dover intervenire sulla struttura;
- Costo relativamente contenuto, a patto di accontentarsi di una qualità limitata;
- Fruizione personale dei contenuti AR, che consente a diverse persone nello stesso ambiente di visualizzare informazioni diverse.
- Popolarità della tecnologia, usata anche in ambito consumer, che garantisce un continuo miglioramento della strumentazione e un costante abbassamento dei prezzi.

1.7. Spatial see-through display (SSTD)

Gli schermi fissi semitrasparenti (spatial see-through display, SSTD) sono sistemi installati all'interno dell'area di lavoro dell'operatore anziché essere indossati. Consentono, attraverso essi, di osservare sia il mondo reale, sia informazioni aggiuntive generate artificialmente.

Anche questi dispositivi si distinguono in Mixed-Video See-through (MV SSTD) e Optical See-through (O SSTD). Come per gli HMD, la differenza sostanziale tra le due tipologie è il modo in cui viene mostrata l'immagine reale, e di conseguenza il tipo di tecnologie e schermi utilizzati per la realizzazione: gli MV SSTD sfruttano uno schermo opaco tradizionale per mostrare un flusso video proveniente da una videocamera, mentre gli O SSTD si basano su schermi semitrasparenti, integrati con specchi, proiettori o tecnologie LED o LCD in modo che l'immagine reale arrivi non trattata all'utilizzatore.

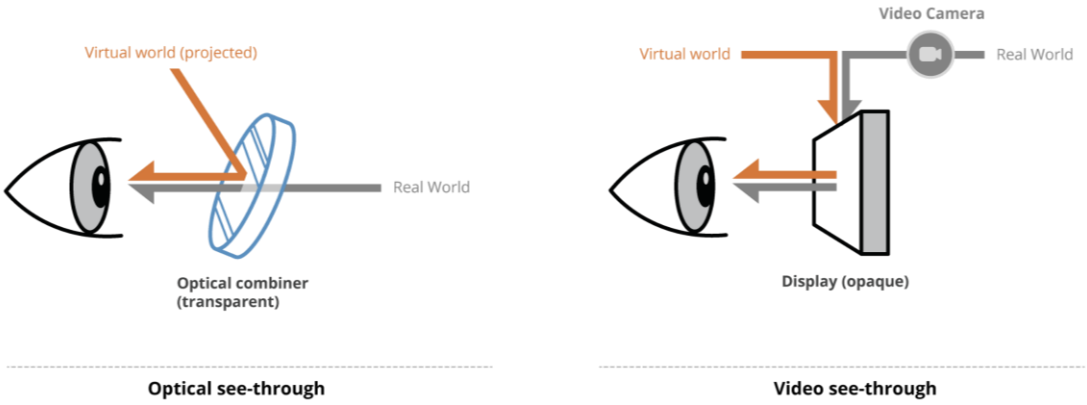


Figura 1-J: schemi di funzionamento di un Optical See-Through Display e di un Video See-Through Display. Niteesh Yadav, Prototypr.io.

Gli O SSTD sono spesso considerati gli "SSTD propriamente detti", a cui sovente è fatta coincidere l'intera categoria SSTD. Di seguito saranno trattati principalmente gli O SSTD e sarà fatto riferimento indistintamente alla nomenclatura SSTD e O SSTD intendendo gli O SSTD.

A questa tipologia di schermi appartiene la prima applicazione storica della realtà aumentata, quando a fine '800 veniva usa per creare effetti speciali in spettacoli teatrali tramite la "Pepper's ghost technique", in cui erano utilizzate lastre di vetro e particolari tecniche di illuminazione per far apparire o scomparire persone ed oggetti, renderli evanescenti o trasformare un oggetto in un altro.

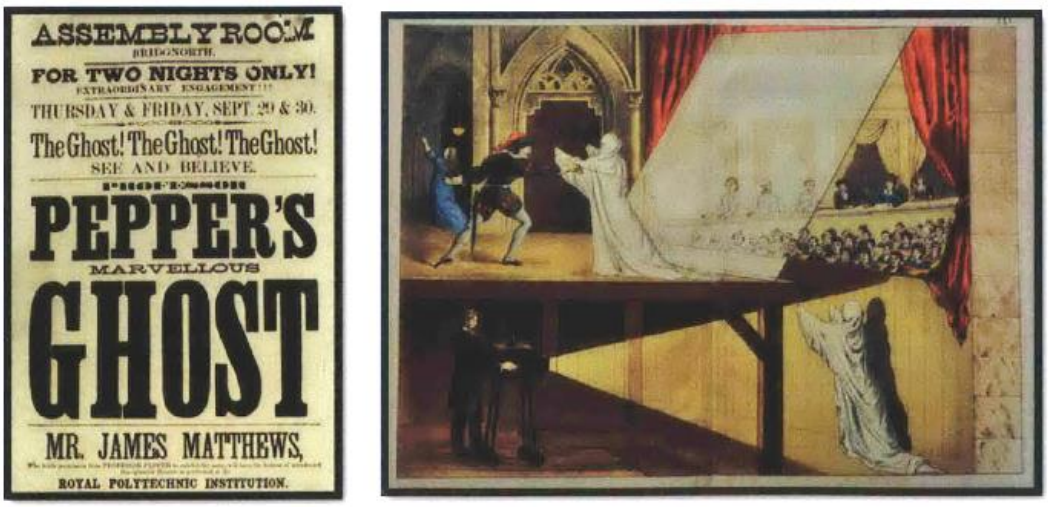


Figura 1-K: Spettacolo "Pepper's ghost" del 1870. Il mago combatte un fantasma virtuale. Sharma A.

L'illusione su cui si basava Pepper's Ghost era stata descritta per la prima volta da Giambattista della Porta, scienziato e filosofo napoletano del XVI secolo. Nel suo “Magia Naturalis” del 1584 descrive un'illusione intitolata "Come possiamo vedere in una camera cose che non ci sono"[16].

Più recentemente, gli SSTD si sono diffusi in applicazioni critiche, come visori per impieghi medici e HUD negli aerei da combattimento, che rappresentano probabilmente la loro espressione più nota e consolidata.

1.8. SSTD: vantaggi e problematiche aperte

Il principale vantaggio degli SSTD rispetto alle soluzioni indossabili risiede nella migliore ergonomia dello strumento, in quanto il peso e l’ingombro dei meccanismi del dispositivo non gravano sull’ utilizzatore, consentendo quindi una migliore libertà di movimento e un minore affaticamento.

Strettamente connessa a questa caratteristica è la mancanza di limiti di autonomia e ingombro: la strumentazione può essere connessa direttamente alla rete elettrica e l’utilizzo di tecnologie più performanti non è limitato dalle loro dimensioni. Questo ultimo aspetto è particolarmente rilevante per il sistema ottico che si occupa di collimare la messa a fuoco dell’immagine virtuale, per fare cui è necessario un certo spazio fisico per il percorso ottico.

[16] Sharma A. (2014) Augmented tools with transparent displays. Massachusetts Institute of Technology.

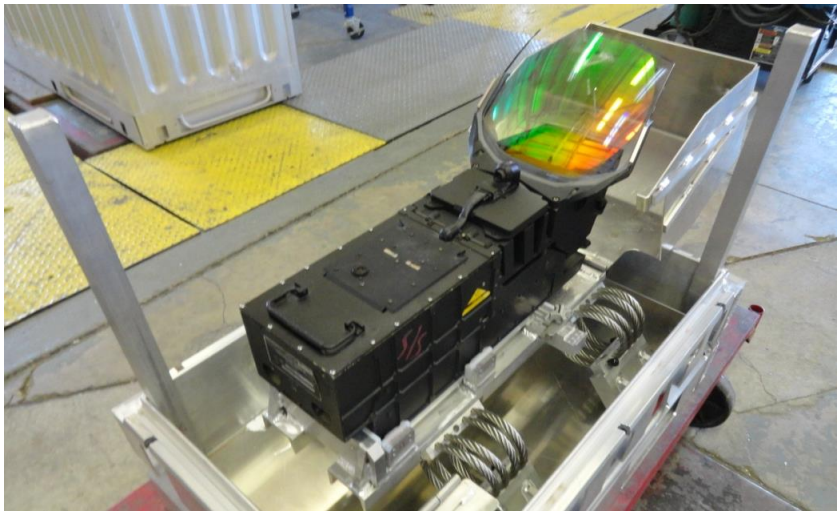


Figura 1-L: HUD di un F-15E. Le dimensioni esterne del container sono 102 cm di lunghezza, 66 cm di larghezza e 85 cm di altezza. www.wpafb.af.mil.

Un ulteriore vantaggio, a seconda del campo di uso, è la possibilità per più persone di usare contemporaneamente la stessa strumentazione, in quanto questa tecnologia non è personale, come invece sono, ad esempio, gli HMD. Questo rende possibile l'interazione e il confronto tra colleghi che osservano la stessa immagine, migliorando quindi la comunicazione e la sicurezza.

Rispetto agli HMD, i sistemi SSTD hanno dimensioni e costi maggiori, di conseguenza la loro diffusione è limitata. Parallelamente, anche la comunità di sviluppatori, le aziende produttrici ed il know-how relativo a questa tecnologia è minore rispetto a quello degli HMD.

Inoltre, un particolare sistema SSTD è concepito per essere installato in una specifica struttura in cui svolgere un compito preciso. Rispetto ai sistemi HMD o hand-held è quindi meno versatile, ed una eventuale modifica della destinazione d'uso avrebbe alti tempi e costi di aggiornamento e installazione.

Una sfida importante che introducono i sistemi SSTD è la miscelazione dell'immagine reale con quella virtuale, in quanto l'operatore ha una certa libertà di movimento rispetto al dispositivo, cosa che invece non avviene con i sistemi HMD. Questo può incidere molto sulla performance visiva della strumentazione, principalmente sulla collimazione dell'immagine reale con quella virtuale, sia

come posizione, ovvero la parallasse tra le due immagini, sia come distanza di messa a fuoco.

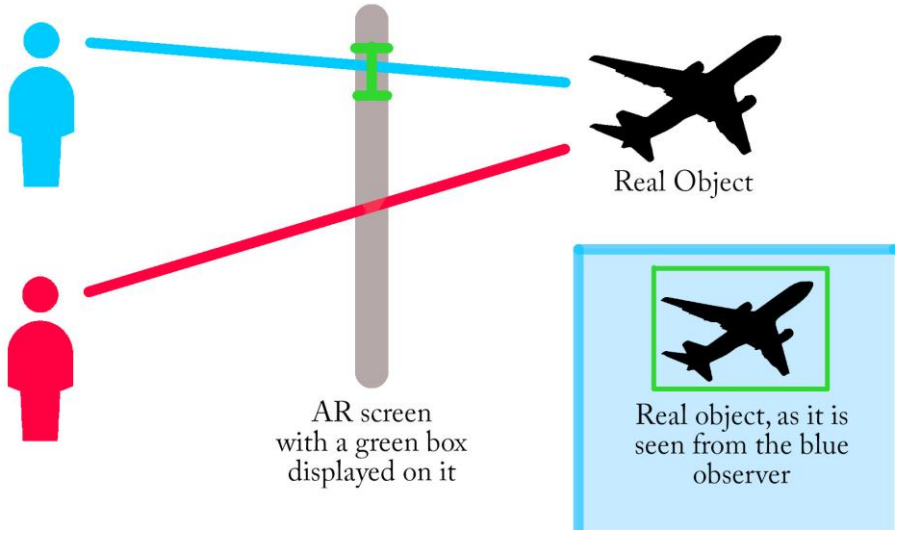


Figura 1-M: problema della parallasse per i sistemi AR SSTD: dalla sua posizione, l'osservatore blu vede correttamente il box verde intorno all'aereo, mentre l'osservatore rosso non lo vede.

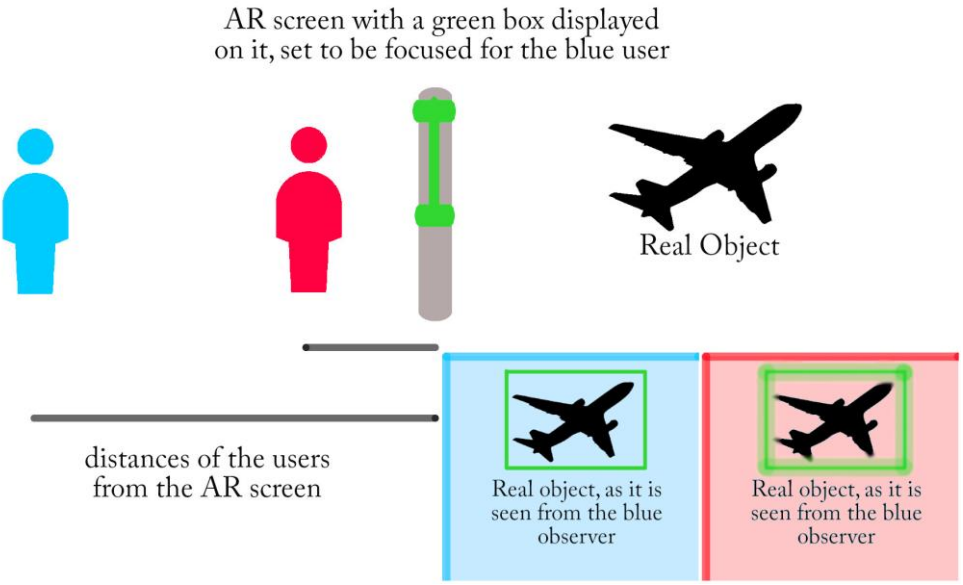


Figura 1-N: problema della messa a fuoco per i sistemi AR SSTD: dalla sua posizione, l'osservatore blu vede correttamente a fuoco il box verde intorno all'aereo, mentre l'osservatore rosso non può vedere entrambe le immagini a fuoco.

1.9. Generazione di contenuti virtuali: disparità binoculare o bioculare?

Optando per una soluzione di AR di tipo spatial see-through display, è importante valutare adeguatamente come visualizzare i contenuti sintetici perché si fondano al meglio con quelli reali, garantendo la migliore percezione visiva e senza affaticare o ingannare lo sguardo dell'operatore. È fondamentale in particolare cercare di soddisfare la percezione della distanza degli oggetti, in modo che un overlay virtuale e l'elemento reale a cui afferisce siano percepiti alla stessa profondità visiva dall'operatore.

In base al livello di qualità che si vuole raggiungere per l'applicazione specifica, si può valutare se generare contenuti virtuali con semplice disparità bioculare (stessa immagine fornita ad entrambi gli occhi, come per un normale schermo) o con disparità binoculare (immagini diverse per i due occhi, alla base della visione stereoscopica).

La percezione umana della distanza degli oggetti è basata sull'integrazione di varie sorgenti (indizi): occlusione, dimensione relativa, densità relativa, prospettiva aerea, altezza sul campo visivo, motion perspective, accomodazione, disparità binoculare, convergenza.

L'affidabilità di questi indizi dipende dalla distanza dell'osservatore. È consolidato in bibliografia il riferimento alla seguente[17]:

[17] Nagata, S. (1981), Visual depth sensitivities of various cues for depth perception. NHK Laboratories Note, 266, 1–11.

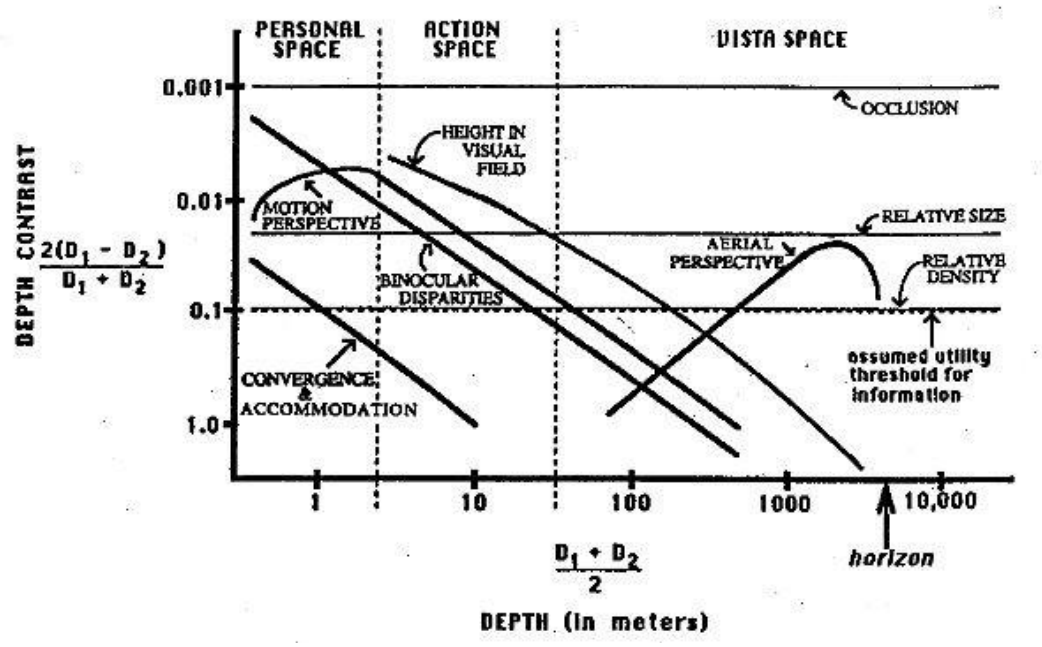


Figura 1-O: limiti di distinguibilità della profondità in funzione del Log della distanza dall'osservatore, da 0.5 a 5000 metri (Nagata, 1981).

E' inoltre descritto un sistema di ranking nel modo in cui la vista sfrutta i vari indizi in base alla distanza:

		Action space		
Source of information	Personal space	All sources	Pictorial sources	Vista space
1. Occlusion and interposition	1	1	1	1
2. Relative size	4	3.5	3	2
3. Relative density	7	6	4	4.5
4. Height in visual field and height in the picture plane	—[a]	2	2	3
5. Aerial perspective and atmospheric perspective	8	7	5	4.5
6. Motion perspective and motion parallax	3	3.5	—	6
7. Convergence	5.5	8.5	—	8.5
8. Accommodation	5.5	8.5	—	8.5
9. Binocular, disparity, stereopsis, and diplopia	2	5	—	7

[a] Dashes indicate data not applicable to source.

Figura 1-P: ranking delle sorgenti di informazione nei tre tipi di spazio visivo (Cutting e Vishton, 1995). I tre spazi sono: personale (0,5 - 1,5 m), d'azione (1.5 – 30 m), di vista (>30 m).

Emerge come la disparità binoculare abbia la più alta sensibilità (dopo l'occlusione) degli indizi di profondità a distanze inferiori ai 10 m, mentre è assodato come la sua efficacia sia scarsa a distanze superiori ai 30 m[18], ma sia comunque significativa almeno fino a 20 m[19]. Recenti ricerche[20] suggeriscono l'efficacia della disparità binoculare anche all'interno del vista space.

Effettivamente, la visione umana è molto sensibile alla stereoscopia, potendo individuare differenze di profondità con disparità di pochi secondi di arco[21], suggerendo che sia possibile distinguere un punto da uno all'infinito fino a 240 m[22].

Nonostante queste ultime evidenze, va ricordato quanto già detto, ovvero come la disparità binoculare perda di valore, seppur presente, rispetto agli altri indizi disponibili nel vista space. In presenza di stimoli multipli il dato stereoscopico è "sovrascritto" da altri più performanti alle lunghe distanze (eg.: dimensione relativa).

In riferimento ad ambienti di VR o AR, la presenza di stimoli in disaccordo può, però, generare fastidio. Inoltre, in certe condizioni alcuni indizi predominanti possono mancare (notte, oggetti simili ma con dimensioni diverse come aerei o navi), per cui il dato proveniente dalla disparità binoculare scala nuovamente la classifica degli indizi a cui fare riferimento.

Risulta perciò che, se possibile, è preferibile generare immagini virtuali con disparità binoculare anche per oggetti a grande distanza.

---

[18] Nagata, S. (1991), How to reinforce perception of depth in single two-dimensional pictures, in S.R. Ellis, M. Kaiser and A. Grunwald, eds, 'Pictorial communication in virtual and real environments', Taylor & Francis, pp. 527–544.
[19] Allison, R., Gillam, B. and Vecellio, E. (2009), 'Binocular depth discrimination and estimation beyond interaction space', Journal of Vision 9(1), 10:1–14.
[20] Palmisano, S., Gillam, B., Govan, D. G., Allison, R. S. and Harris, J. M. (2010), 'Stereoscopic perception of real depths at large distances', Journal of vision 10(6), 19:1–16.
[21] Howard, H. J. (1919). A test for the judgment of distance. Transactions of the American Ophthalmological Society, 17, 195–235.
[22] Helmholtz, H. von (1909/1925), Physiological optics, vol. 3, trans. J. P. C. Southall. Optical Society of America.

1.10. Eye-tracking e Head-tracking

Fornire contenuti in disparità binoculare richiede la conoscenza costante della posizione dell'operatore, in particolare della sua testa o, meglio ancora, dei suoi occhi, in modo da usarli come punti macchina virtuali per la generazione dei contenuti sintetici. Per ottenere ciò, ci si avvale di tecniche di Head-tracking ed Eye-tracking che, tramite l'uso di videocamere, sensori infrarossi, target applicati direttamente all'operatore e software di riconoscimento delle immagini, riescono a monitorare la posizione di testa o occhi all'interno del frame 2D o dello spazio 3D della stanza. Ovviamente, un sistema di Head-tracking restituirà un solo punto come risultato e, a meno di particolari tecniche, non può consentire la generazione di contenuti binoculari.

Per quanto visto nel paragrafo precedente, in certe situazioni può risultare importante la disponibilità dell'indizio stereoscopico di percezione visiva, pertanto un sistema di Eye-tracking consente di generare contenuti AR più raffinati e con una migliore performance del sistema, riducendo l'affaticamento dell'operatore e migliorandone la prestazione.

2. SVILUPPO DI UN ALGORITMO DI EYE-TRACKING CON SENSORE MICROSOFT KINECT®

Per ottenere la misura real-time della posizione degli occhi di un operatore si è scelto di sfruttare il dispositivo Microsoft Kinect®. Tale strumento dispone di caratteristiche hardware e API di interazione che consentono, tramite lo sviluppo di un opportuno software, di monitorare le coordinate spaziali di vari punti del viso di una persona. Tali valori, codificati come vettori posizionali, possono essere quindi inviati a un'interfaccia di comunicazione per essere elaborati dal sistema che genera i contenuti AR. Lo sviluppo del software di cui sopra è la parte centrale di questo lavoro, e sarà svolto usando il linguaggio di programmazione C# e markup XAML, producendo un applicativo Windows Presentation Foundation (WPF).

2.1. Microsoft Kinect® 2 for Windows

Il dispositivo di riferimento con cui è realizzata questa ricerca è Microsoft Kinect®, reso disponibile a partire da Novembre 2010. È un accessorio progettato per l'intrattenimento videoludico, sensibile al movimento e in grado di eseguire comandi impartiti gestualmente e oralmente, capace di identificare la posizione di un utente all'interno di uno spazio 3D. Una seconda versione migliorata, Kinect ONE®, anche noto come Kinect 2.0® o Kinect V2®, è stata messa in commercio nel 2013. Quest'ultima versione è quella usata per questa ricerca e l'unica a cui si farà riferimento in seguito.

Figura 2-A: Microsoft Kinect(R) V1 eV2. programarfacil.com.

Kinect 2.0® è dotato di una videocamera a tempo di volo (time of flight camera, TOF-camera), uno strumento in grado di misurare la distanza tra la videocamera e gli oggetti inquadrati, stimando il tempo impiegato da un impulso luminoso a percorre il tragitto videocamera-oggetto-videocamera, come nei sistemi RADAR. La TOF-camera di Kinect 2.0® è di tipo ad infrarossi e ha una risoluzione di 512×424 pixel, affiancata da un proiettore ad infrarossi che genera un pattern di riferimento per la videocamera.

Figura 2-B: schermata della flusso video 3D, con informazioni di profondità, di Kinect(R). A destra è visualizzata l'immagine frontale, a sinistra una ricostruzione 3D in real-time. Le zone colorate di blu sono quelle più distanti, quelle verdi le più vicine. developer.microsoft.com.

È presente anche una videocamera RGB standard con risoluzione 1080p a 30 fps, utilizzabile per l'acquisizione di flussi video a colori in alta definizione. Al comparto video è affiancato un array di 4 microfoni direzionali in grado di rilevare la posizione delle sorgenti sonore e cancellare eco e rumori.

Grazie alle sue API dedicate, Kinect 2.0® è in grado di distinguere fino a sei utilizzatori contemporaneamente, di identificarne sommariamente le parti del corpo e giunture (Junction point, joint) e i gesti, come il movimento di un arto o anche una mano aperta piuttosto che chiusa.

Disponibile dal 2014 al prezzo di 149$ come dispositivo indipendente dalla console X-Box One® di Microsoft, ha conosciuto una grande diffusione sia tra i videogiocatori, sia tra ricercatori e sviluppatori, proprio grazie alle sue grandi capacità a fronte di un costo estremamente contenuto.

Microsoft, inoltre, ha supportato e favorito il fiorire di un'animata comunità di sviluppatori di applicazioni per Kinect®, in primis mettendo a disposizione la suite "Kinect per Windows SDK 2.0", che comprende le API software e la piattaforma "Kinect Studio", assieme a diversi esempi, per impratichirsi nella realizzazione di programmi che sfruttano l'hardware Kinect®. Inoltre, Microsoft coordina ed assiste una rete di sviluppatori indipendenti attraverso il suo portale Microsoft Developer Network, msdn.microsoft.com, in cui si trovano sezioni dedicate a Kinect® ed è possibile mettersi in contatto con altri sviluppatori per ricevere aiuto su problematiche di sviluppo o idee per nuove applicazioni. Accanto ai canali ufficiali di Microsoft, sono nate e cresciute comunità e portali di riferimento indipendenti, alcuni specifici per Kinect®, come developkinect.com, altri con sezioni dedicate a Kinect®, come stackoverflow.com.

Tutto questo ha reso Kinect® estremamente popolare e supportato, nonostante la sua commercializzazione sia cessata nel 2017, per cui è una piattaforma di sviluppo ottimale per la ricerca o la creazione di concept design a basso costo.

2.2. Approccio utilizzato

Come già citato, Kinect® è in grado di distinguere una persona da uno sfondo, di individuarne le parti del corpo e comprenderne alcuni gesti. Tramite opportune API, Kinect può riconoscere alcuni tratti biometrici del volto di una persona. In particolare, Kinect 2.0® può tracciare 1347 punti di riferimento facciale nello spazio 3D. I punti sono enumerati e visualizzabili ed è possibile risalire alle loro coordinate spaziali.

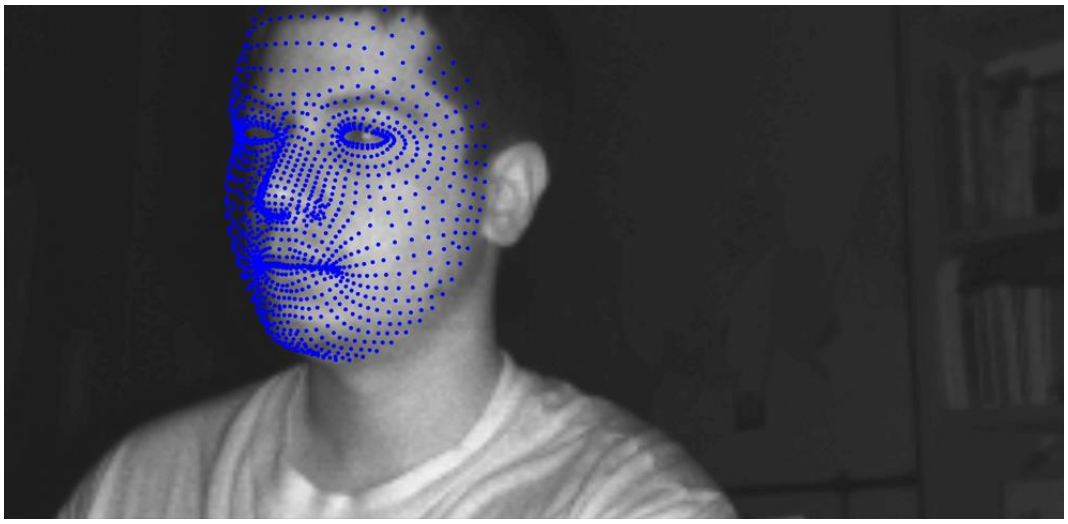


Figura 2-C: i punti di riferimento individuati da Kinect(R) sovrapposti all'immagine reale. codeproject.com.

Tra questi riferimenti sono presenti anche vari punti del contorno degli occhi, ma non il centro degli occhi stessi, che è il dato ricercato. Per ottenere le coordinate del centro degli occhi si può trovare il punto medio tra due bordi opposti. Di seguito si scriverà “occhio” per riferirsi al punto medio, ovvero il centro dell’occhio, dove il contesto non generi confusione.

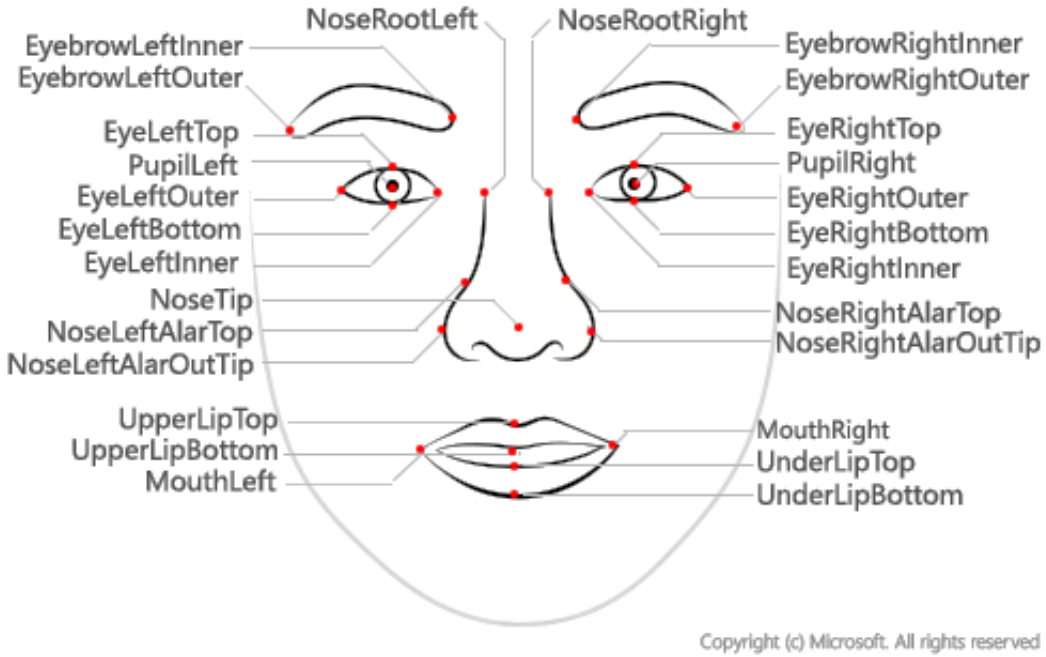


Figura 2-D: mappa parziale dei punti di riferimento facciali. PupilLeft e PupilRight non sono rilevabili nativamente da Kinect. microsoft.com

La scelta iniziale è ricaduta sull'angolo esterno e quello interno dell'occhio, poiché sono i punti più evidenti e distinti dagli altri del bordo, ed inoltre sono quelli che si trovano alla maggiore distanza l'uno dall'altro, consentendo quindi di mantenere un basso errore relativo sulla misura calcolata del centro.

Purtroppo, durante le varie prove del software, si è evidenziato come tali punti siano soggetti ad essere "persi" dal sistema di tracciamento, perché in certe orientazioni del viso, essi vengono nascosti dal naso o dalla rotondità cranica.

Per mitigare questo inconveniente, si è quindi optato per i due punti medi inferiore e superiore dell'occhio, che hanno un'esposizione maggiore degli altri perché in posizione più avanzata. Questo consente a tali riferimenti di essere individuabili anche con importanti rotazioni del capo.

Figura 2-E: diversi angoli di rotazione e inclinazione del capo. Si noti come gli angoli degli occhi siano spesso nascosti, mentre i bordi superiore e inferiore resistano a rotazioni più pronunciate. visagetechnologies.com, testo aggiunto dall'autore.

In realtà, il centro degli occhi misurato come media degli angoli oculari è in una posizione diversa rispetto a quello misurato come media del bordo superiore e inferiore, questo perché l'occhio umano non è simmetrico rispetto al suo centro, o meglio, di fatto non esiste un centro (se non in forma baricentrica integrale) geometrico dell'occhio umano.

Figura 2-F: differenza di posizione tra il centro oculare calcolato a partire dagli angoli (verde) o dai bordi superiore e inferiore.

La differenza è comunque di pochi millimetri a fronte dei metri che sono restituiti come distanze degli occhi da Kinect. In sede di creazione dei contenuti AR, si può tenere conto di questa discrepanza per perfezionare le immagini virtuali generate.

I punti individuati corrispondenti agli occhi sono, ovviamente, due, mentre è noto che per generare un vettore nello spazio tridimensionale servano 3 punti con le rispettive coordinate. Tale vettore è importante per avere un'informazione della direzione in cui stia guardando l'operatore, specialmente per quanto riguarda l'inclinazione verso l'alto e la rotazione panoramica, in modo da poter generare contenuti AR coerenti con la direzione visiva. Ad esempio, alcune informazioni come l'orario, temperatura e messaggi di allarme potrebbero essere riportate in modo fisso rispetto al punto di vista dell'osservatore, mentre altre essere registrate con l'ambiente reale, come un box intorno ad un aereo. Inoltre, si potrebbe evitare di generare i contenuti afferenti alle zone dove l'utilizzatore non sta in quel momento guardando, risparmiando carico computazionale e mantenendo un ambiente libero per gli altri operatori presenti.

In teoria, il terzo punto può essere scelto arbitrariamente tra tutti i riferimenti facciali. In pratica, conviene scegliere un punto che sia facilmente distinguibile e ben esposto. La scelta migliore ricade sulla punta del naso, che ha anche il vantaggio di essere già centrata rispetto agli occhi e di essere già direzionata, a meno dell'inclinazione in basso, come la direzione visiva.

Il software prodotto sarà suddiviso in tre parti: il listato in C# si occuperà dell'effettiva interazione con Kinect®, della lettura dei flussi di dati e della loro elaborazione, consegnando poi ad un'interfaccia XAML il compito di mostrare i dati e ad un socket di intercomunicazione l'invio degli stessi al sistema AR.

2.3. Strumenti e sviluppo

Lo sviluppo software e i relativi test sono svolti all'interno del Laboratorio di Realtà Virtuale (V-Lab) della Seconda Facoltà di Ingegneria di Forlì, ovvero la stessa struttura in cui è condotta parte dello sviluppo di RETINA. È ivi disponibile

un teatro di realtà virtuale a schermi multipli CAVE e i dispositivi necessari allo sviluppo dell'applicativo oggetto di questa ricerca, ovvero 2 esemplari di Microsoft Kinect 2.0® e 3 postazioni PC ad alte prestazioni.

Figura 2-G: V-lab con CAVE in funzione.

Si sono inizialmente verificati diversi problemi di installazione tra Kinect® e i PC. Microsoft mette a disposizione Kinect Configuration Verifier, un software per controllare se le caratteristiche del PC in uso siano compatibili con Kinect®.

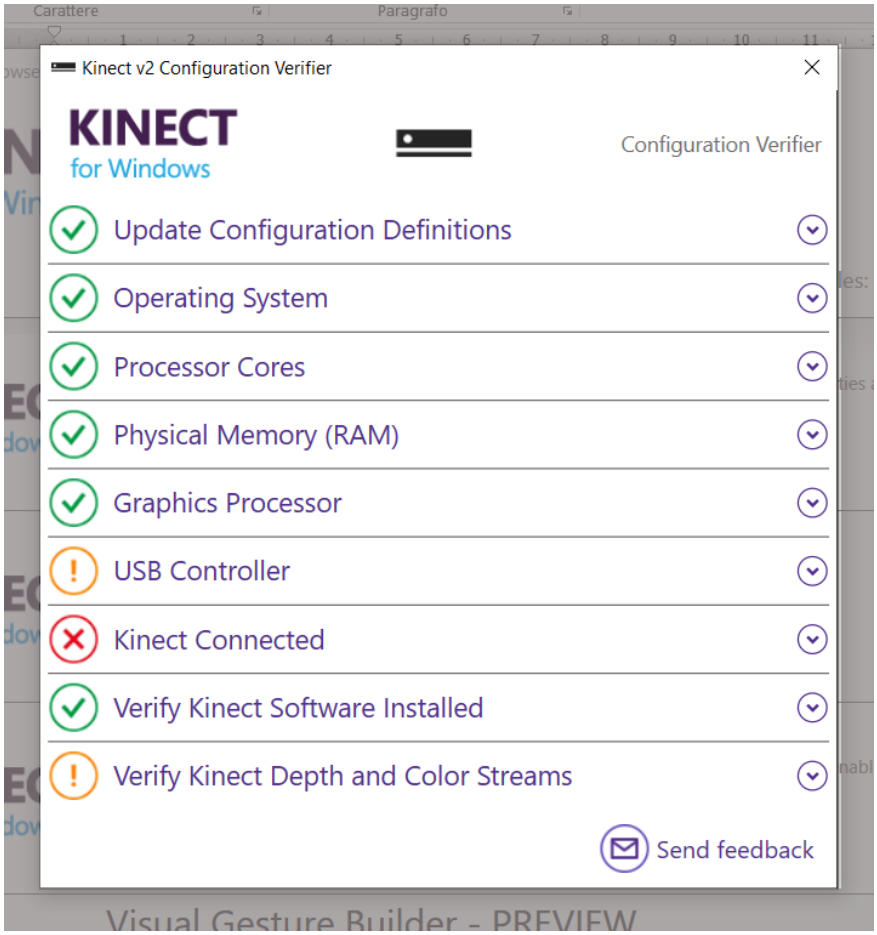


Figura 2-H: finestra di Kinect Configuration Verifier che mostra quali parametri della configurazione in uso soddisfino quelli necessari per l'utilizzo di Kinect(R).

Il dispositivo è risultato essere molto sensibile a:

- Tipo di controller USB 3.0 presente nel pc (chip D720202 vs D720200);
- Driver di controllo della scheda USB 3.0 (v0.96 vs v1.00);
- Architettura del PC (AMD vs Intel);
- Sistema operativo (Windows 8 vs Windows 8.1 vs Windows 10);
- Porta USB 3.0 utilizzata tra le varie disponibili sulla stessa scheda;
- Aggiornamenti installati.

Nonostante siano presenti certe indicazioni sulle specifiche richieste, spesso queste si sono mostrate in disaccordo con i fatti e il raggiungimento di una configurazione funzionante è avvenuto sostanzialmente per tentativi, assemblando diversi PC con diverse schede USB 3.0 e installando driver di versioni differenti.

Sono state provate diverse configurazioni hardware e software, portando alla conclusione che i PC basati su architettura Intel accettano Kinect® senza problemi, nella maggior parte dei casi, mentre le architetture AMD hanno bisogno di un particolare chipset della scheda USB con una particolare versione del driver, a seconda del caso, più recente o più datata, in modo del tutto non prevedibile. Il passaggio al sistema operativo Windows 10 ha ridotto i problemi per alcune configurazioni basate su AMD, mentre per altre non ha portato a cambiamenti. In alcuni assemblaggi, Kinect funziona su una porta USB 3.0 ma non su quella accanto.

La configurazione finale utilizzata per la maggior parte dello sviluppo è stata la seguente:

- Processore AMD Phenom™ II X4 965 3.41 GHz;
- Memoria RAM installata 8.0 GB
- Scheda grafica Nvidia GeForce GTX 660 Ti con 3072 MB di memoria;
- Scheda USB 3.0 dedicata Renesas con driver 0.96 (Microsoft);
- Sistema Operativo Windows 10 Pro 64 bit;
- Primo ingresso USB 3.0 della scheda.

Il software è stato scritto tramite l'ambiente di sviluppo integrato (Integrated Development Environment, IDE) Microsoft Visual Studio, che supporta programmazione multi-linguaggio (C#, Visual Basic, C++…) e multi-piattaforma hardware (PC, mobile, console…). Si basa sul framework .NET e può essere integrato da numerose estensioni ad-hoc per specifici compiti. Visual Studio possiede un debugger interno per la gestione degli errori e una sezione dedicata all'analisi prestazionale del codice. È disponibile gratuitamente e a sua installazione è scalabile in base alle necessità.

All'applicativo sviluppato è stato dato il nome "KET – Kinect HD Eye Tracker", per il quale è stata anche disegnata un'icona per il file .exe di lancio.

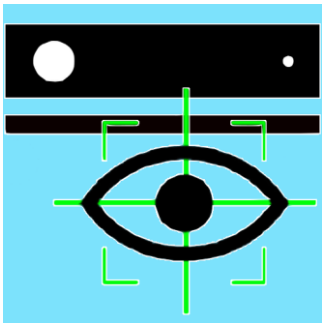


Figura 2-I: Icona realizzata per il launcher.

2.4. Linguaggio di programmazione: C#

C# è un linguaggio di programmazione orientato agli oggetti nato nel 2000, che trae le sue origini dai vantaggi di C++ e Java, da cui ha ereditato anche parte della struttura e sintassi. È standardizzato ECMA International come ECMA-334 e da ISO/IEC come standard iso/IEC 23270.

È un linguaggio moderno, particolarmente performante per la scrittura di software longevi e aggiornabili costituiti da pacchetti di funzionalità tra di loro indipendenti che si identificano tramite proprietà, metodi, eventi ed attributi[23]. C# mette a disposizione diverse funzionalità che aiutano nella progettazione di applicativi efficienti, quali la gestione delle eccezioni per il rilevamento degli errori e il Garbage Collection per liberare la memoria da variabili inutilizzate e aumentare le prestazioni.

È altresì presente un controllo nativo delle versioni di C# nelle librerie del progetto in sviluppo, in modo che l'esecuzione su macchine con differenti versioni possano essere gestite senza incorrere nel blocco del software. A Maggio

[23] AA.VV (2017), "Introduzione a C#", Microsoft.com.

2018 la versione corrente è la 7.3, ovvero la decima tra le versioni principali, a cui corrisponde la versione 4.7.2 di .NET framework, rilasciate entrambe con Visual Studio 2017 Ver 15.7.

Le caratteristiche salienti della sintassi del linguaggio C# sono:

- I nomi sono case-sensitive, ovvero valutano differentemente lettere minuscole e maiuscole;
- Ogni specifica, ovvero un'istruzione, deve terminare con il punto e virgola;
- Le specifiche sono raggruppate logicamente, nel listato, tramite parentesi graffe;
- Le specifiche sono, di norma, contenute in metodi (cioè funzioni), a loro volta raggruppati in classi, le quali sono infine riunite in namespace.

Un esempio della sintassi procedurale del linguaggio C# è visibile dal classico codice "Hello World":

```
using System;

class Hello
{
    static void Main() {
        Console.WriteLine("Hello, World");
    }
}
```

Figura 2-J: Listato del codice "Hello World" in C#.

"System" è lo spazio dei nomi a cui fa riferimento la direttiva "using". System contiene la classe "Console". Avendo usato "using System", si può scrivere "Console.Writeline" anziché "System.Console.Writeline". La classe "Hello" ha il solo membro "Main", che è un metodo, modificato con la dichiarazione "static", che indica di operare senza riferirsi ad un determinato oggetto. Main è anche, per convenzione, il punto di ingresso di un programma. L'output è prodotto dal metodo "Writeline" della classe "Console" e consiste nel testo "Hello, World".

2.5. Sviluppo del codice sorgente (in appendice)

Il codice sorgente in C# che è stato sviluppato è consultabile per intero in appendice 1. In questo paragrafo ne sono descritti i punti salienti e le scelte effettuate durante la sua stesura. Saranno riportati e commentati i blocchi di codice riguardanti l'individuazione, tracciamento e calcolo dei punti di riferimento cercati e le relative istruzioni di inizializzazione e gestione dei metodi. Le parti che riguardano la gestione dei dati trovati e la veste grafica dell'interfaccia saranno invece trattate nei rispettivi paragrafi. All'interno del codice sono presenti blocchi disattivati, che si presentano quindi come commenti, che sono però mantenuti per attivare o disattivare all'occorrenza funzioni alternative del software (ad esempio, la scrittura dei dati all'interno di un file .txt).

Si precisa che i commenti all'interno del listato C# sono generalmente resi in colore verde e seguono la seguente sintassi:

```
2   /* questo è un commento,
3      eventualmente lungo più righe */
4
5   // questo è un commento su una riga sola
6
```

Figura 2-K: sintassi dei commenti in C#.

Si tenga in considerazione che, per pura convenzione, il carattere underscore (_), presente davanti al nome di alcune variabili, denota che esse siano di tipo "private", nella maggior parte delle occorrenze.

Per accedere ai flussi di dati di Kinect® e a quelli afferenti l'identificazione dei riferimenti facciali, è necessario accedere ai rispettivi namespace;

```
23  using Microsoft.Kinect;
24  using Microsoft.Kinect.Face;
```

Figura 2-L: namespace specifici per Kinect(R).

Si possono quindi inizializzare il Kinect stesso e le variabili che saranno utilizzate per la ricezione e lettura dei flussi provenienti da Kinect®:

```
34          // Provides a Kinect sensor reference.
35          private KinectSensor _sensor = null;
36
37          // Acquires body frame data.
38          private BodyFrameSource _bodySource = null;
39
40          // Reads body frame data.
41          private BodyFrameReader _bodyReader = null;
```

Figura 2-M: inizializzazione delle variabili.

In particolare, la variabile privata _sensor, di tipo KinectSensor serve per accedere al dispositivo, BodyFrameSource attiva il flusso dati relativo al riconoscimento dei corpi, BodyFrameReader legge i dati presenti nello stream. Il paradigma XxYySource e XxYyReader si ripete per il tipo HighDefinitionFaceFrame, che consente di individuare i riferimenti facciali, supportato da FaceAlignmen e FaceModel, tramite cui, nota l'inclinazione della testa, può essere generata un modello standard iniziale del volto su cui poi è perfezionata la mappa dei punti di riferimento facciali (indicati nel listato come "vertices").

È anche inizializzato un enumeratore (lista) di ellissi "_points", che sarà poi riempita man mano che saranno individuati i vertices. Ogni ellisse sarà associata ad un vertice e diventerà un punto visibile sull'interfaccia, nella posizione corrispondente al vertice reale:

```
61          // Used to display 1,000 points on screen.
62          private List<Ellipse> _points = new List<Ellipse>();
```

Figura 2-N: generazione dell'insieme delle ellissi.

È quindi configurato il socket di trasmissione (di cui si tratterà più specificatamente nel paragrafo 2.8) e inizializzati i componenti:

```
88              //initializing KinectSensor
89              _sensor = KinectSensor.GetDefault();
90
91              if (_sensor != null)
92              {
93                  // Listen for body data
```

Figura 2-O: associazione di _sensor al Kinect(R) disponibile e verifica che sia effettivamente presente.

L'istruzione alla riga 91 si preoccupa di verificare che vi sia effettivamente un sensore Kinect collegato e attivo, altrimenti (_sensor = null) non si giungerà all'esecuzione dello script.

Vengono quindi attivati i sistemi di lettura dei flussi "Body" e "HighDefinitionFace", basati su un sistema di frame successivi come un flusso video. I frame sono resi disponibili dal "Source", letti dal "Reader" e aggiunti uno dopo l'altro (+=) all'interno di una variabile privata _XxYyReader:

```
92              {
93                  // Listen for body data.
94                  _bodySource = _sensor.BodyFrameSource;
95                  _bodyReader = _bodySource.OpenReader();
96                  _bodyReader.FrameArrived += BodyReader_FrameArrived;
97
```

Figura 2-P: passaggio dei valori dallo stream "Source", letto dal "Reader" e implementato ad ogni nuovo frame arrivato nella variabile "_bodyReader.FrameArrived".

Viene quindi effettivamente attivato Kinect tramite il comando:

```
110
111                 // Start tracking!
112                 _sensor.Open();
113             }
```

Figura 2-Q: accensione del sensore.

Il flusso dati HighDefinitionFace è sottoposto al controllo dell'evento FrameArrived e della verifica che il viso sia effettivamente tracciato. Verificate queste condizioni, i vertices sono aggiornati con la funzione "UpdateFacePoints()":

```
213         private void FaceReader_FrameArrived(object sender,
              HighDefinitionFaceFrameArrivedEventArgs e)
214         {
215             using (var frame = e.FrameReference.AcquireFrame())
216             {
217                 if (frame != null && frame.IsFaceTracked)
218                 {
219                     frame.GetAndRefreshFaceAlignmentResult(_faceAlignment);
220                     UpdateFacePoints();
221                 }
222             }
223         }
```

Figura 2-R: Controllo dell'evento "FrameArrived", dell'effettiva presenza di frame e dello status di tracciamento del viso.

Il metodo "UpdateFacePoints()" è la parte che effettivamente si occupa di restituire le posizioni cercate di occhi e punta del naso. È sviluppato quindi con un'istruzione di controllo sul contenitore dei dati "_faceModel" e riempie l'array "vertices" con i valori in arrivo dal flusso dati dei punti di riferimento facciali:

```
253         private void UpdateFacePoints()
254         {
255
256
257             if (_faceModel == null) return;
258
259             var vertices = _faceModel.CalculateVerticesForAlignment
                  (_faceAlignment);
```

Figura 2-S: Riempimento dell'array "vertices" con i dati provenienti dallo stream.

In "vertices" sono contenuti tutti i landmarks facciali, ognuno con un suo numero di riferimento. Alcuni di questi punti hanno, oltre al numero indiciale, un nome proprio descrittivo per agevolarne l'accesso all'interno dell'enumeratore "HighDetailFacePoints" (Appendice 2, tabella 2).

Ci si addentra quindi all'interno del contenitore "vertices" da cui si estraggono i punti cercati. Questi si presentano in forma di vettori (es. "NoseTip") di componenti X, Y, Z, (es. "NoseTip.X") che sono quindi scomposti e convertiti ognuno in 3 variabili di tipo "Single" (es. "NoseTipX", con lettera direzionale maiuscola), a loro volta successivamente copiate in forma di stringhe (es.

"NoseTipx", con lettera direzionale minuscola) con 4 decimali, per essere inviate alla console e al socket di intercomunicazione:

```
267             //NoseTip position
268             var NoseTip = vertices[(int)HighDetailFacePoints.NoseTip];
269             //convert to single
270             Single NoseTipX = NoseTip.X;
271             Single NoseTipY = NoseTip.Y;
272             Single NoseTipZ = NoseTip.Z;
273
274             string NoseTipx = NoseTipX.ToString("0.0000");
275             string NoseTipy = NoseTipY.ToString("0.0000");
276             string NoseTipz = NoseTipZ.ToString("0.0000");
```

Figura 2-T: estrazione delle coordinate della punta del naso.

Per gli occhi, il metodo è analogo, a cui è aggiunta la parte di operazioni relative al calcolo della posizione centrale come media dei due bordi superiore e inferiore, tanto per l'occhio destro quanto per il sinistro:

```
310             //EyeRight position
311
312             //Catch outer eye mid top position
313             var RighteyeMidtop = vertices[(int)
                  HighDetailFacePoints.RighteyeMidtop];
314             //convert to single
315             Single RighteyeMidtopX = RighteyeMidtop.X;
316             Single RighteyeMidtopY = RighteyeMidtop.Y;
317             Single RighteyeMidtopZ = RighteyeMidtop.Z;
318
319             //Catch inner eye mid bottom position
320             var RighteyeMidbottom = vertices[(int)
                  HighDetailFacePoints.RighteyeMidbottom];
321             //convert to single
322             Single RighteyeMidbottomX = RighteyeMidbottom.X;
323             Single RighteyeMidbottomY = RighteyeMidbottom.Y;
324             Single RighteyeMidbottomZ = RighteyeMidbottom.Z;
325
326             //Calculate eye position as a mean between top and bottom eye
                  mid
327             Single RighteyeX = (RighteyeMidbottomX + RighteyeMidtopX) / 2;
328             Single RighteyeY = (RighteyeMidbottomY + RighteyeMidtopY) / 2;
329             Single RighteyeZ = (RighteyeMidbottomZ + RighteyeMidtopZ) / 2;
330
331             string Righteyex = RighteyeX.ToString("0.0000");
332             string Righteyey = RighteyeY.ToString("0.0000");
333             string Righteyez = RighteyeZ.ToString("0.0000");
```

Figura 2-U: estrazione ed elaborazione delle coordinate degli occhi.

Oltre ai valori base delle posizioni di naso e occhi, è anche elaborato il valore di distanza interoculare, utile per avere un riferimento immediato per la generazione di contenuti stereoscopici. È anche una valida e rapida verifica della bontà della misura effettuata (come trattato nel paragrafo di test e risultati). È posta particolare cura alla scelta degli operatori di calcolo, in modo da richiedere meno lavoro computazionale che, date le alte frequenze di aggiornamento dei dati, è sempre un carico rilevante anche per operazioni relativamente semplici:

```
...thDrEyeTracking\KinectHDFEyeTracking\MainWindow.xaml.cs                    8
340             //Calculate interocular distance
341             Single EyedistanceX = RighteyeX - LefteyeX;
342             Single EyedistanceY = RighteyeY - LefteyeY;
343             Single EyedistanceZ = RighteyeZ - LefteyeZ;
344             //"S" denotes "square": for computational cost it's better to
                  use the square distance
345             //and calculate the squareroot only if need.
346             //for the same reason, it's better to calculate a square as a
                  product instead of as a power 2.
347             Single EyedistanceS = EyedistanceX * EyedistanceX + EyedistanceY
                   * EyedistanceY + EyedistanceZ * EyedistanceZ;
348             //extract squareroot
349             double EyedistanceR = Math.Sqrt((double)EyedistanceS);
350             //convert to string
351             string Eyedistancer = EyedistanceR.ToString("0.0000");
352
```

Figura 2-V: Calcolo della distanza interoculare

Ottenuti tutti i valori in tempo reale, è quindi necessario mostrarli sull'interfaccia, come sarà esposto nel relativo paragrafo. Qui è esposta solo la parte di codice relativo. Basti sapere che si vuole rappresentare una riproduzione 3d del viso completo formata da punti blu per i punti generici, gialli i punti ricercati per gli occhi. Sarà anche mostrata una rappresentazione ortogonale dello spazio intorno a Kinect® con le posizioni dei due occhi in tempo reale. Per il migliaio di punti del volto 3D:

```
381
382             if (vertices.Count > 0)
383             {
384                 if (_points.Count == 0)
385                 {
386                     for (int index = 0; index < vertices.Count; index++)
387                     {
388                         if (index == 1090 || index == 241 || index == 1104
                        || index == 731)
389                         {
390                             Ellipse ellipse = new Ellipse
391                             {
392                                 Width = 4.0,
393                                 Height = 4.0,
394                                 Fill = new SolidColorBrush(Colors.Yellow)
395                             };
396                             _points.Add(ellipse);
397                         }
398
399                         else
400                         {
401                           Ellipse ellipse = new Ellipse
402                           {
403                                Width = 1.0,
404                              Height = 1.0,
405                              Fill = new SolidColorBrush(Colors.Blue)
406                            };
407                             _points.Add(ellipse);
408                         }
409
410                     }
411
412                     foreach (Ellipse ellipse in _points)
413                     {
414                         canvas.Children.Add(ellipse);
415                     }
416                 }
417
```

Figura 2-W: generazione dell'insieme di ellissi gialle e blu da mostrare nella finestra del software.

Le ellissi associati ai vertici 3D sono quindi riportati alle coordinate dello spazio RGB 2D e inviati al canvas dell'interfaccia:

```
418                 for (int index = 0; index < vertices.Count; index++)
419                 {
420
421                     CameraSpacePoint vertice = vertices[index];
422                     DepthSpacePoint point =
                    _sensor.CoordinateMapper.MapCameraPointToDepthSpace
                    (vertice);
423
424                     if (float.IsInfinity(point.X) || float.IsInfinity
                    (point.Y)) return;
425
426                     Ellipse ellipse = _points[index];
427
428                     Canvas.SetLeft(ellipse, point.X);
429                     Canvas.SetTop(ellipse, point.Y);
```

Figura 2-X: conversione delle coordinate delle ellissi dallo spazio 3D a quello 2D dello schermo.

Per quanto riguarda le viste ortogonali della posizione degli occhi nello spazio intorno a Kinect, si utilizzano direttamente i valori trovati di RighteyeX, RighteyeY, RighteyeZ, LefteyeX, LefteyeY, LefteyeZ opportunamente scalati di un valore 50 per riempire al meglio lo spazio dell'interfaccia e associati a due coppie di ellissi verde e rosso. Per la mappa XY:

```
434             //draw eyes on mapXY
435             mapXY.Children.Clear();
436             Ellipse eyeRxy = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Green };
437             Ellipse eyeLxy = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Red };
438
439             Canvas.SetTop(eyeRxy, 145 - 50 * RighteyeY);
440             Canvas.SetLeft(eyeRxy, 180 + 50 * RighteyeX);
441             Canvas.SetTop(eyeLxy, 145 - 50 * LefteyeY);
442             Canvas.SetLeft(eyeLxy, 180 + 50 * LefteyeX);
443
444             //mapXY.Children.Remove(eyeR);
445             TranslateTransform trEyeRxy = new TranslateTransform(50 *
                  RighteyeX, -50 * RighteyeY);
446             TranslateTransform trEyeLxy = new TranslateTransform(50 *
                  LefteyeX, -50 * LefteyeY);
447             eyeRxy.RenderTransform = trEyeRxy;
448             eyeLxy.RenderTransform = trEyeLxy;
449             mapXY.Children.Add(eyeRxy);
450             mapXY.Children.Add(eyeLxy);
```

Figura 2-Y: adattamento dei punti oculari per essere mostrati sulla proiezione XY.

I valori sono centrati rispetto al canvas facendo riferimento alle dimensioni di quest'ultimo (dallo XAML: "KinectMapXZ" Width="360" Height="290") e dimezzate. Le ellissi sono quindi traslate in tempo reale inserendo le loro coordinate in variabili di tipo "TranslateTransform", il cui valore viene sovrascritto per ogni nuovo frame letto. La trasformazione è quindi aggiunta al canvas con "mapXY.Children.Add(eyeRxy)". Un procedimento analogo è utilizzato per la proiezione sulla mappa XZ.

2.6. Design dell'interfaccia

Lo scopo del rilevamento delle coordinate di occhi e naso è l'invio dei loro valori al generatore di contenuti AR. Oltre a questo, è opportuno disporre di un sistema per monitorare la lettura di tali dati. Si è quindi sviluppata un'interfaccia che mostra i parametri individuati, sia in forma testuale che in forma grafica.

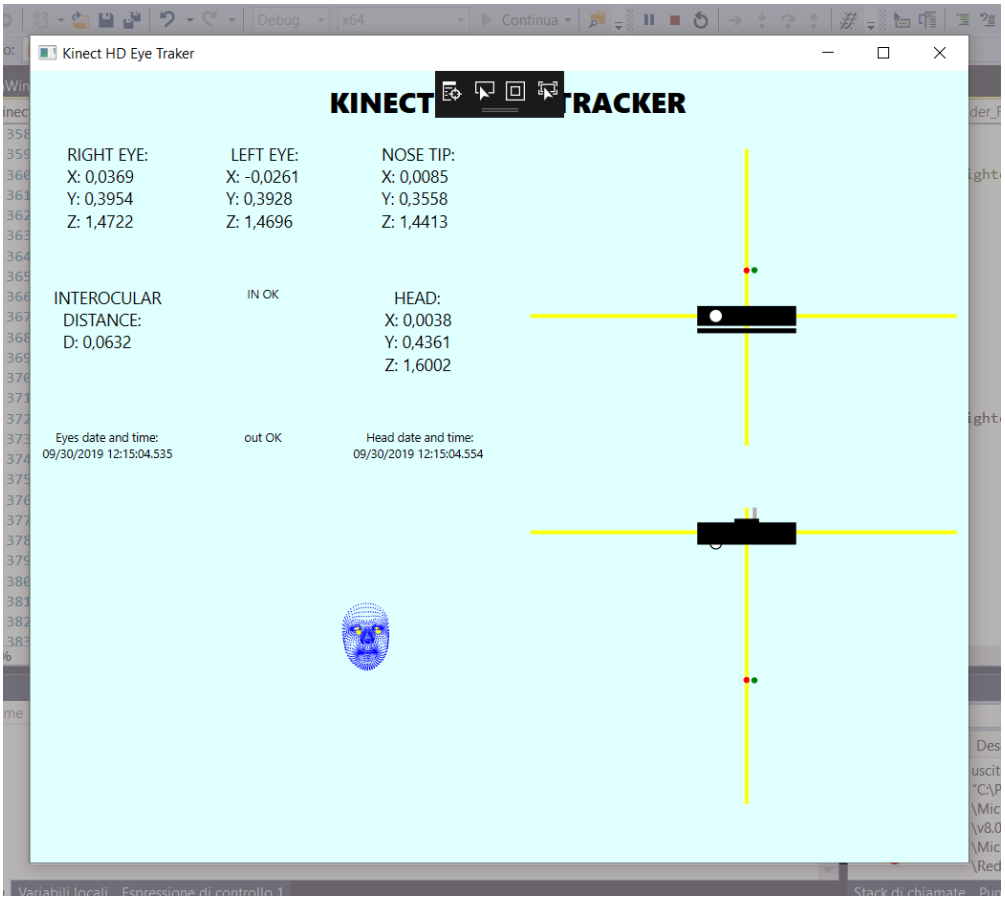


Figura 2-Z: interfaccia di KET - Kinect HD Eye Traker.

La finestra è divisa in 3 parti: il quadrante in alto a sinistra mostra i dati testuali in metri, quello in basso a sinistra presenta la ricostruzione 3D del viso dell'operatore formato dal migliaio di punti blu coincidenti su cui sono evidenziati in giallo i punti utilizzati per il calcolo della posizione degli occhi. Nella parte di destra è presente la rappresentazione ortogonale della posizione degli occhi rispetto a Kinect®, tramite una visuale frontale e una dall'altro. Le dimensioni del dispositivo non sono in scala rispetto alle coordinate.

La generazione della grafica è effettuata in markup XAML sempre tramite Visual Studio, all'interno della stessa soluzione che contiene il software. La parte di codice XAML è consultabile in Appendice 3.

È stata approntata anche una versione puramente testuale dell'interfaccia, denominata "LIGHT", a cui sono state tolte tutte le parti relative alla

visualizzazione dei punti, in modo da richiedere un minore sforzo computazionale.

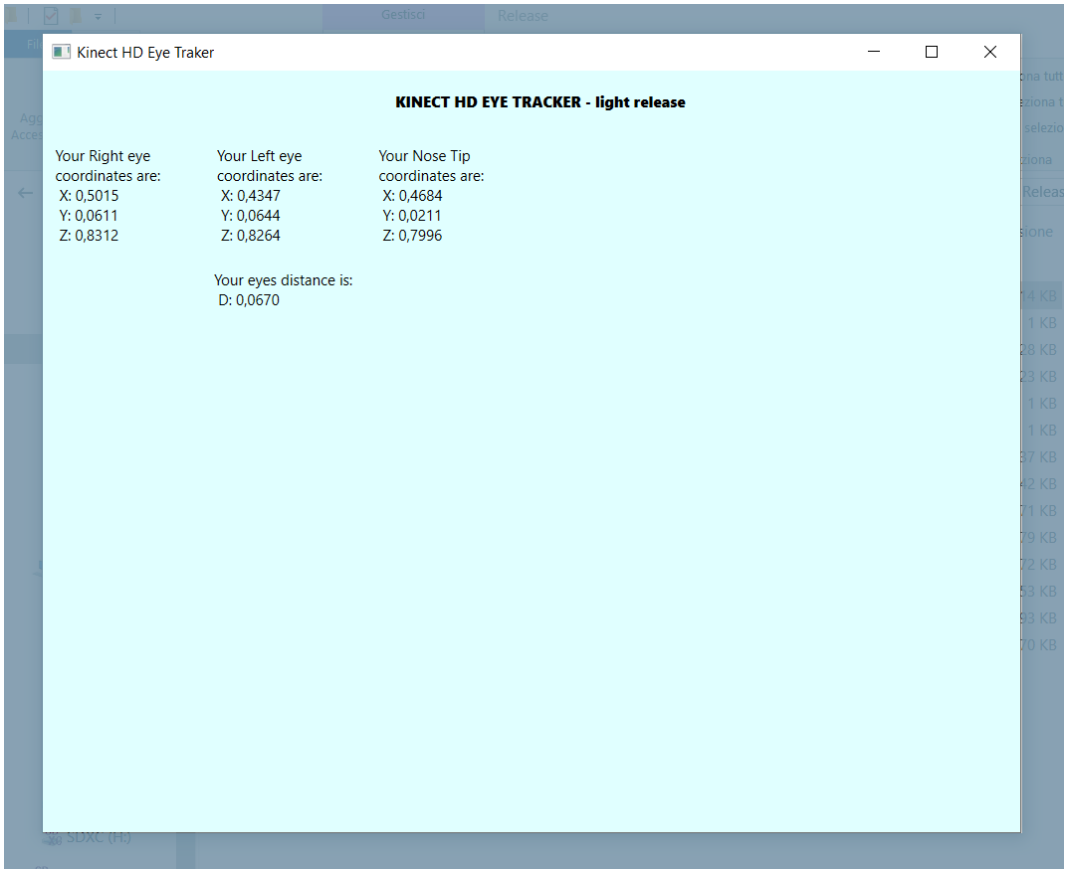


Figura 2-AA: interfaccia "LIGHT" di KET.

2.7. Gestione dei dati rilevati

I dati acquisiti sono opportunamente elaborati per essere inviati tramite socket. Di seguito sono commentati i blocchi di codice relativi all'intercomunicazione. Per una completa visione del listato, si rimanda ancora all'Appendice 1.

Si fa anzitutto riferimento agli opportuni namespace di .NET che contengono le funzioni di comunicazione:

```
using System.Net;
using System.IO;
using System.Net.Sockets;
```

Figura 2-BB: puntamento ai namespace.

È quindi configurato l'output con una variabile di tipo Socket, utilizzando il protocollo Udp. L'indirizzo IP della macchina di destinazione è impostato su "Loopback", in modo da poter essere inviato alla stessa macchina che sta utilizzando Kinect®, ma ad un qualsiasi altro software. È anche generato l'end-point "ep".

```
        //------------
        //output configuration

        //setting socket to send output stream. ProtocolType is set to Udp
        Socket socket = new Socket(AddressFamily.InterNetwork,
          SocketType.Dgram, ProtocolType.Udp);
        //setting IP to send stream to. as I have to send the stream to this
           very same machine, ip is in Loopback
        IPAddress IP = IPAddress.Loopback;
        //setting endpoint
        IPEndPoint ep;
        //------------
```

Figura 2-CC: setting del socket con protocollo e destinazione.

L'end-point è collegato alla porta 9999.

```
            //configuring end point
            ep = new IPEndPoint(IP, 9999);
```

Figura 2-DD: l'end-poi è collegato alla porta 9999

Dopo la parte di rilevamento e calcoli, i dati ottenuti possono essere inviati al socket in forma di stringa con spazio caratteri UTF8:

```
371             //send to socket
372             string message = Lefteyex + "\n" + Lefteyey + "\n" + Lefteyez +
                  "\n" + Righteyex + "\n" + Righteyey + "\n" + Righteyez + "\n"
                  + NoseTipx + "\n" + NoseTipy + "\n" + NoseTipz;
373
374             byte[] stream = Encoding.UTF8.GetBytes(message);
375             socket.SendTo(stream, ep);
```

Figura 2-EE: invio della stringa contenente i dati al socket.

È stata anche preparata una parte in grado di scrivere i dati in un file .txt, in modo da poterli agevolmente controllare anche dopo l'esecuzione del software. Poiché la frequenza di aggiornamento è molto alta, è chiaro come questo file, seppur contenga semplice testo, aumenti le sue dimensioni molto velocemente. Per associare i dati ad un certo istante è stata aggiunta l'informazione "NowString" con il metodo "DateTime", collegata in modo univoco ai valori letti in quel momento:

```
261             //introducing time association
262             DateTime Now = DateTime.Now;
263             consoleNow.Text = String.Format("    Eyes date and time: {0: MM/
                  dd/yyy HH:mm:ss.fff}", Now);
264             string NowString = Now.ToString("MM/dd/yyy HH:mm:ss.fff");
```

Figura 2-FF: generazione della stringa con i valori temporali.

Può quindi essere redatto il file di testo. Di seguito è riportato il suo blocco in forma di commento, poiché questa funzione è normalmente non utilizzata nella versione in uso dell'applicativo, mentre lasciarla attiva creerebbe grandi file .txt che saturerebbero la memoria del PC. Il file è nominato "EyePositionTime.txt" ed è localizzato sul desktop dell'utente corrente:

```
355             /*
356             //create a .txt file with captured data coupled with time
                  reference,
357             //due to be able to plot them on a chart
358
359             //creating a string with eyeposition data associated with time
360             string EyePositionTimeol = NowString + "\t\t" + "ED= " +
                  Eyedistancer + "\t\t" + "RX= " + Righteyex + "\t\t" + "RY= " +
                   Righteyey + "\t\t" + "RZ= " + Righteyez + "\t\t" + "LX= " +
                  Lefteyex + "\t\t" + "LY= " + Lefteyey + "\t\t" + "LZ= " +
                  Lefteyez + "\n";
361             //writing into .txt file
362             string TxtPath = "C:\\Users\\simon\\Desktop\\";
363             string TxtName = "EyePositionTime.txt";
364             using (StreamWriter outputFile = new StreamWriter(TxtPath + @"\"
                   + TxtName, true))
365             {
366                 outputFile.WriteLine(EyePositionTimeol);
367             }
368             */
```

Figura 2-GG: blocco del codice (disattivato) che crea il file .txt con i dati rilevati e accoppiati al dato temporale.

3. TEST DI FUNZIONAMENTO E RISULTATI

Il software, una volta compilato, funziona correttamente, restituendo valori delle posizioni di occhi e naso compatibili con quelli reali. Per verificare la bontà di tali valori, sono eseguite una serie di prove sperimentali, delle quali sono analizzati i risultati.

3.1. Confronto della misura della distanza interoculare stimata e reale: impostazione dell'esperimento

Poiché il centro del sistema cartesiano di misure di Kinect® si trova all'interno del dispositivo stesso, è complicato porsi con uno strumento di misura fisico nello stesso punto zero. Si ricorre quindi al confronto della differenza di due misure, rilevate con Kinect® grazie al software sviluppato e con strumenti fisici. L'idea è che la differenza delle misure dovrebbe essere indipendente dal centro del sistema di riferimento utilizzato e quindi identica, o quantomeno prossima, nonostante i due strumenti differenti utilizzati.

La scelta di quale differenza di distanze da calcolare è naturalmente ricaduta sulla distanza interoculare, dato che è già elaborata dal software e mostrata all'interno dell'interfaccia, e che è facile da misurare fisicamente tramite un semplice metro a nastro.

Come è stato evidenziato in precedenza, il centro dell'occhio ha una posizione diversa se calcolato come media delle posizioni degli angoli oculari piuttosto che dei bordi superiore e inferiore. Di conseguenza, anche le distanze interoculari saranno leggermente diverse. Per completezza, le distanze sono valutate con entrambi i metodi tramite KET, mentre la misura fisica è riferita al bordo superiore dell'occhio.

Si è quindi predisposta, all'interno del V-Lab, un'area di prova con un Kinect® e software KET, segnando sulla pedana sopraelevata del CAVE alcune basi (del tutto arbitrarie) in cui si sarebbero posizionati i volontari, di fronte al sensore Kinect®.

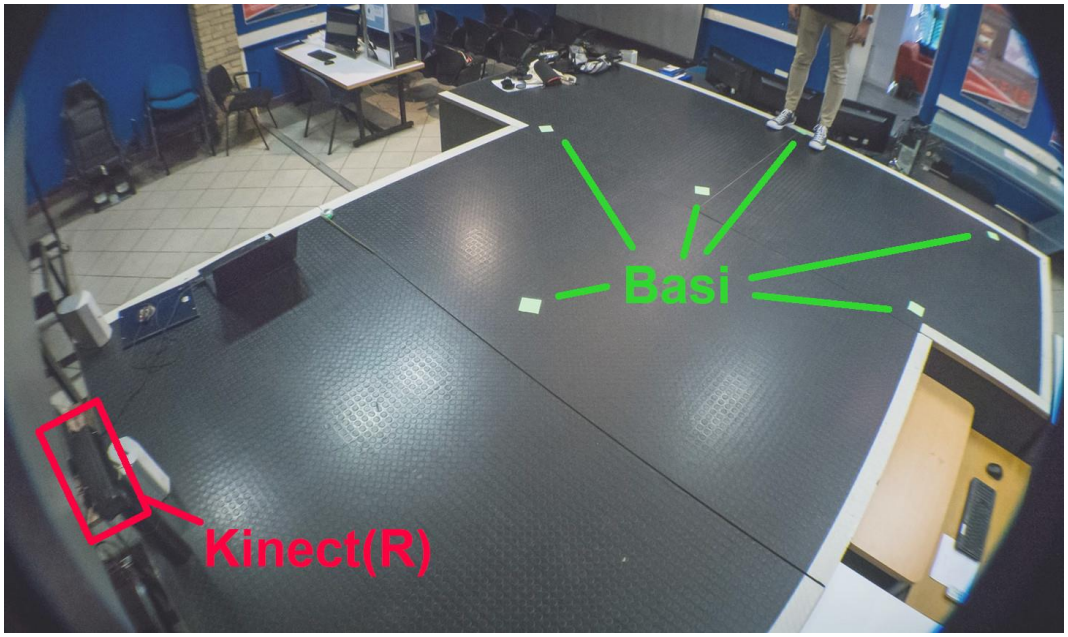


Figura 3-A: area di prova: pedana rialza con 6 basi segnate a terra difronte a Kinect(R).

Si sono effettuate le misurazioni della distanza interoculare con KET (per ogni volontario in ogni posizione, in due prove successive per i due diversi metodi di valutazione del centro oculare) e con il metro a nastro (una volta per ogni volontario).

La misura fisica avviene con il seguente procedimento: è apposto un metro a nastro a contatto aderente con la fronte del volontario, in modo che sia prossimo al bordo degli occhi senza oscurarli. È fatto sommariamente coincidere il 10 cm del metro con il bordo superiore più alto dell'occhio destro, in modo da avere un riferimento comodo per la misura. È quindi scattata una fotografia frontale al volontario. La fotografia è esaminata al pc tramite un software di fotoritocco, utile per tracciare riferimenti, in modo da fare una lettura quanto più precisa possibile della misura della distanza interoculare.

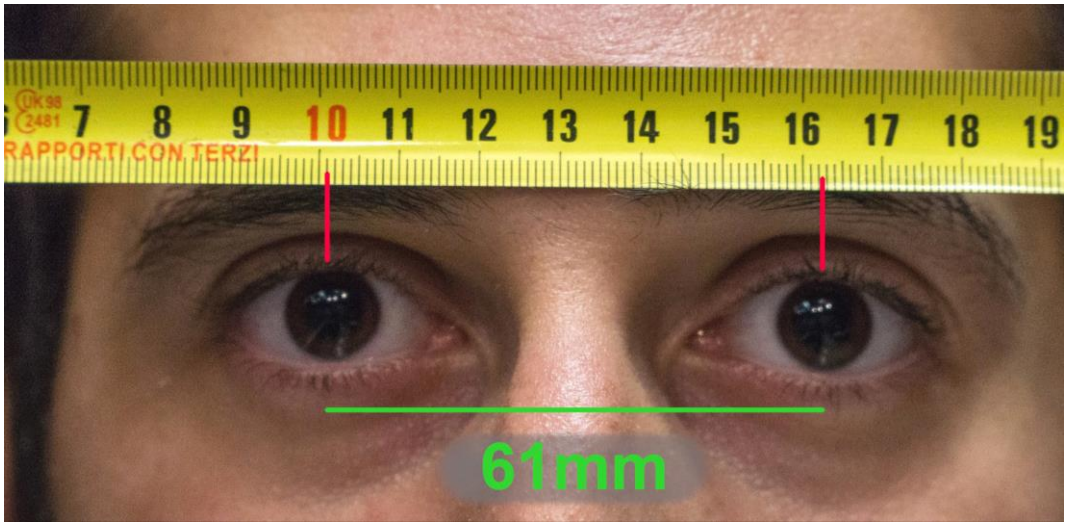


Figura 3-B: misura della distanza interoculare tramite metro a nastro, con il supporto di un software di grafica digitale. il centro è preso in coincidenza del bordo superiore dell'occhio.

La fotografia è scattata con lo zoom esteso (105mm su sensore APS-C, equivalente ad un 150mm circa) e diaframma chiuso. Questo contribuisce a "schiacciare" la tridimensionalità dell'immagine, migliorando la possibilità di lettura della stessa.

Il terzo volontario presenta cavità oculari particolarmente scavate, per cui è difficile individuare il punto più alto del bordo oculare. È quindi misurata la distanza interoculare con il metodo degli angoli oculari.

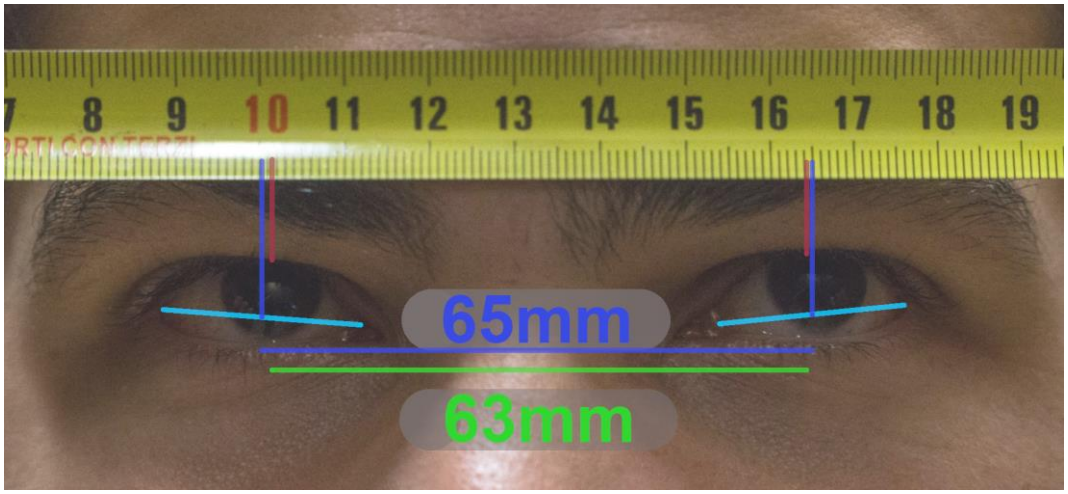


Figura 3-C: distanza interoculare calcolata con il metodo degli angoli oculari per il terzo volontario.

Per raffronto, in Appendice 4 sono presenti le fotografie con metro apposto dei 4 volontari.

3.2. Risultati

Sono stati valutati 2 gruppi di 6 misure ciascuno, per 4 diversi volontari, per un totale di 48 misure. In Appendice 5 è presentata la totalità dei dati grezzi, con anche una elaborazione preliminare. Di seguito sono presentati i risultati salienti.

Il primo confronto è quello tra la distanza interoculare misurata da KET e quella misurata fisicamente. Ovviamente, il dato fisico misurato con il metro si mantiene costante per ogni volontario, variando tra un volontario e l'altro. È considerata anche la norma del vettore delle coordinate della punta del naso (Nose Tip Norm, NTN) come riferimento per conoscere la distanza a cui si sta misurando con KET. Questo valore rientra come divisore nel calcolo della differenza relativa tra la misura di KET e quella fisica "(KET-ruller) /NTN %". L'idea è che tanto più lontano sto misurando, tanto più posso accettare errori nella misura. È considerata anche la differenza relativa con divisore la misura fisica "(KET-ruller) /ruller %".

Entrambe le misure relative "(KET-ruller) /NTN %" e "(KET-ruller) /ruller %" sono espresse in forma percentuale e mantengono il segno: positivo se la misura di KET è maggiore di quella effettuata con il metro, negativo viceversa.

VOLUNTEER	NOSE TIP COORDINATES	KET	RULLER	RELATIVE DIFFERENCE %	
Volontario	Nose Tip Norm NTN [m]	interocular Distance MID [m]	interocular Distance MID [m]	(KET-ruller) /ruller %	(KET-ruller) /NTN %
1	2,6979	0,0616	0,061	0,98361	0,02224
1	3,3103	0,0615	0,061	0,81967	0,01510
1	2,9754	0,0614	0,061	0,65574	0,01344
1	3,8761	0,0617	0,061	1,14754	0,01806
1	1,9070	0,0632	0,061	3,60656	0,11536
1	2,0638	0,0631	0,061	3,44262	0,10176

2	2,7821	0,0614	0,062	-0,96774	-0,02157
2	2,9535	0,0620	0,062	0,00000	0,00000
2	2,8879	0,0623	0,062	0,48387	0,01039
2	1,7203	0,0608	0,062	-1,93548	-0,06976
2	3,8643	0,0622	0,062	0,32258	0,00518
2	4,1522	0,0617	0,062	-0,48387	-0,00723
3	2,8048	0,0616	0,063	-2,22222	-0,04991
3	2,9617	0,0619	0,063	-1,74603	-0,03714
3	2,9335	0,0615	0,063	-2,38095	-0,05113
3	1,7260	0,0608	0,063	-3,49206	-0,12746
3	4,0265	0,0614	0,063	-2,53968	-0,03974
3	4,2203	0,0619	0,063	-1,74603	-0,02606
4	2,8445	0,0621	0,061	1,80328	0,03867
4	3,0328	0,0623	0,061	2,13115	0,04287
4	2,9604	0,0622	0,061	1,96721	0,04054
4	1,7665	0,0581	0,061	-4,75410	-0,16416
4	4,0002	0,0621	0,061	1,80328	0,02750
4	4,2242	0,0623	0,061	2,13115	0,03078

Tabella 1: sintesi dei dati rilevati con il metodo del bordo superiore degli occhi.

Per il volontario 3 è considerata anche la distanza interoculare calcolata a partire dalla media degli angoli interoculari.

Volontario	Nose Tip Norm NTN [m]	KET interocular Distance CORNER [m]	RULLER interocular Distance CORNER [m]	(KET-ruller) /ruller %	(KET-ruller) /NTN %
3	2,7545	0,0669	0,065	2,92308	0,06898
3	2,9650	0,0670	0,065	3,07692	0,06745
3	2,9494	0,0669	0,065	2,92308	0,06442
3	1,7148	0,0668	0,065	2,76923	0,10497
3	3,9725	0,0670	0,065	3,07692	0,05035
3	4,1757	0,0659	0,065	1,38462	0,02155

Tabella 2: sintesi dei dati rilevati con il metodo degli angoli oculari (solo volontario 3).

Si nota subito che la discrepanza tra la misura rilevata con Kinect® e software KET e la misura fisica effettuata con metro a nastro si mantiene sempre sotto il

5%. Se la differenza è riportata anche alla distanza a cui le misure KET sono effettuata, l'errore relativo è sempre inferiore allo 0,2%.

È possibile graficare i dati come serie di misure, indipendenti dal particolare volontario, che può comunque essere identificato dal tratto costante della spezzata dei dati relativi al metro fisico:

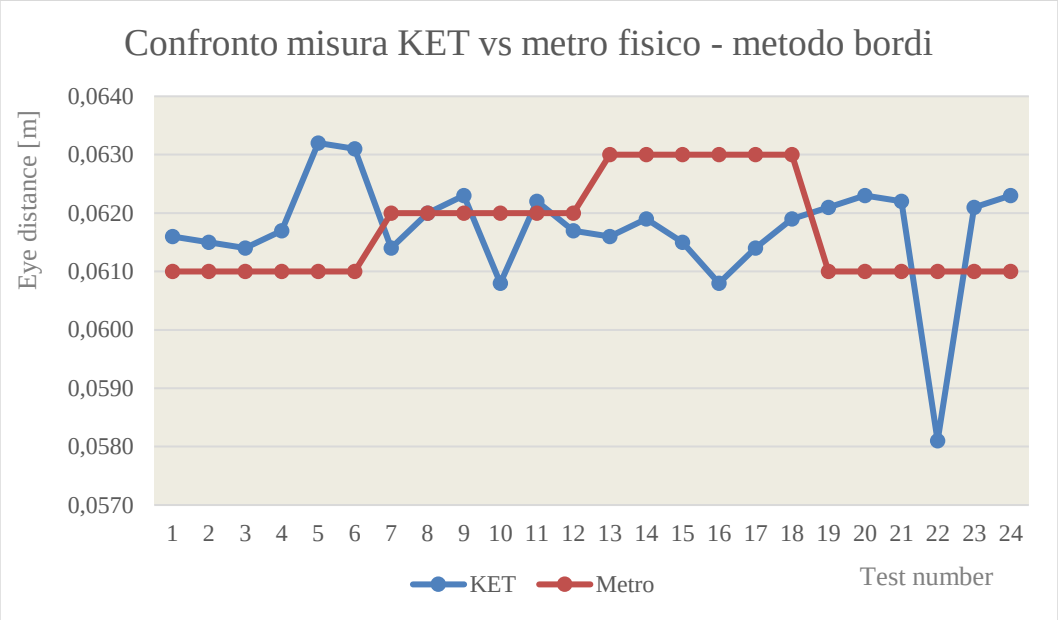


Grafico 3.1. Confronto misura KET vs metro fisico - metodo bordi.

Un grafico simile si può ottenere per il volontario 3 con la distanza interoculare calcolata con il metodo degli angoli oculari:

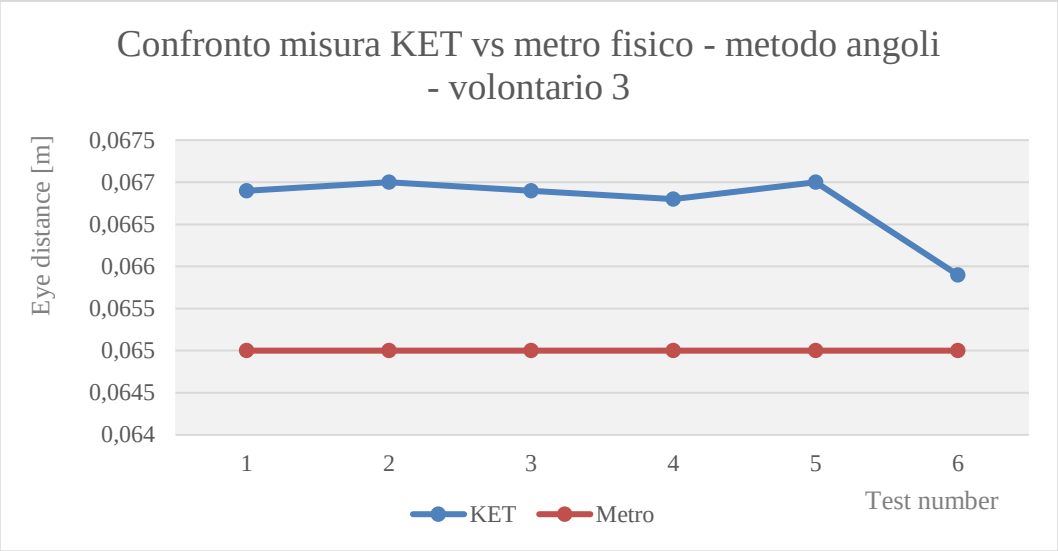


Grafico 3.2. Confronto misura KET vs metro fisico – metodo degli angoli oculari, per il solo volontario numero 3.

Una ulteriore valutazione può essere fatta mettendo in relazione la precisione della misura (differenza relativa "(KET-ruller) /ruller %") con la distanza a cui la misura KET è effettuata, ovvero la Nose Tip Norm, NTN:

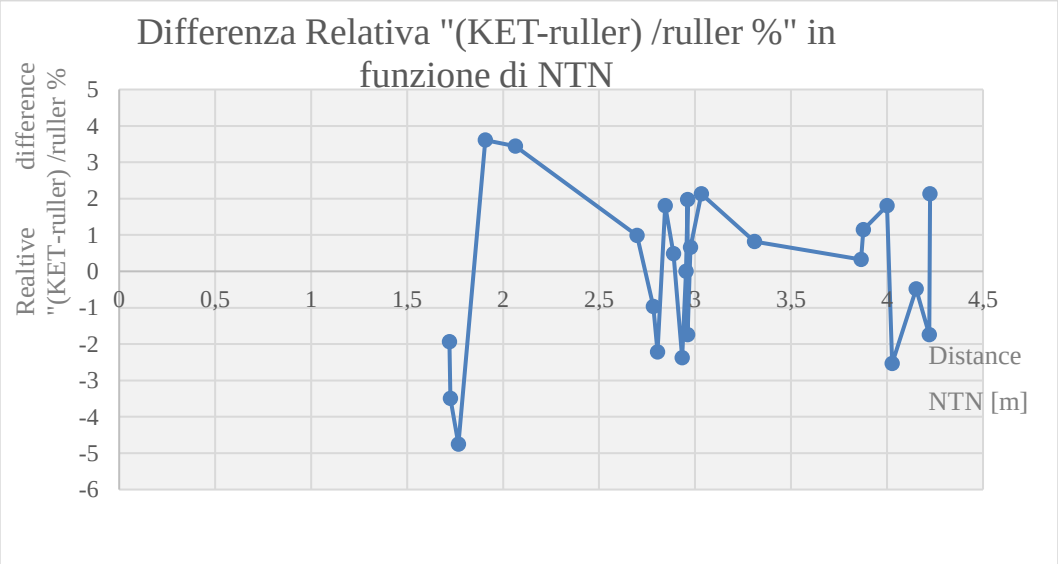


Grafico 3.3. Differenza Relativa "(KET-ruller) /ruller %" in funzione di NTN.

Da quest'ultimo grafico sembrerebbe, controintuitivamente, che la misura sia più precisa allontanandosi dal dispositivo Kinect® anziché avvicinandosi. Questo

fatto è spiegabile conoscendo il modo in cui funziona la TOF-Camera di Kinect®: alle distanze brevi, il fascio conico del proiettore IR non riesce ad aprirsi a sufficienza per distinguere al meglio i vari punti che proietta, mentre da una certa distanza in poi i punti sono sufficientemente separati per essere individuati senza sovrapposizioni.

4. CONCLUSIONI

In questo lavoro di ricerca è stata valutata la necessità di disporre di una visione binoculare per fruire di contenuti AR all'interno delle torri di controllo del traffico aereo. È emerso che l'indizio visivo dato dalla disparità binoculare sia importante anche a distanze superiori ai 30 metri, specialmente in situazioni in cui altre informazioni siano carenti.

Si è quindi sviluppato il software KET in linguaggio C# e markup XAML, in grado di interfacciarsi al dispositivo Kinect® per misurare in modo preciso e non invadente la posizione degli occhi e della punta del naso dell'utilizzatore, ricavando le loro coordinate rispetto al sensore e adattandole per essere inviate tramite socket al sistema di generazione dei contenuti AR,

Sono state svolte delle prove sperimentali per valutare la bontà delle misure rilevate, misurando le distanze interoculari di alcuni volontari, sia tramite KET che con un metro fisico. I valori ottenuti sono risultati altamente compatibili, con un errore relativo delle misure sempre inferiore al 5%, e addirittura inferiore allo 0,2% se si tiene conto anche della distanza di misurazione.

Dalle prove è emerso anche che le misure siano meno precise trovandosi in prossimità del dispositivo (distanza minore di 2,5 metri), mentre la precisione è stabile oltre i 2,5 metri. Si ipotizza che questo sia dovuto al modo in cui funziona la TOF-Camera di Kinect®, che si avvale di un fascio IR proiettato per generare dei riferimenti luminosi per la videocamera IR.

Per un futuro sviluppo del software, sarebbe opportuno renderlo più resistente alla presenza di più persone nel campo visivo di Kinect®, eventualmente rendendo KET in grado di misurare più utenti contemporaneamente.

RINGRAZIAMENTI

Con questa tesi concludo un dilungatissimo percorso di studi. Ringrazio le persone che mi hanno dato fiducia ancora e ancora, o semplicemente hanno portato pazienza, nella prospettiva che prima o poi avessi raggiunto questo traguardo, aiutandomi anche perché continuassi, tra passetti avanti e indietro e di fianco, a puntare alla conclusione degli studi. In particolare ringrazio la mia famiglia, Sara e Nicola per avermi affidato questa ricerca, la mia ragazza Alice, che mi ha spronato nell'indolenza e sostenuto nelle fatiche, Dario che si è ridotto ad uscire in barca da solo per lasciarmi studiare, Costanza, che ogni Luglio mi ha ricordato che può sempre succedere qualcosa di inaspettato.

Nelle storie lunghe, per quanto belle, ad un certo punto fa piacere leggere la parola FINE.

APPENDICE 1:

CODICE SORGENTE SVILUPPATO “KINECT HD EYE-TRACKING”

...tHDFEyeTracking\KinectHDFEyeTracking\MainWindow.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Drawing;
5  using System.Globalization;
6  using System.Linq;
7  using System.Text;
8  using System.Net;
9  using System.IO;
10 using System.Net.Sockets;
11 using System.Threading.Tasks;
12 using System.Timers;
13 using System.Windows;
14 using System.Windows.Controls;
15 using System.Windows.Data;
16 using System.Windows.Documents;
17 using System.Windows.Input;
18 using System.Windows.Media;
19 using System.Windows.Media.Media3D;
20 using System.Windows.Media.Imaging;
21 using System.Windows.Navigation;
22 using System.Windows.Shapes;
23 using Microsoft.Kinect;
24 using Microsoft.Kinect.Face;
25
26 namespace KinectHDFEyeTracking
27 {
28     /// <summary>
29     /// Logica di interazione per MainWindow.xaml
30     /// </summary>
31     public partial class MainWindow : Window
32     {
33
34         // Provides a Kinect sensor reference.
35         private KinectSensor _sensor = null;
36
37         // Acquires body frame data.
38         private BodyFrameSource _bodySource = null;
39
40         // Reads body frame data.
41         private BodyFrameReader _bodyReader = null;
42 /**/
43         Body[] bodies = new Body[6];
44 /**/
45
46         // Acquires HD face data.
47         private HighDefinitionFaceFrameSource _faceSource = null;
48
49         // Reads HD face data.
50         private HighDefinitionFaceFrameReader _faceReader = null;
51 /**/
52         // Reads multisurce frame data.
53         private MultiSourceFrameReader _multiReader = null;
```

```
54  /**/
55          // Required to access the face vertices.
56          private FaceAlignment _faceAlignment = null;
57
58          // Required to access the face model points.
59          private FaceModel _faceModel = null;
60
61          // Used to display 1,000 points on screen.
62          private List<Ellipse> _points = new List<Ellipse>();
63
64
65
66
67
68          //------------
69          //output configuration
70
71          //setting socket to send output stream. ProtocolType is set to Udp
72          Socket socket = new Socket(AddressFamily.InterNetwork,
              SocketType.Dgram, ProtocolType.Udp);
73          //setting IP to send stream to. as I have to send the stream to this
               very same machine, ip is in Loopback
74          IPAddress IP = IPAddress.Loopback;
75          //setting endpoint
76          IPEndPoint ep;
77          //------------
78
79
80          public MainWindow()
81          {
82              InitializeComponent();
83
84
85              //configuring end point
86              ep = new IPEndPoint(IP, 9999);
87
88              //initializing KinectSensor
89              _sensor = KinectSensor.GetDefault();
90
91              if (_sensor != null)
92              {
93                  // Listen for body data.
94                  _bodySource = _sensor.BodyFrameSource;
95                  _bodyReader = _bodySource.OpenReader();
96                  _bodyReader.FrameArrived += BodyReader_FrameArrived;
97
98  /**/
99                  // Listen for multisurce data.
100                 _multiReader = _sensor.OpenMultiSourceFrameReader
                      (FrameSourceTypes.Color);
101                 _multiReader.MultiSourceFrameArrived +=
                      MultiReader_MultiSourceFrameArrived;
102 /**/
```

```
103                 // Listen for HD face data.
104                 _faceSource = new HighDefinitionFaceFrameSource(_sensor);
105                 _faceReader = _faceSource.OpenReader();
106                 _faceReader.FrameArrived += FaceReader_FrameArrived;
107
108                 _faceModel = new FaceModel();
109                 _faceAlignment = new FaceAlignment();
110
111                 // Start tracking!
112                 _sensor.Open();
113             }
114         }
115
116         private void BodyReader_FrameArrived(object sender,
              BodyFrameArrivedEventArgs e)
117         {
118
119             //debugin.Text = String.Format("IN OK");
120             /**/
121             using (BodyFrame bf = e.FrameReference.AcquireFrame())
122             {
123                 if (bf != null)
124                 {
125                     bf.GetAndRefreshBodyData(bodies);
126                     foreach (Body body in bodies)
127                     {
128                         if (body.IsTracked)
129                         {
130                             Joint headJoint = body.Joints[JointType.Head];
131                             if (headJoint.TrackingState ==
                        TrackingState.Tracked)
132                             {
133                                 Single X = headJoint.Position.X;
134                                 Single Y = headJoint.Position.Y;
135                                 Single Z = headJoint.Position.Z;
136
137                                 string x = X.ToString("0.0000");
138                                 string y = Y.ToString("0.0000");
139                                 string z = Z.ToString("0.0000");
140
141                                 consoleHeadJoint.Text = String.Format("
                        HEAD: \n X: {0} \n Y: {1} \n Z: {2} \n", x, y, z);
142
143                                 //introducing time association
144                                 DateTime NowH = DateTime.Now;
145                                 consoleNowH.Text = String.Format("    Head
                        date and time: {0: MM/dd/yyy HH:mm:ss.fff}", NowH);
146                                 string NowHString = NowH.ToString("MM/dd/yyy
                         HH:mm:ss.fff");
147                                 /*
148                                 //creating a string with head data
                        associated with time
149                                 string HeadPositionTimeol = NowHString + "\t
```

```
                    \t" + "HX= " + x + "\t\t" + "HY= " + y + "\t\t" + "HZ= "
                    + z + "\n";
                                //writing into .txt file
                                string TxtPath = "C:\\Users\\simon\\Desktop\
                    \";
                                string TxtName = "HeadPositionTime.txt";
                                using (StreamWriter outputFile = new
                    StreamWriter(TxtPath + @"\" + TxtName, true))
                                {
                                    outputFile.WriteLine
                    (HeadPositionTimeol);
                                }
                                */
                                //debugout.Text = String.Format("out OK");
                            }
                        }
                    }
                }
            }
            /**/


            using (var frame = e.FrameReference.AcquireFrame())
            {
                debugin.Text = String.Format("IN OK");
                if (frame != null)
                {
                    Body[] bodies = new Body[frame.BodyCount];
                    frame.GetAndRefreshBodyData(bodies);

                    Body body = bodies.Where(b =>
                    b.IsTracked).FirstOrDefault();

                    if (!_faceSource.IsTrackingIdValid)
                    {
                        if (body != null)
                        {
                            _faceSource.TrackingId = body.TrackingId;
                        }
                    }
                }
                debugout.Text = String.Format("out OK");
            }
        }




        /**/

        void MultiReader_MultiSourceFrameArrived(object sender,
```

```
            MultiSourceFrameArrivedEventArgs ecolor)
197         {
198             // Get a reference to the multi-frame
199             var reference = ecolor.FrameReference.AcquireFrame();
200 
201             // Open color frame
202             using (var frame = reference.ColorFrameReference.AcquireFrame())
203             {
204                 if (frame != null)
205                 {
206                     camera.Source = ToBitmap(frame);
207                 }
208             }
209         }
210 /**/
211 
212 
213         private void FaceReader_FrameArrived(object sender,
              HighDefinitionFaceFrameArrivedEventArgs e)
214         {
215             using (var frame = e.FrameReference.AcquireFrame())
216             {
217                 if (frame != null && frame.IsFaceTracked)
218                 {
219                     frame.GetAndRefreshFaceAlignmentResult(_faceAlignment);
220                     UpdateFacePoints();
221                 }
222             }
223         }
224 
225         /**/
226         private ImageSource ToBitmap(ColorFrame frame)
227         {
228             int width = frame.FrameDescription.Width;
229             int height = frame.FrameDescription.Height;
230             PixelFormat format = PixelFormats.Bgr32;
231 
232             byte[] pixels = new byte[width * height *
                  ((PixelFormats.Bgr32.BitsPerPixel + 7) / 8)];
233 
234             if (frame.RawColorImageFormat == ColorImageFormat.Bgra)
235             {
236                 frame.CopyRawFrameDataToArray(pixels);
237             }
238             else
239             {
240                 frame.CopyConvertedFrameDataToArray(pixels,
                      ColorImageFormat.Bgra);
241             }
242 
243             int stride = width * format.BitsPerPixel / 8;
244 
245             return BitmapSource.Create(width, height, 96, 96, format, null,
```

```
                pixels, stride);
246
247         }
248         /**/
249
250
251
252
253         private void UpdateFacePoints()
254         {
255
256
257             if (_faceModel == null) return;
258
259             var vertices = _faceModel.CalculateVerticesForAlignment
                  (_faceAlignment);
260
261             //introducing time association
262             DateTime Now = DateTime.Now;
263             consoleNow.Text = String.Format("    Eyes date and time: {0: MM/
                  dd/yyy HH:mm:ss.fff}", Now);
264             string NowString = Now.ToString("MM/dd/yyy HH:mm:ss.fff");
265
266
267             //NoseTip position
268             var NoseTip = vertices[(int)HighDetailFacePoints.NoseTip];
269             //convert to single
270             Single NoseTipX = NoseTip.X;
271             Single NoseTipY = NoseTip.Y;
272             Single NoseTipZ = NoseTip.Z;
273
274             string NoseTipx = NoseTipX.ToString("0.0000");
275             string NoseTipy = NoseTipY.ToString("0.0000");
276             string NoseTipz = NoseTipZ.ToString("0.0000");
277
278             consoleNoseTip.Text = String.Format(" NOSE TIP: \n X: {0} \n Y:
                  {1} \n Z: {2} \n", NoseTipx, NoseTipy, NoseTipz);
279
280
281             //EyeLeft position
282
283             //Catch mid eye top position
284             var LefteyeMidtop = vertices[(int)
                  HighDetailFacePoints.LefteyeMidtop];
285             //convert to single
286             Single LefteyeMidtopX = LefteyeMidtop.X;
287             Single LefteyeMidtopY = LefteyeMidtop.Y;
288             Single LefteyeMidtopZ = LefteyeMidtop.Z;
289
290             //Catch mid eye bottom position
291             var LefteyeMidbottom = vertices[(int)
                  HighDetailFacePoints.LefteyeMidbottom];
292             //convert to single
```

```
293             Single LefteyeMidbottomX = LefteyeMidbottom.X;
294             Single LefteyeMidbottomY = LefteyeMidbottom.Y;
295             Single LefteyeMidbottomZ = LefteyeMidbottom.Z;
296
297             //Calculate eye position as a mean between top and bottom eye
                  mid
298             Single LefteyeX = (LefteyeMidbottomX + LefteyeMidtopX) / 2;
299             Single LefteyeY = (LefteyeMidbottomY + LefteyeMidtopY) / 2;
300             Single LefteyeZ = (LefteyeMidbottomZ + LefteyeMidtopZ) / 2;
301
302             string Lefteyex = LefteyeX.ToString("0.0000");
303             string Lefteyey = LefteyeY.ToString("0.0000");
304             string Lefteyez = LefteyeZ.ToString("0.0000");
305
306             consoleEyeLeft.Text = String.Format("  LEFT EYE: \n X: {0} \n Y:
                   {1} \n Z: {2} \n", Lefteyex, Lefteyey, Lefteyez);
307
308
309
310             //EyeRight position
311
312             //Catch outer eye mid top position
313             var RighteyeMidtop = vertices[(int)
                  HighDetailFacePoints.RighteyeMidtop];
314             //convert to single
315             Single RighteyeMidtopX = RighteyeMidtop.X;
316             Single RighteyeMidtopY = RighteyeMidtop.Y;
317             Single RighteyeMidtopZ = RighteyeMidtop.Z;
318
319             //Catch inner eye mid bottom position
320             var RighteyeMidbottom = vertices[(int)
                  HighDetailFacePoints.RighteyeMidbottom];
321             //convert to single
322             Single RighteyeMidbottomX = RighteyeMidbottom.X;
323             Single RighteyeMidbottomY = RighteyeMidbottom.Y;
324             Single RighteyeMidbottomZ = RighteyeMidbottom.Z;
325
326             //Calculate eye position as a mean between top and bottom eye
                  mid
327             Single RighteyeX = (RighteyeMidbottomX + RighteyeMidtopX) / 2;
328             Single RighteyeY = (RighteyeMidbottomY + RighteyeMidtopY) / 2;
329             Single RighteyeZ = (RighteyeMidbottomZ + RighteyeMidtopZ) / 2;
330
331             string Righteyex = RighteyeX.ToString("0.0000");
332             string Righteyey = RighteyeY.ToString("0.0000");
333             string Righteyez = RighteyeZ.ToString("0.0000");
334
335             consoleEyeRight.Text = String.Format(" RIGHT EYE: \n X: {0} \n
                  Y: {1} \n Z: {2} \n", Righteyex, Righteyey, Righteyez);
336
337
338             //interocular distance
339
```

```
340             //Calculate interocular distance
341             Single EyedistanceX = RighteyeX - LefteyeX;
342             Single EyedistanceY = RighteyeY - LefteyeY;
343             Single EyedistanceZ = RighteyeZ - LefteyeZ;
344             //"S" denotes "square": for computational cost it's better to
                  use the square distance
345             //and calculate the squareroot only if need.
346             //for the same reason, it's better to calculate a square as a
                  product instead of as a power 2.
347             Single EyedistanceS = EyedistanceX * EyedistanceX + EyedistanceY
                   * EyedistanceY + EyedistanceZ * EyedistanceZ;
348             //extract squareroot
349             double EyedistanceR = Math.Sqrt((double)EyedistanceS);
350             //convert to string
351             string Eyedistancer = EyedistanceR.ToString("0.0000");
352
353             consoleEyedistancer.Text = String.Format(" INTEROCULAR \n
                  DISTANCE: \n   D: {0}", Eyedistancer);
354
355             /*
356             //create a .txt file with captured data coupled with time
                  reference,
357             //due to be able to plot them on a chart
358
359             //creating a string with eyeposition data associated with time
360             string EyePositionTimeol = NowString + "\t\t" + "ED= " +
                  Eyedistancer + "\t\t" + "RX= " + Righteyex + "\t\t" + "RY= " +
                   Righteyey + "\t\t" + "RZ= " + Righteyez + "\t\t" + "LX= " +
                  Lefteyex + "\t\t" + "LY= " + Lefteyey + "\t\t" + "LZ= " +
                  Lefteyez + "\n";
361             //writing into .txt file
362             string TxtPath = "C:\\Users\\simon\\Desktop\\";
363             string TxtName = "EyePositionTime.txt";
364             using (StreamWriter outputFile = new StreamWriter(TxtPath + @"\"
                   + TxtName, true))
365             {
366                 outputFile.WriteLine(EyePositionTimeol);
367             }
368             */
369
370
371             //send to socket
372             string message = Lefteyex + "\n" + Lefteyey + "\n" + Lefteyez +
                  "\n" + Righteyex + "\n" + Righteyey + "\n" + Righteyez + "\n"
                  + NoseTipx + "\n" + NoseTipy + "\n" + NoseTipz;
373
374             byte[] stream = Encoding.UTF8.GetBytes(message);
375             socket.SendTo(stream, ep);
376
377
378
379             //drawing vertices to points
380
```

```
381
382             if (vertices.Count > 0)
383             {
384                 if (_points.Count == 0)
385                 {
386                     for (int index = 0; index < vertices.Count; index++)
387                     {
388                         if (index == 1090 || index == 241 || index == 1104
                        || index == 731)
389                         {
390                             Ellipse ellipse = new Ellipse
391                             {
392                                 Width = 4.0,
393                                 Height = 4.0,
394                                 Fill = new SolidColorBrush(Colors.Yellow)
395                             };
396                             _points.Add(ellipse);
397                         }
398
399                         else
400                         {
401                           Ellipse ellipse = new Ellipse
402                           {
403                                Width = 1.0,
404                              Height = 1.0,
405                              Fill = new SolidColorBrush(Colors.Blue)
406                            };
407                             _points.Add(ellipse);
408                         }
409
410                     }
411
412                     foreach (Ellipse ellipse in _points)
413                     {
414                         canvas.Children.Add(ellipse);
415                     }
416                 }
417
418                 for (int index = 0; index < vertices.Count; index++)
419                 {
420
421                         CameraSpacePoint vertice = vertices[index];
422                         DepthSpacePoint point =
                        _sensor.CoordinateMapper.MapCameraPointToDepthSpace
                        (vertice);
423
424                         if (float.IsInfinity(point.X) || float.IsInfinity
                        (point.Y)) return;
425
426                         Ellipse ellipse = _points[index];
427
428                         Canvas.SetLeft(ellipse, point.X);
429                         Canvas.SetTop(ellipse, point.Y);
```

```
430
431                 }
432             }
433
434             //draw eyes on mapXY
435             mapXY.Children.Clear();
436             Ellipse eyeRxy = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Green };
437             Ellipse eyeLxy = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Red };
438
439             Canvas.SetTop(eyeRxy, 145 - 50 * RighteyeY);
440             Canvas.SetLeft(eyeRxy, 180 + 50 * RighteyeX);
441             Canvas.SetTop(eyeLxy, 145 - 50 * LefteyeY);
442             Canvas.SetLeft(eyeLxy, 180 + 50 * LefteyeX);
443
444             //mapXY.Children.Remove(eyeR);
445             TranslateTransform trEyeRxy = new TranslateTransform(50 *
                  RighteyeX, -50 * RighteyeY);
446             TranslateTransform trEyeLxy = new TranslateTransform(50 *
                  LefteyeX, -50 * LefteyeY);
447             eyeRxy.RenderTransform = trEyeRxy;
448             eyeLxy.RenderTransform = trEyeLxy;
449             mapXY.Children.Add(eyeRxy);
450             mapXY.Children.Add(eyeLxy);
451
452
453
454             //draw eyes on mapXZ
455             mapXZ.Children.Clear();
456             Ellipse eyeR = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Green };
457             Ellipse eyeL = new Ellipse() { Width = 5, Height = 5, Fill =
                  Brushes.Red };
458
459             Canvas.SetTop(eyeR, 50 * RighteyeZ);
460             Canvas.SetLeft(eyeR, 180 + 50 * RighteyeX);
461             Canvas.SetTop(eyeL, 50 * LefteyeZ);
462             Canvas.SetLeft(eyeL, 180 + 50 * LefteyeX);
463
464             //mapXZ.Children.Remove(eyeR);
465             TranslateTransform trEyeR = new TranslateTransform(50 *
                  RighteyeX, 50 * RighteyeZ);
466             TranslateTransform trEyeL = new TranslateTransform(50 *
                  LefteyeX, 50 * LefteyeZ);
467             eyeR.RenderTransform = trEyeR;
468             eyeL.RenderTransform = trEyeL;
469             mapXZ.Children.Add(eyeR);
470             mapXZ.Children.Add(eyeL);
471
472
473         }
474     }
```



```
475 }
476
```

APPENDICE 2:

ENUMERATORE “HighDetailFacePoints”

E INDICI ASSOCIATI

Key	Index
DetailFacePoints_LefteyeInnercorner	210
DetailFacePoints_LefteyeOutercorner	469
DetailFacePoints_LefteyeMidtop	241
DetailFacePoints_LefteyeMidbottom	1104
DetailFacePoints_RighteyeInnercorner	843
DetailFacePoints_RighteyeOutercorner	1117
DetailFacePoints_RighteyeMidtop	731
DetailFacePoints_RighteyeMidbottom	1090
DetailFacePoints_LefteyebrowInner	346
DetailFacePoints_LefteyebrowOuter	140
DetailFacePoints_LefteyebrowCenter	222
DetailFacePoints_RighteyebrowInner	803
DetailFacePoints_RighteyebrowOuter	758
DetailFacePoints_RighteyebrowCenter	849
DetailFacePoints_MouthLeftcorner	91
DetailFacePoints_MouthRightcorner	687
DetailFacePoints_MouthUpperlipMidtop	19
DetailFacePoints_MouthUpperlipMidbottom	1072
DetailFacePoints_MouthLowerlipMidtop	10
DetailFacePoints_MouthLowerlipMidbottom	8
DetailFacePoints_NoseTip	18
DetailFacePoints_NoseBottom	14
DetailFacePoints_NoseBottomleft	156
DetailFacePoints_NoseBottomright	783
DetailFacePoints_NoseTop	24
DetailFacePoints_NoseTopleft	151
DetailFacePoints_NoseTopright	772
DetailFacePoints_ForeheadCenter	28
DetailFacePoints_LeftcheekCenter	412
DetailFacePoints_RightcheekCenter	933
DetailFacePoints_Leftcheekbone	458
DetailFacePoints_Rightcheekbone	674

DetailFacePoints_ChinCenter	4
DetailFacePoints_LowerjawLeftend	1307
DetailFacePoints_LowerjawRightend	1327

Tabella 3: Enumeratore HighDetailFacePoints con indici e nomi dei riferimenti facciali principali. pterneas.com.

APPENDICE 3:
CODICE XAML PER LA CREAZIONE DELL'INTERFACCIA GRAFICA

...yeTracking - Copia\KinectHDFEyeTracking\MainWindow.xaml 1

```
<Window x:Class="KinectHDFEyeTracking.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:KinectHDFEyeTracking"
        mc:Ignorable="d"
        Title="Kinect HD Eye Traker" Height="700" Width="800">
    <Grid x:Name="mainGrid" Background="LightCyan" Width="800" Height="950">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="400" />
            <ColumnDefinition Width="400" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="50" />
            <RowDefinition Height="300" />
            <RowDefinition Height="300" />
        </Grid.RowDefinitions>

        <Label Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"
          Content="KINECT HD EYE TRACKER" FontSize="24"
          HorizontalAlignment="Center" VerticalAlignment="Center"
          Width="Auto" FontWeight="ExtraBlack"
          RenderTransformOrigin="0.5,3.401"/>

        <Viewbox Grid.Row="2" Grid.Column="0" Width="400" Height="300"
          Margin="5">
            <!--
            <Image Name="camera" />
            -->
        </Viewbox>

        <Viewbox Grid.Row="2" Grid.Column="0" Width="400" Height="300"
          Margin="5">
            <Canvas Name="canvas" Width="360" Height="290" >
                <!-- here will be drawn the face point-->
            </Canvas>
        </Viewbox>

        <Grid Grid.Row="1" Grid.Column="0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="130" />
                <ColumnDefinition Width="130" />
                <ColumnDefinition Width="130" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="120" />
                <RowDefinition Height="120" />
                <RowDefinition Height="60" />
            </Grid.RowDefinitions>
            <TextBlock x:Name="consoleEyeRight" Grid.Row="0" Grid.Column="0"
              HorizontalAlignment="Center" FontSize="14" TextWrapping="Wrap"
```

```
                Text="EyeRight" VerticalAlignment="Top" Margin="10"/>
46              <TextBlock x:Name="consoleEyeLeft"  Grid.Row="0" Grid.Column="1"
                  HorizontalAlignment="Center" FontSize="14" TextWrapping="Wrap"
                  Text="EyeLeft" VerticalAlignment="Top" Margin="10"/>
47              <TextBlock x:Name="consoleNoseTip" Grid.Row="0" Grid.Column="2"
                  HorizontalAlignment="Center" FontSize="14" TextWrapping="Wrap"
                  Text="NoseTip" VerticalAlignment="Top" Margin="10"/>
48              <TextBlock x:Name="consoleEyedistancer" Grid.Row="1"
                  Grid.Column="0"  HorizontalAlignment="Center" FontSize="14"
                  TextWrapping="Wrap" Text="EyeDistance" VerticalAlignment="Top"
                  Margin="10"/>
49              <TextBlock x:Name="consoleHeadJoint" Grid.Row="1" Grid.Column="2"
                     HorizontalAlignment="Center" FontSize="14" Text="HeadJoint"
                  VerticalAlignment="Top" Margin="10"/>
50              <TextBlock x:Name="consoleNow" Grid.Row="2" Grid.Column="0"
                  HorizontalAlignment="Center" FontSize="10" TextWrapping="Wrap"
                  Text="NowDateTime" VerticalAlignment="Top" Margin="10"/>
51              <TextBlock x:Name="consoleNowH" Grid.Row="2" Grid.Column="2"
                  HorizontalAlignment="Center" FontSize="10" TextWrapping="Wrap"
                  Text="NowDateTimeHead" VerticalAlignment="Top" Margin="10"/>
52
53              <TextBlock x:Name="debugin" Grid.Row="1" Grid.Column="1"
                  HorizontalAlignment="Center" FontSize="10" TextWrapping="Wrap"
                  Text="debugin" VerticalAlignment="Top" Margin="10"/>
54              <TextBlock x:Name="debugout" Grid.Row="2" Grid.Column="1"
                  HorizontalAlignment="Center" FontSize="10" TextWrapping="Wrap"
                  Text="debugout" VerticalAlignment="Top" Margin="10"/>
55
56          </Grid>
57
58
59          <Viewbox Grid.Row="1" Grid.Column="2" Width="400" Height="300"
              Margin="5">
60              <Canvas Name="KinectMapXY" Width="360" Height="290">
61                  <!-- draw axis -->
62                  <Line X1="5" Y1="145" X2="350" Y2="145" Stroke="Yellow"
                      StrokeThickness="3"/>
63                  <Line X1="180" Y1="10" X2="180" Y2="250" Stroke="Yellow"
                      StrokeThickness="3"/>
64                  <!-- draw Kinect -->
65                  <Rectangle Canvas.Top="137" Canvas.Left="140" Width="80"
                      Height="16" Fill="Black" />
66                  <Rectangle Canvas.Top="155" Canvas.Left="140" Width="80"
                      Height="4" Fill="Black" />
67                  <Ellipse Canvas.Top="140" Canvas.Left="150" Fill="White"
                      Height="10" Width="10"/>
68              </Canvas>
69          </Viewbox>
70          <Viewbox Grid.Row="1" Grid.Column="2" Width="400" Height="300"
              Margin="5">
71              <Canvas Name="mapXY" Width="360" Height="290">
72                  <!-- here will be drawn the eyes point-->
73              </Canvas>
```

```
74          </Viewbox>
75
76          <Viewbox Grid.Row="2" Grid.Column="2" Width="400" Height="300"
              Margin="5">
77              <Canvas Name="KinectMapXZ" Width="360" Height="290">
78                  <!-- draw axis -->
79                  <Line X1="5" Y1="30" X2="350" Y2="30" Stroke="Yellow"
                      StrokeThickness="3"/>
80                  <Line X1="180" Y1="10" X2="180" Y2="250" Stroke="Yellow"
                      StrokeThickness="3"/>
81                  <!-- draw Kinect -->
82                  <Ellipse Canvas.Top="34" Canvas.Left="150" Fill="White"
                      Height="10" Width="10" Stroke="Black"/>
83                  <Rectangle Canvas.Top="10" Canvas.Left="185" Width="3"
                      Height="18" Fill="DarkGray" />
84                  <Rectangle Canvas.Top="19" Canvas.Left="170" Width="20"
                      Height="10" Fill="Black" />
85                  <Rectangle Canvas.Top="22" Canvas.Left="140" Width="80"
                      Height="18" Fill="Black"/>
86              </Canvas>
87          </Viewbox>
88          <Viewbox Grid.Row="2" Grid.Column="2" Width="400" Height="300"
              Margin="5">
89              <Canvas Name="mapXZ" Width="360" Height="290">
90                  <!-- here will be drawn the eyes point-->
91              </Canvas>
92          </Viewbox>
93      </Grid>
94  </Window>
95
```

APPENDICE 4:
IMMAGINI FRONTALI DEI VOLONTARI CON METRO APPOSTO E RIFERIMENTI

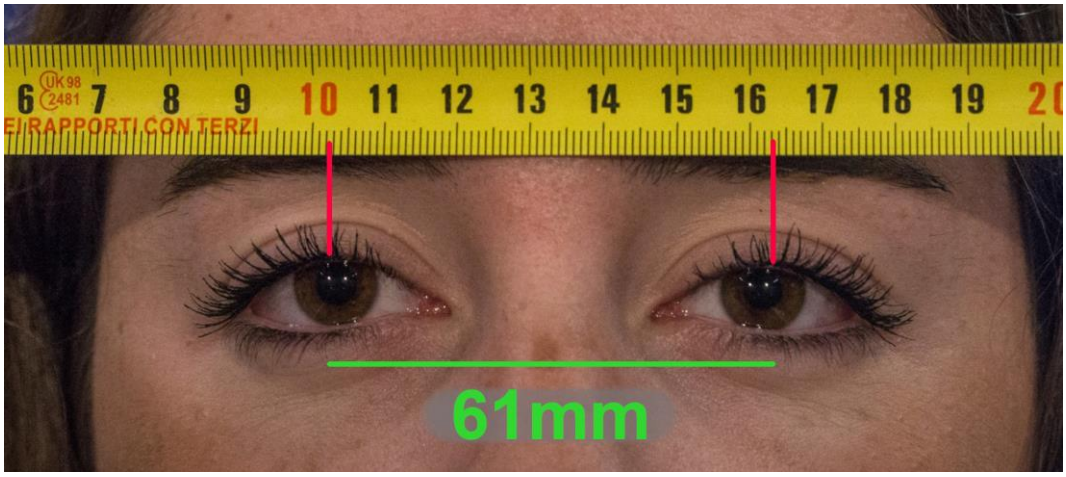


Figura 0-A: Volontario 1.

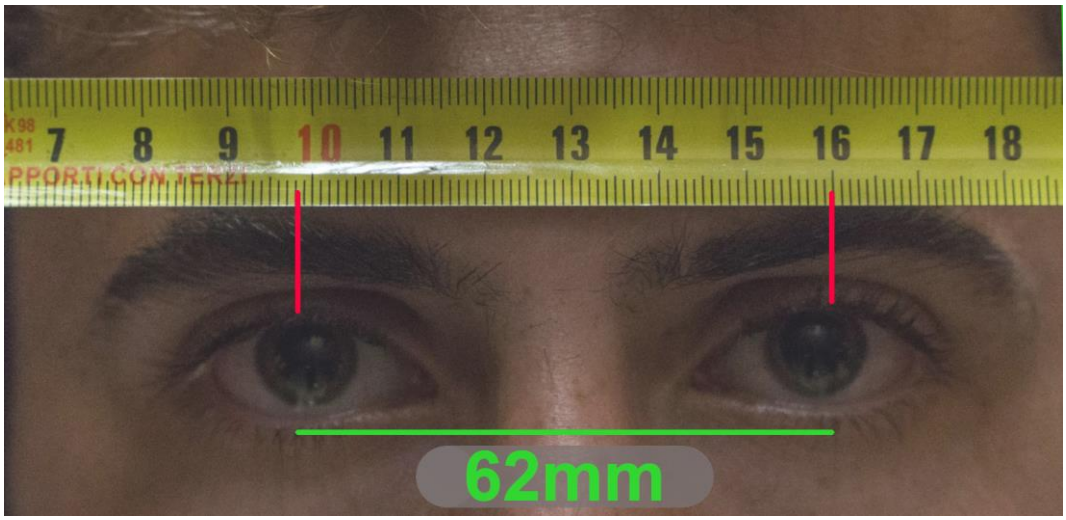


Figura 0-B: Volontario 2.

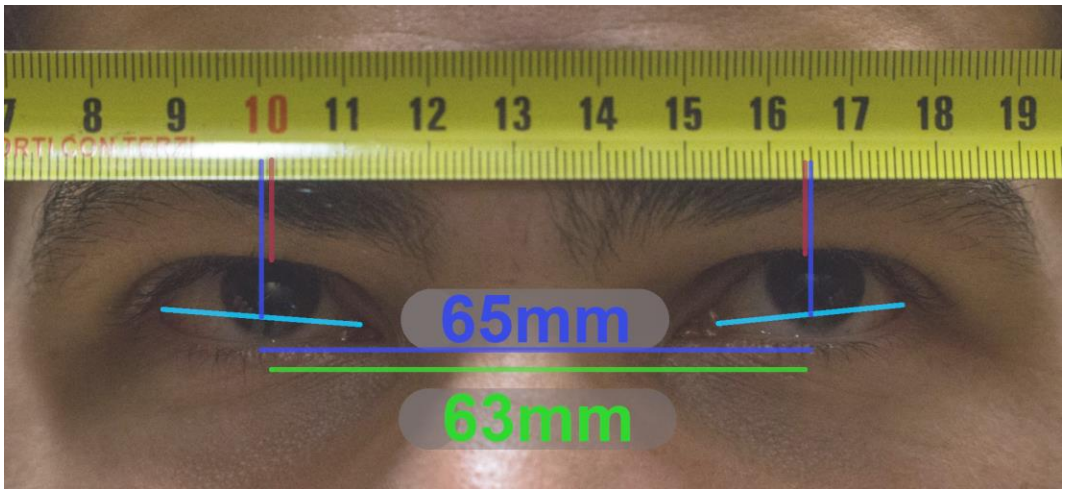


Figura 0-C: Volontario 3. Misura col metodo degli angoli oculari.

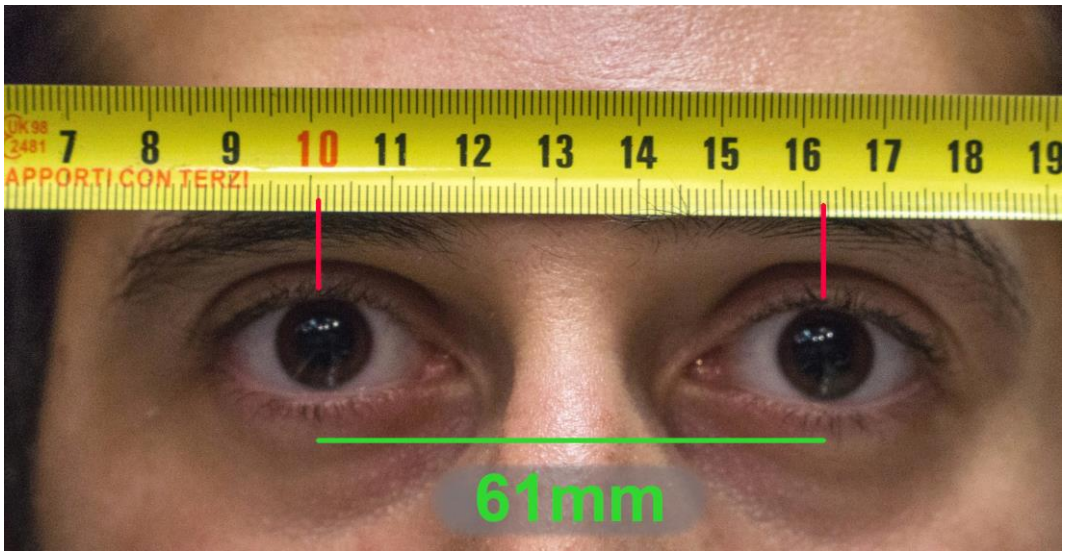
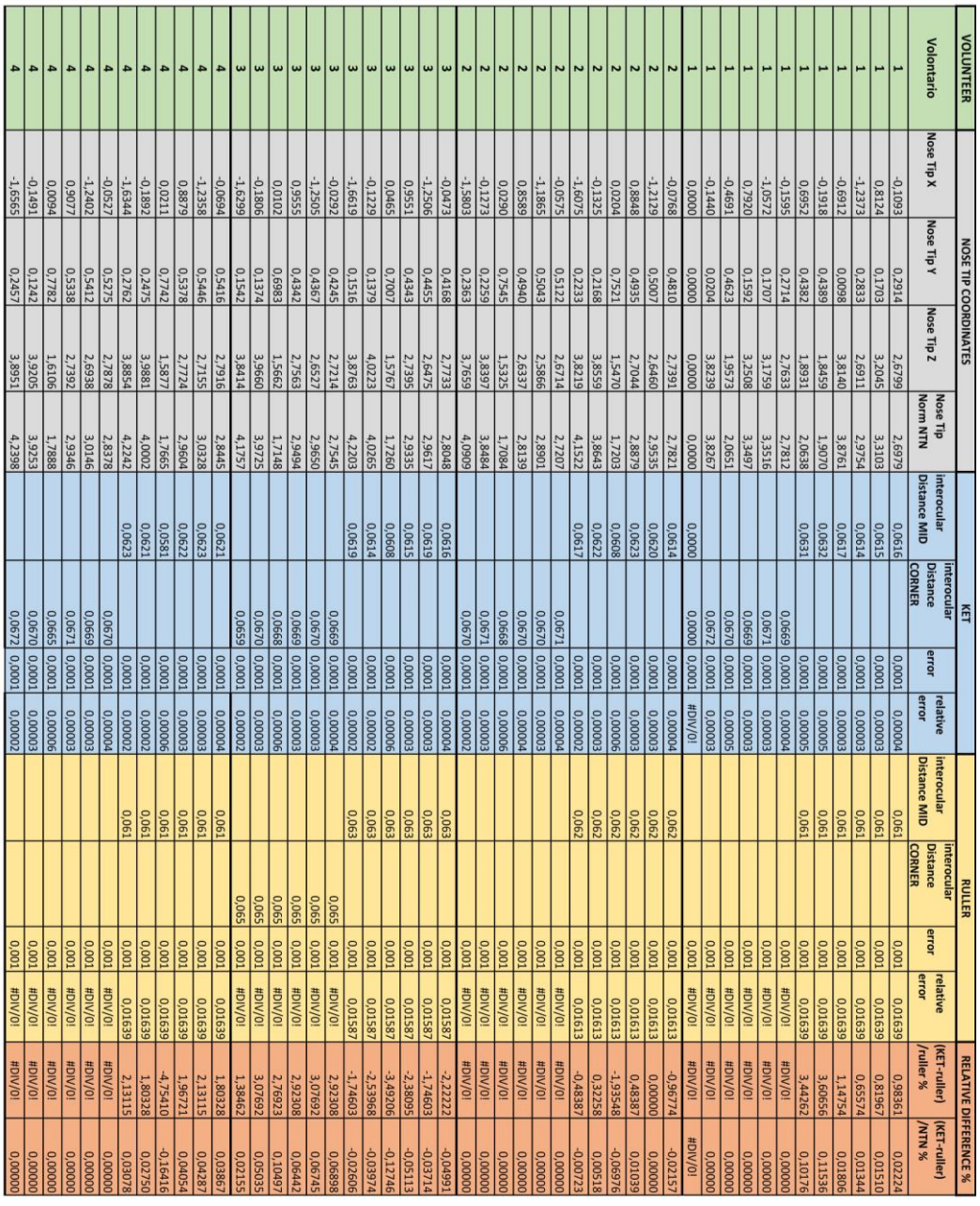


Figura 0-D: Volontario 4.

APPENDICE 5:
DATI GREZZI DELLE PROVE DI MISURA DELLE DISTANZE INTEROCULARI



VOLUNTEER		NOSE TIP COORDINATES				KET			RULLER			RELATIVE DIFFERENCE %	
	Volontario	Nose Tip X	Nose Tip Y	Nose Tip Z	Nose Tip Norm NTN	interocular Distance MID	interocular Distance CORNER	error	relative error	interocular Distance MID	interocular Distance CORNER	error	relative error	(KET-ruller) /ruller %	(KET-ruller) /NTN %
1	1	-0,1093	0,2914	2,6799	2,6979	0,0616		0,0001	0,00004	0,061		0,001	0,01639	0,98361	0,02224
1	1	0,8124	0,1703	3,2045	3,3103	0,0615		0,0001	0,00003	0,061		0,001	0,01639	0,81967	0,01510
1	1	-1,2373	0,2833	2,6911	2,9754	0,0614		0,0001	0,00003	0,061		0,001	0,01639	0,65574	0,01344
1	1	-0,6912	0,0098	3,8140	3,8761	0,0617		0,0001	0,00003	0,061		0,001	0,01639	1,14754	0,01806
1	1	-0,1918	0,4389	1,8459	1,9070	0,0632		0,0001	0,00005	0,061		0,001	0,01639	3,60656	0,11536
1	1	0,6952	0,4382	1,8931	2,0638	0,0631		0,0001	0,00005	0,061		0,001	0,01639	3,44262	0,10176
1	1	-0,1595	0,2714	2,7633	2,7812		0,0669	0,0001	0,00004			0,001	#DIV/0!	#DIV/0!	0,00000
1	1	-1,0572	0,1707	3,1759	3,3516		0,0671	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
1	1	0,7920	0,1592	3,2508	3,3497		0,0669	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
1	1	-0,4691	0,4623	1,9573	2,0651		0,0670	0,0001	0,00005			0,001	#DIV/0!	#DIV/0!	0,00000
1	1	-0,1440	0,0204	3,8239	3,8267		0,0672	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
1	1	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0001	#DIV/0!	0,062		0,001	#DIV/0!	#DIV/0!	0,00000
2	2	-0,0768	0,4810	2,7391	2,7821	0,0614		0,0001	0,00004	0,062		0,001	0,01613	-0,96774	-0,02157
2	2	-1,2129	0,5007	2,6460	2,9535	0,0620		0,0001	0,00003	0,062		0,001	0,01613	0,00000	0,00000
2	2	0,8848	0,4935	2,7044	2,8879	0,0623		0,0001	0,00003	0,062		0,001	0,01613	0,48387	0,01039
2	2	0,0204	0,7521	1,5470	1,7203	0,0608		0,0001	0,00006	0,062		0,001	0,01613	-1,93548	-0,06976
2	2	-0,1325	0,2168	3,8559	3,8643	0,0622		0,0001	0,00003	0,062		0,001	0,01613	0,32258	0,00518
2	2	-1,6075	0,2233	3,8219	4,1522	0,0617		0,0001	0,00002	0,062		0,001	0,01613	-0,48387	-0,00723
2	2	-0,0575	0,5122	2,6714	2,7207		0,0671	0,0001	0,00004			0,001	#DIV/0!	#DIV/0!	0,00000
2	2	-1,1865	0,5043	2,5866	2,8901		0,0670	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
2	2	0,8589	0,4940	2,6337	2,8139		0,0670	0,0001	0,00004			0,001	#DIV/0!	#DIV/0!	0,00000
2	2	0,0290	0,7545	1,5325	1,7084		0,0668	0,0001	0,00006			0,001	#DIV/0!	#DIV/0!	0,00000
2	2	-0,1273	0,2259	3,8397	3,8484		0,0671	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
2	2	-1,5803	0,2363	3,7659	4,0909		0,0670	0,0001	0,00002			0,001	#DIV/0!	#DIV/0!	0,00000
3	3	-0,0473	0,4168	2,7733	2,8048	0,0616		0,0001	0,00004	0,063		0,001	0,01587	-2,22222	-0,04991
3	3	-1,2506	0,4455	2,6475	2,9617	0,0619		0,0001	0,00003	0,063		0,001	0,01587	-1,74603	-0,03714
3	3	0,9551	0,4343	2,7395	2,9335	0,0615		0,0001	0,00003	0,063		0,001	0,01587	-2,38095	-0,05113
3	3	0,0465	0,7007	1,5767	1,7260	0,0608		0,0001	0,00006	0,063		0,001	0,01587	-3,49206	-0,12746
3	3	-0,1229	0,1379	4,0223	4,0265	0,0614		0,0001	0,00002	0,063		0,001	0,01587	-2,53968	-0,03974
3	3	-1,6619	0,1516	3,8763	4,2203	0,0619		0,0001	0,00002	0,063		0,001	0,01587	-1,74603	-0,02606
3	3	-0,0292	0,4245	2,7214	2,7545		0,0669	0,0001	0,00004		0,065	0,001	#DIV/0!	2,92308	0,06898
3	3	-1,2505	0,4367	2,6527	2,9650		0,0670	0,0001	0,00003		0,065	0,001	#DIV/0!	3,07692	0,06745
3	3	0,9555	0,4342	2,7563	2,9494		0,0669	0,0001	0,00003		0,065	0,001	#DIV/0!	2,92308	0,06442
3	3	0,0102	0,6983	1,5662	1,7148		0,0668	0,0001	0,00006		0,065	0,001	#DIV/0!	2,76923	0,10497
3	3	-0,1806	0,1374	3,9660	3,9725		0,0670	0,0001	0,00003		0,065	0,001	#DIV/0!	3,07692	0,05035
3	3	-1,6299	0,1542	3,8414	4,1757		0,0659	0,0001	0,00002		0,065	0,001	#DIV/0!	1,38462	0,02155
4	4	-0,0694	0,5416	2,7916	2,8445	0,0621		0,0001	0,00004	0,061		0,001	0,01639	1,80328	0,03867
4	4	-1,2358	0,5446	2,7155	3,0328	0,0623		0,0001	0,00003	0,061		0,001	0,01639	2,13115	0,04287
4	4	0,8879	0,5378	2,7724	2,9604	0,0622		0,0001	0,00003	0,061		0,001	0,01639	1,96721	0,04054
4	4	0,0211	0,7742	1,5877	1,7665	0,0581		0,0001	0,00006	0,061		0,001	0,01639	-4,75410	-0,16416
4	4	-0,1892	0,2475	3,9881	4,0002	0,0621		0,0001	0,00002	0,061		0,001	0,01639	1,80328	0,02750
4	4	-1,6344	0,2762	3,8854	4,2242	0,0623		0,0001	0,00002	0,061		0,001	0,01639	2,13115	0,03078
4	4	-0,0527	0,5275	2,7878	2,8378		0,0670	0,0001	0,00004			0,001	#DIV/0!	#DIV/0!	0,00000
4	4	-1,2402	0,5412	2,6938	3,0146		0,0669	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
4	4	0,9077	0,5338	2,7392	2,9346		0,0671	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
4	4	0,0094	0,7782	1,6106	1,7888		0,0665	0,0001	0,00006			0,001	#DIV/0!	#DIV/0!	0,00000
4	4	-0,1491	0,1242	3,9205	3,9253		0,0670	0,0001	0,00003			0,001	#DIV/0!	#DIV/0!	0,00000
4	4	-1,6565	0,2457	3,8951	4,2398		0,0672	0,0001	0,00002			0,001	#DIV/0!	#DIV/0!	0,00000

Tabella 4: Dati Grezzi rilevati. Tutte le dimensioni lineari sono in metri.

INDICE DELLE FIGURE

INDICE DEI GRAFICI

INDICE DELLE TABELLE

BIBLIOGRAFIA

- AA.VV (2017), “Introduzione a C#”, Microsoft.com.
- Allison, R., Gillam, B. and Vecellio, E. (2009), “Binocular depth discrimination and estimation beyond interaction space”, Journal of Vision 9(1), 10:1–14.
- Azuma, R. T. (1997), “A survey of augmented reality”.
- Cutting, J. E. and Vishton, P. M. (1995), “Perceiving layout and knowing distances: The integration, relative potency, and contextual use of different information about depth”, in W. Epstein and S. Rogers, eds, ‘Handbook of perception and cognition’, Vol. 5, Academic press, San Diego, CA., pp. 69–117.
- Ellis S. R. (2006), “Towards Determination of Visual Requirements for Augmented Reality Displays and Virtual Environments for the Airport Tower”.
- Helmholtz, H. von (1909/1925), “Physiological optics”, vol. 3, trans. J. P. C. Southall. Optical Society of America.
- Howard, H. J. (1919), “A test for the judgment of distance”. Transactions of the American Ophthalmological Society, 17, 195–235.
- Nagata, S. (1981), “Visual depth sensitivities of various cues for depth perception”. NHK Laboratories Note, 266, 1–11.
- Nagata, S. (1991), “How to reinforce perception of depth in single two-dimensional pictures”, in S.R. Ellis, M. Kaiser and A. Grunwald, eds, ‘Pictorial communication in virtual and real environments’, Taylor & Francis, pp. 527–544.

- Palmisano, S., Gillam, B., Govan, D. G., Allison, R. S. and Harris, J. M. (2010), “Stereoscopic perception of real depths at large distances”, Journal of vision 10(6), 19:1–16.
- Ronald J. Reisman, David M. Brown (2006), “Design of Augmented Reality Tools for Air Traffic Control Towers”.
- Sharma A. (2014) “Augmented tools with transparent displays”. Massachusetts Institute of Technology.