Università di Pisa

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Realtà aumentata per operazioni di manutenzione e servicing: uso di tag multipli per la registrazione automatica di impianti di grandi dimensioni

Tesi di

Cecchi Tommaso

Relatori:

Dott. Ing. Tecchia Franco .....

Prof. Domenici Andrea.....

Sessione di Laurea 9 luglio 2009

Anno Accademico 2008-2009

Introduzione

Il lavoro di tesi riassunto nel presente documento si inserisce nell'ampia tematica delle metodologie informatiche innovative di potenziale applicazione in ambito industriale. In particolare, si sono volute sperimentare e, dove possibile, affinare le potenzialità dei metodi tipici della Realtà Aumentata (approfonditamente definita in un capitolo dedicato) applicati alla tematica della manutenzione di macchinari complessi. Ci si è inoltre voluti riferire ad un caso industriale specifico, attraverso l'attiva collaborazione con un'azienda fortemente interessata alla tematica.

In questo contesto, l'obiettivo prefisso per questo lavoro di tesi è di investigare secondo quali modalità e con quali problematiche sia possibile utilizzare tecniche di computer vision per sovra - imporre modelli CAD tridimensionali a macchinari di grandi dimensioni.

Questi risultati sono possibili attraverso l'uso di tecniche attualmente consolidate nella letteratura specifica, ma la cui pratica applicazione dimostra in più occasioni il permanere di alcune insidie e difficoltà operative che rappresentano di fatto una barriera al loro utilizzo fattivo. Nella presente si sono voluti investigare tali limiti, proponendo poi un approccio al problema che permettesse di superare alcuni di questi; in particolare ci si è proposti di automatizzare una delle fasi più complesse e laboriose, *l'instrumentazione del macchinario*, ossia la pratica necessaria a far sì che posizione e orientamento di un macchinario complesso inquadrato da una telecamera sia conosciuta dal calcolatore, in maniera da poter sovrimporre ad esso il modello CAD corrispondente.

Figura 1: Applicazione di markers ad un macchinario industriale e relativa sovraimpressione del modello CAD

Si è data alla presente tesi un'organizzazione in 6 capitoli principali:

Il **Capitolo 1** offre una panoramica generale sulle tematiche inerenti la Realtà Aumentata: espone i vari ambiti applicativi in cui può essere impiegata, i vari metodi di tracking e presenta la tipica architettura di un sistema AR.

Il **Capitolo 2** analizza il funzionamento del tracking ottico basato su marker ed in particolare la stima della posizione della camera una volta che l'immagine del marker è stata acquisita. Vengono inoltre presentate ed analizzate varie librerie di sviluppo basate su questi algoritmi.

Il **Capitolo 3** descrive il caso in cui più marker sono utilizzati contemporaneamente. Espone le varie tecniche presenti per la creazione automatica del *marker field* e descrive dettagliatamente l'algoritmo implementato per ottenerne una produzione *automatica*.

Il **Capitolo 4** presenta dettagli implementativi su i moduli sviluppati durante la presente tesi, ed in particolare per l'algoritmo di creazione automatica del marker field e per la visualizzazione dei modelli virtuali.

Il **Capitolo 5** espone i risultati dei test effettuati sull'algoritmo in varie condizioni operative ed analizza l'accuratezza con la quale viene creato il marker field.

Infine, il **Capitolo 6** espone le conclusioni finali e i possibili sviluppi futuri del presente lavoro.

Indice

Elenco delle figure

Capitolo 1

La Realtà Aumentata

La *realtà aumentata* (o AR: Augmented Reality) è una tecnologia che permette ad oggetti virtuali, creati al computer, di essere sovrapposti esattamente ad oggetti fisici reali, in tempo reale.

Essa è una variante della realtà virtuale: mentre lo scopo della realtà virtuale è quello di creare un ambiente totalmente artificiale che un utilizzatore può esplorare interattivamente attraverso i propri sensi (vista, audio, tatto), la realtà aumentata si propone invece di accrescere la percezione del mondo reale. Questo obiettivo può essere perseguito aggiungendo informazioni generate dal computer (testo, immagini 3D, suoni ecc..) alla realtà percepita dall'utente del sistema AR. Le informazioni generate artificialmente possono essere utilizzate per vari scopi, ad esempio possono aiutare un tecnico a svolgere più velocemente ed efficacemente operazioni di assemblaggio, disassemblaggio e manutenzione.

Ad oggi esistono principalmente due definizioni ugualmente riconosciute di AR. Secondo la definizione proposta da R. Azuma [4], un sistema AR deve

avere le seguenti proprietà:

- deve combinare il reale con il virtuale
- deve essere eseguito interattivamente e in real time
- deve allineare oggetti reali con quelli virtuali e viceversa

La prima proprietà, e cioè quella di combinare il mondo reale con contenuti virtuali, è quella fondamentale per un sistema AR. La seconda richiede che il sistema reagisca agli input dell'utente e si aggiorni in real time. La terza proprietà distingue la realtà aumentata dal concetto più generale di mixed reality: le immagini virtuali 3D devono essere allineate geometricamente agli oggetti reali nel mondo reale.

Con la capacità dei computer di generare in real-time grafica 3D, questi possono visualizzare su un display ambienti artificiali che possono dare all'utente l'impressione di essere immerso in un mondo virtuale: questa tecnologia viene definita come realtà virtuale (VR) ed è utilizzata per simulare al computer il mondo fisico che una persona può normalmente vedere. Opposto alla realtà virtuale c'è il mondo reale, anche se può apparire leggermente modificato se viene visto attraverso un head-mounted display o una videocamera (distorsione, cambiamento di luminosità, definizione dell'immagine ecc..). La realtà aumentata è quindi una combinazione del reale e del virtuale: secondo il reality-virtuality continuum proposto da Milgram [21] (Figura 1.1 nella pagina seguente), l'AR è una delle possibili manifestazione del mixed reality, che unisce il reale e il virtuale in un unico display.

Se l'ambiente reale si trova ad un estremo del continuum e la realtà virtuale si trova all'estremo opposto, l' AR si inserisce nella zona occupata dalla mixed reality, accanto al mondo reale. Più un sistema è prossimo alla realtà virtuale e più gli elementi reali si riducono, come per esempio nell'Augmented Virtuality, dove un ambiente virtuale viene integrato con immagini reali.

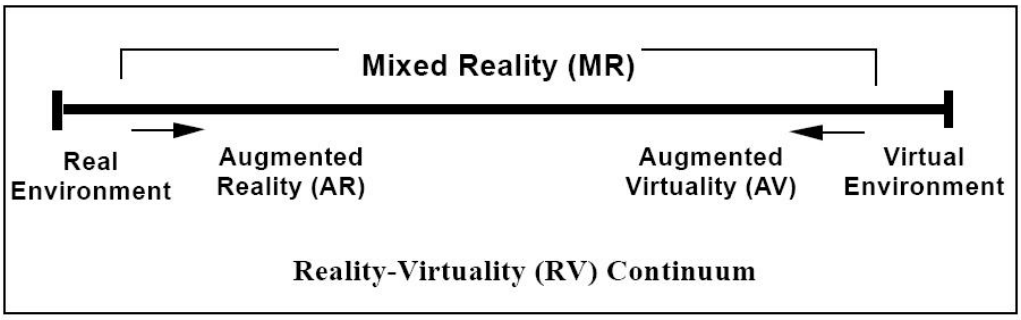


Figura 1.1: Reality-Virtuality Continuum di Milgram [21]

1.1 Panoramica Storica

Il primo sistema AR completamente funzionante risale al 1968 quando Ivan Sutherland e il suo team costruì un see-through head mounted display (HMD) con tracking meccanico, mediante il quale l'utilizzatore del dispositivo poteva vedere le informazioni virtuali generate dal computer insieme agli oggetti fisici. Negli anni 70 e 80 la realtà aumentata diventò un' area di ricerca in diverse istituzioni americane come l'U.S. Air Force's Armstrong Laboratory, il NASA Ames Research Center, il Massachusetts Institute Of Technology e l'università del North Carolina a Chapell Hill.

Fu solo nel 1990, nei centri di ricerca della Boeing Corporation, che l'idea di sovrapporre la grafica generata dal computer al mondo reale ha ricevuto il suo nome attuale. Due ricercatori della Boeing cercando di semplificare l'assemblaggio dell'impianto elettrico per le persone addette alla costruzione degli aerei, si riferirono alla soluzione da loro proposta, e cioè di sovrapporre le informazioni virtuali alle immagini reali, come realtà aumentata.

A metà degli anni 90, grazie alla continua miniaturizzazione dei componenti elettronici e alla maggiore potenza di essi, cominciarono ad essere implementati i primi dispositivi AR mobili (MARS: Mobile Augmented Reality System), composti da un computer mobile, dispositivi di tracking e un HMD. Uno dei primi prototipi fu il Columbia *Touring Machine* (Figura 1.2 nella pagina successiva), che presentava informazioni grafiche 3D ai visitatori del campus, relativamente agli edifici che l'utente stava attualmente guardando

attraverso l'HMD.

Nell'ottobre del 1998 si è tenuta l'*International Workshop on Augmented Reality*, la prima conferenza sulla realtà aumentata, presso San Francisco.

(a) Touring Machine MARS (b) Informazioni visualizzate sul display

Figura 1.2: Prototipo MARS Columbia *Touring Machine* [16]

Grazie al continuo sviluppo tecnologico, oggi sono stati sviluppati sistemi AR in grado di funzionare anche su PDA e telefoni cellulari.

1.2 Applicazioni di un sistema AR

1.2.1 Assemblaggio, Manutenzione e Riparazione

Un ambito in cui la realtà aumentata può risultare molto utile è quello industriale dove può essere utilizzata per assistere gli operai nell'assemblaggio, nella manutenzione oppure nella riparazione di determinati macchinari [16] [27]. Molte industrie, sempre più spesso, cercano di spostare le fasi di progettazione e costruzione al compuetr, in modo da sostituire i prototipi fisici con prototipi virtuali per il packaging, assemblaggio e per effettuare dei test sicuri. L'utilità di questo risulta particolarmente evidente nel caso di compagnie di costruzione di aerei ed auto, dove i prototipi fisici sono costosi e il time-to-market è un fattore determinante. Le istruzioni per assemblare

o riparare macchinari infatti, sono più semplici da capire se sono disponibili sotto forma di immagini 3D e non come manuali cartacei: ad esempio possono essere generate istruzioni su un HMD, sovrapposte al dispositivo reale, che mostrano con delle animazioni (ad esempio frecce che indicano la posizione ed il verso in cui inserire un pezzo) i passi da compiere per l'assemblaggio o la riparazione (Figura 1.3). In questo modo può essere notevolmente velocizzato il processo di produzione e manutenzione e ridotta anche la probabilità di errore da parte di un operaio.

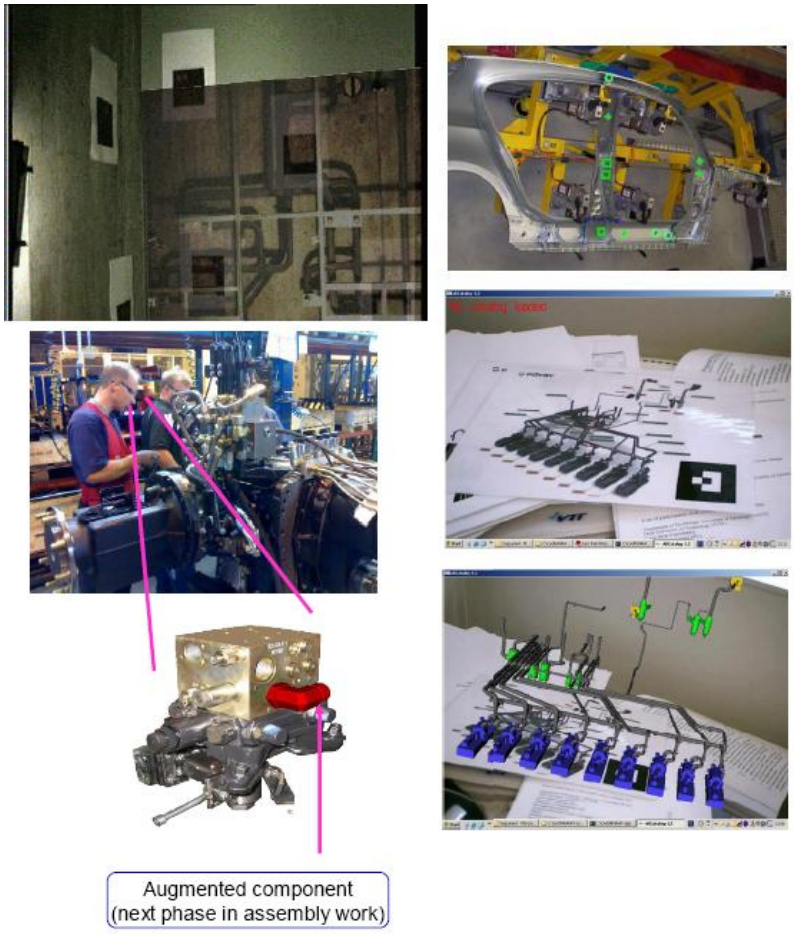


Figura 1.3: Esempio di utilizzo dell'AR in ambito industriale [35] [18]

La Boeing, come già detto precedentemente, è stata la prima ad utilizzare

l'AR per questo scopo, sviluppando un dispositivo che permetteva di assistere gli operai nell'assemblaggio del circuito elettrico degli aerei visualizzando informazioni 3D su un display.

Negli ultimi anni sono nati molti progetti riguardo l'AR in ambito industriale come per esempio ARVIKA[1], AugAsse[2], PLAMOS[3].

1.2.2 Medicina

La realtà aumentata nel campo della medicina può venire in aiuto dei chirurghi in fase di studio di un intervento su un paziente. Mediante sensori come il *Magnetic Resonance Imaging* (MRI) o il *Computed Tomography scan* (CT) è possibile ottenere immagini tridimensionali del paziente. Queste immagini, visualizzate su un display HMD, possono essere allineate e sovrapposte alla vista reale del paziente. Questo risulta utile in caso di un intervento di chirurgia mini invasiva permettendo al chirurgo di avere una vista interna del corpo mediante immagini tridimensionali sovrapposte a quelle reali, senza la necessità di ricorrere a profonde incisioni (Figura 1.4 nella pagina successiva).

I medici possono utilizzare la realtà aumentata anche durante il loro giro di visita ai pazienti, visualizzando informazioni su di essi direttamente su un display invece di leggere le cartelle cliniche. Può essere utilizzata anche per scopi didattici, fornendo ai nuovi chirurghi istruzioni tridimensionali sovrapposte al paziente per compiere una determinata operazione, senza la necessità quindi di distogliere lo sguardo dal paziente per leggere il manuale.

1.2.3 Archeologia e Architettura

La realtà aumentata, in questo ambito, può essere utilizzata, per esempio, per visualizzare in modo tridimensionale, nella realtà, i progetti di edifici o ponti che verranno poi costruiti in un particolare sito, oppure per ricostruire

[1] http://www.arvika.de/www/e/home/home.htm
[2] www.vtt.fi/proj/augasse
[3] http://virtual.vtt.fi/virtual/proj2/multimedia/index.html

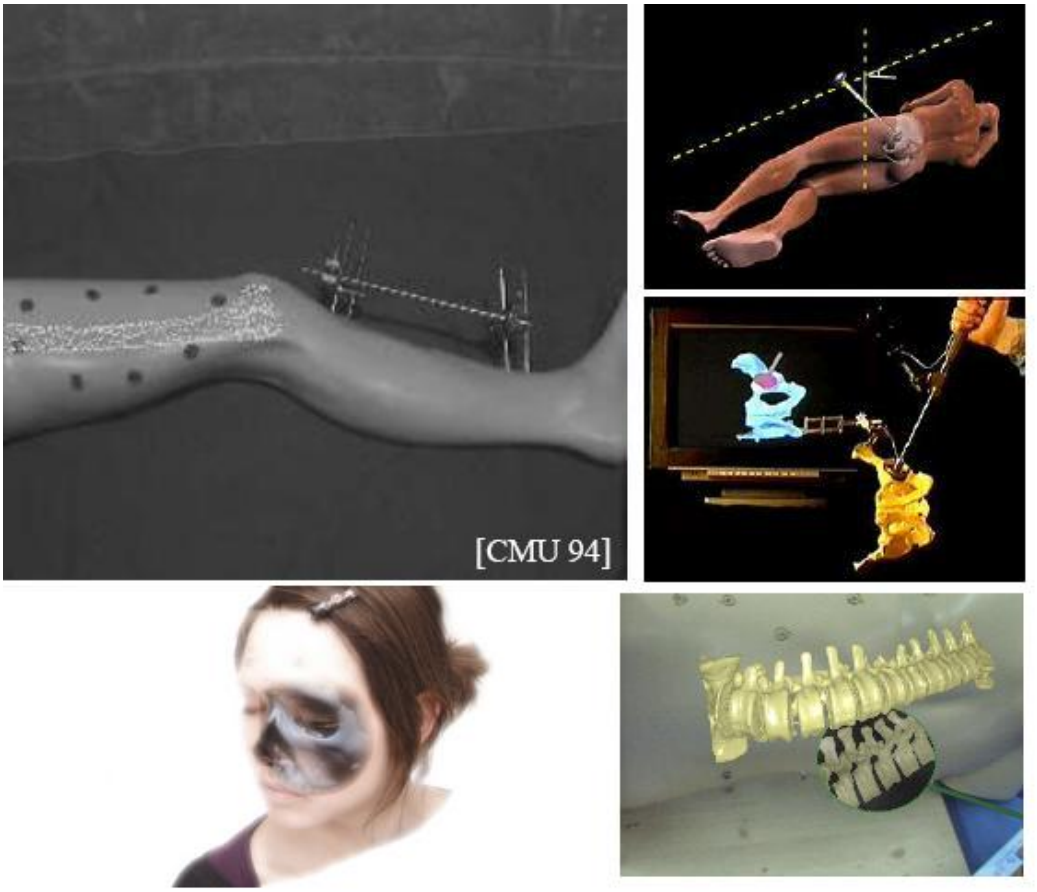


Figura 1.4: Realtà Aumentata applicata alla medicina [18]

siti archeologici o per visualizzare antichi edifici nella loro locazione originale [16].

Un esempio è il progetto *ARCHEOGUIDE*[4] sponsorizzato dalla comunità europea che permette di ricostruire siti archeologici, ricostruendo in 3D gli edifici mancanti e visualizzando le parti mancanti di un manufatto su un HMD (Figura 1.5 nella pagina seguente).

1.2.4 Turismo

E' possibile utilizzare la realtà aumentata non solo per guidare un turista verso una particolare destinazione, ma anche per visualizzare informazioni.

[4]http://www.archeoguide.it/

Figura 1.5: Esempi di AR in architettura e archeologia (progetto ARCHEOGUIDE) [18]

Invece di avere informazioni (storiche e non) su una guida turistica, il turista può visualizzarle su un HMD mentre sta guardando l'edificio o il luogo d'interesse.

Ad esempio il *Touring Machine* della Columbia University permette di guidare un turista all'interno del campus universitario e di ottenere informazioni 3D relative ad un certo edificio su un HMD.

1.2.5 Intrattenimento

Può essere utilizzata poi per combinare attori reali con ambienti virtuali, riducendo così i costi di produzione di un film. L'attore si muove davanti ad un grande schermo blu, mentre una telecamera mobile registra la scena. Poiché la posizione della telecamera in ogni istante è nota così come la posizione dell'attore, è possibile allora far muovere l'attore in un ambiente 3D. Anche i videogames possono far uso della realtà virtuale: esempi sono *ARQuake*[5] (Figura 1.6 nella pagina successiva), sviluppato dal *Wereable Computer Lab* alla *University of South Australia* che permette di combattere contro mostri virtuali camminando in un ambiente reale, oppure *The Eye Of Judgement* per Playstation 3.

[5]ARQuake: http://wearables.unisa.edu.au/arquake/

Figura 1.6: ARQuake: Quake in versione Augmented Reality

1.2.6 Campo Militare

Molte missioni militari hanno luogo in territori sconosciuti ai soldati. La realtà aumentata può essere utilizzata per mostrare una mappa tridimensionale del campo di battaglia, fornendo informazioni aggiuntive, come, ad esempio, nomi delle strade, a cose è adibito un particolare edificio (ospedale, stazione di comunicazione), la posizione dei soldati nemici, il tutto senza distogliere il soldato dalla vista reale dell'ambiente ed evitare quindi situazioni pericolose per la propria vita.

1.3 Architettura di un Sistema AR

In una architettura AR sono sempre presenti quattro componenti: un display, dei dispositivi di tracking, dei dispositivi per acquisizione di immagini (come per esempio webcam, videocamere) e un computer portatile per generare la grafica tridimensionale.

Ci soffermeremo sui primi due componenti in quanto sono quelli che hanno il maggior impatto nell'architettura di un sistema di realtà aumentata.

1.3.1 Display

I display impiegati nella realtà aumentata sono hardware che impiegano componenti ottiche ed elettroniche per generare immagini nella traiettoria

visiva tra gli occhi dell'osservatore e l'oggetto fisico. La figura 1.7 mostra una classificazione dei vari tipi di display in funzione della posizione occupata rispetto all'osservatore e all'oggetto osservato.

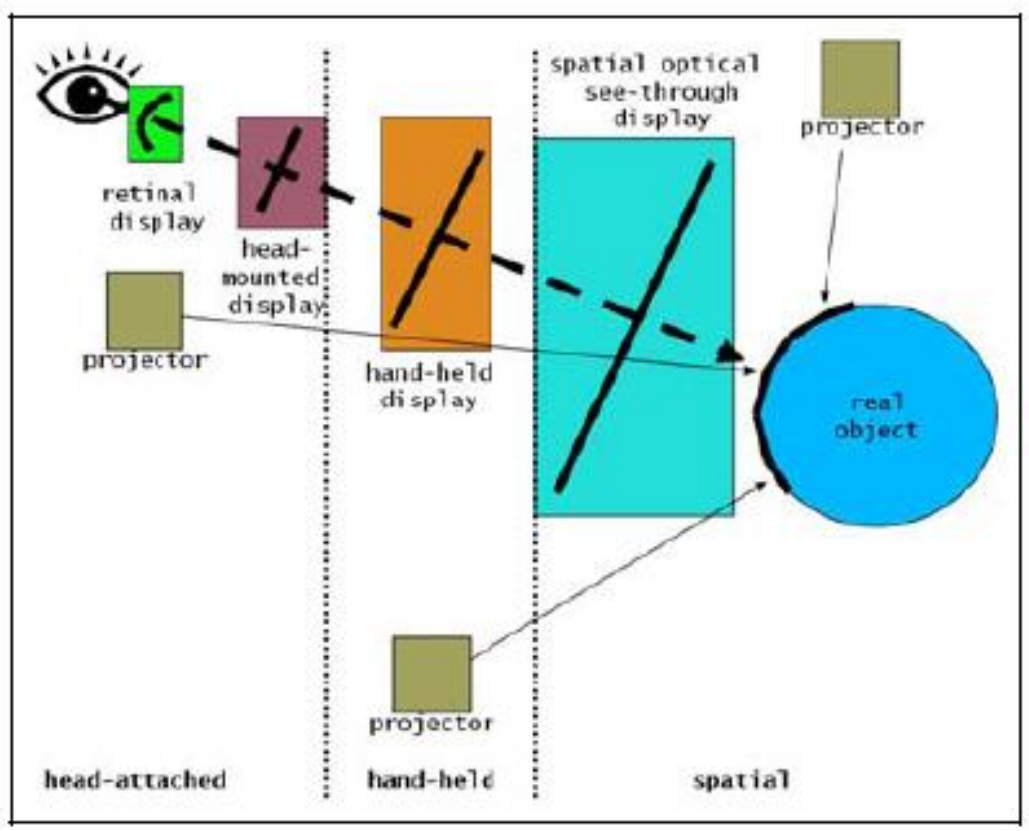


Figura 1.7: Classificazione dei vari displays

I *near-eye displays* (o *head-attached displays)*, come ad esempio i retinal displays, gli head-mounted displays ed gli head-mounted projectors, devono essere indossati dall'osservatore. Altri tipi di display sono invece *hand-held*, cioè tenuti tra le mani dall'osservatore, come per esempio i cellulari o palmari. Infine la terza categoria comprende quei displays che sono allineati spazialmente tra l'osservatore e l'oggetto reale. Questi generalmente fanno uso di videoproiettori ed apposite superfici sulle quali visualizzae le immagini [38] [25].

Sono stati individuati diversi fattori che devono essere considerati nella scelta di quale tecnologia utilizzare per visualizzare le informazioni:

- Latenza: quantità di tempo che intercorre da quando l'utente si muove a quando l'immagine finale viene visualizzata sul display.
- Risoluzione della scena reale e distorsione: risoluzione con la quale la realtà viene presentata all'utente e quali cambiamenti sono introdotti dall'ottica.
- Campo visivo: porzione di vista permessa dal display all'utente.
- Fattori di costo: In certe applicazioni sono richieste ottiche complesse che alzano i costi.

Inoltre quando gli oggetti virtuali sono disegnati di fronte agli oggetti reali è desiderabile che l'oggetto virtuale faccia una corretta occlusione di quello reale.

1.3.1.1 Near-Eye Displays

Nei near-eye displays, il sistema di visualizzazione è indossato dall'utente sulla testa. In base alla tecnologia di generazione dell'immagine esistono tre diverse tipologie di near-eye displays:

- Head-mounted displays
- Retinal displays
- Head-mounted projectors

Gli head-mounted displays (HMDs) sono attualmente i sistemi più utilizzati nelle applicazioni di realtà aumentata.

Un *see-through HMD* è un dispositivo utilizzato per combinare il reale ed il virtuale. I *closed-view HMD* di tipo standard non permettono nessuna visione diretta della realtà. Diversamente invece un see-through display permette all'utente di vedere il mondo reale con oggetti virtuali sovraimpressi attraverso tecnologie ottiche o video.

Sono possibili due scelte tecnologiche per questi dispositivi: ottiche o video.

Un *optical see-through HMD* (Figura 1.8) funziona posizionando dei combinatori ottici di fronte agli occhi dell'utente. Questi combinatori sono parzialmente trasmissivi: in questo modo l'utente può guardare direttamente attraverso essi per vedere il mondo reale. Questi combinatori sono anche parzialmente riflessivi, in modo tale che l'utente possa vedere le immagini virtuali di fronte agli occhi.

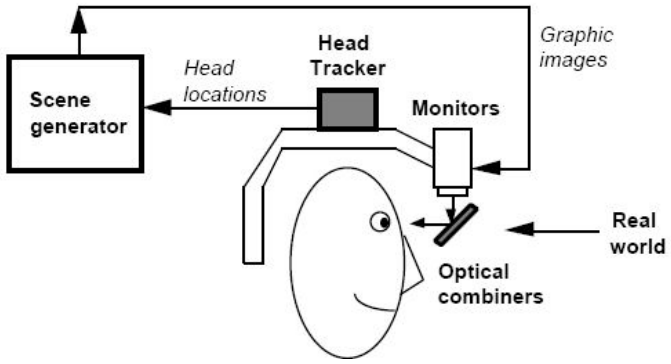


Figura 1.8: Optical See-Through Display [4]

Un *video see-through HMD* (Figura 1.9) funziona combinando un closed-loop HMD con una o più *head-mounted video cameras*, le quali forniscono all'utente le immagini del mondo reale. Il video fornito da queste videocamere viene combinato con le immagini generate dal computer, unendo quindi le immagini reali con quelle virtuali. Il risultato viene inviato ad un monitor posto di fronte agli occhi dell'utente.

L'approccio ottico ha i seguenti vantaggi rispetto a quello video [4]:

- Semplicità: unire le immagini virtuali e reali è più semplice ed economico rispetto alla tecnologia video. L'approccio ottico ha infatti un solo video stream di cui preoccuparsi: le immagini grafiche generate dal computer. Il mondo reale infatti è visto direttamente attraverso i combinatori ottici e generalmente il ritardo è di pochi nanosecondi. Nell'approccio video invece devono essere gestiti due flussi video: quello proveniente dalle videocamere e quello delle immagini virtuali. Questi sistemi hanno un ritardo in termini di decine di millisecondi.

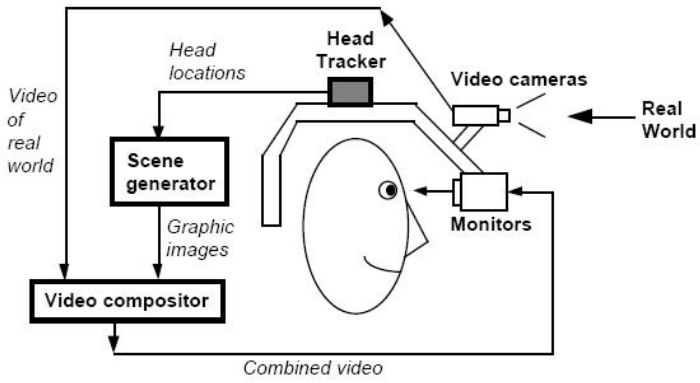


Figura 1.9: Video See-Through Display [4]

- Risoluzione: nella tecnologia video, la risoluzione con la quale l'utente vede sia l'immagine reale che quella virtuale è limitata alla risoluzione del display. Anche l'optical see-through visualizza le immagini virtuali alla risoluzione del display, ma l'immagine reale non viene degradata in quanto l'utente la vede direttamente con i propri occhi.

- Sicurezza: il video see-through HMD è essenzialmente un closed-view HMD modificato. In caso di mancata alimentazione l'utente è completamente cieco ed in certi scenari questo può essere un pericolo. Se invece l'alimentazione è rimossa da un optical-see through HMD, l'utente è ancora capace di vedere il mondo reale.

- No eye offset: con il video see-through, la vista del mondo reale viene fornita attraverso videocamere. In molte configurazioni le videocamete non sono poste esattamente dove si trovano gli occhi dell'utente, creando quindi un offset tra le videocamere e gli occhi delle persone. Questa differenza genera uno scostamento tra quello che l'utente vede rispetto a quello che si aspetterebbe di vedere. Per esempio se le videocamere sono poste sopra gli occhi dell'utente, allora vedrà il mondo reale da un punto di vista più alto.

In contrasto la tecnologia video offre i seguenti vantaggi rispetto a quella ottica:

- Larghezza del campo visivo: nei sistemi ottici le distorsioni sono funzioni della distanza radiale dall'asse ottico. Più uno guarda lontano dal centro di vista, più è elevata la distorsione. Una immagine digitale catturata attraverso un sistema ottico che presenta distorsione può essere corretta attraverso tecniche di elaborazione di immagini.

- Il ritardo di visualizzazione del reale e virtuale può essere uguagliato: con l'approccio video è possibile ridurre o evitare i problemi causati dalla differenza temporale con cui sono pronte per essere visualizzate le immagini reali e quelle virtuali. L'optical see-through HMD permette una visione istantanea del mondo reale ma l'immagine virtuale viene visualizzata con un certo ritardo. Con l'approccio video è possibile ritardare il video del mondo reale per uguagliare il ritardo dello stream dell'immagine virtuale.

- E' più facile uguagliare il colore e la brillantezza delle immagini reali e virtuali.

I retinal displays (RDs) (Figura 1.10) utilizzano laser a semiconduttori a bassa potenza per proiettare le immagini direttamente nella retina dell'occhio umano.

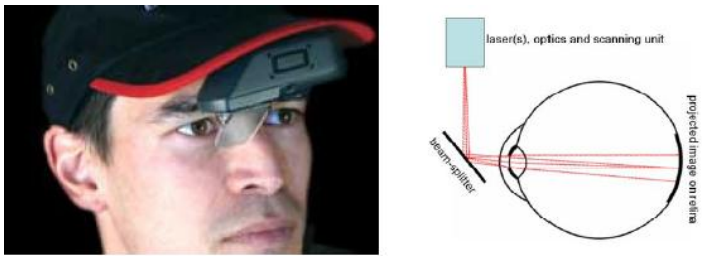


Figura 1.10: HMD retinal display e schema di funzionamento

I principali vantaggi di questi dispositivi sono:

- immagine più luminosa e più definita.

- campo visivo maggiore rispetto alle tecnologie basate sull'utilizzo di schermi.

- basso consumo energetico

Al contrario i principali svantaggi sono:

- immagini monocromatiche.
- la lunghezza focale risulta fissa
- non esistono versioni stereoscopiche di RDs.

Questi dispositivi sono adatti per applicazioni mobili in spazi all'aperto dove non si può avere un controllo sulla luce ambientale e dove è fondamentale tenere sotto controllo il consumo energetico.

Gli head-mounted projectors (HMPs) sono dispositivi che indirizzano il fascio di luce proiettata con un divisore di fascio ottico in modo tale che l'immagine sia diretta verso superfici retroriflettenti, collocate di fronte all'utente. Una superficie retroriflettente è costituita da migliaia di piccoli grano ognuno dei quali ha la proprietà ottica di riflettere la luce lungo la sua direzione di incidenza. Queste superfici permettono immagini più chiare delle normali superfici che diffondono la luce.

Gli HMPs fornisono un campo visivo maggiore rispetto agli HMDs ed evitano distorsioni dovute ad errori di parallasse causati dalla inter-pupil distance (IPD). Tuttavia la luminosità delle immagini dipende essenzialmente dall'illuminazione dell'ambiente circostante e gli attuali modelli di HMPs sono scomodi, ingombranti e troppo pesanti per essere utilizzati in ambiti professionali.

1.3.1.2 Hand-Held Displays

Esempi di hand-held displays sono cellulari, Palmtops, tablet PC. Questi dispositivi inglobano in un'unica apparecchiatura processore, memoria, display e dspositivo di acquisizione video e permettono di realizzare sistemi portatili senza fili (Figura 1.11).

Con questi dispositivi abbiamo un approccio video see-throguh: la videocamera presente nell'apparecchio cattura un flusso video dell'ambiente e

Figura 1.11: Handheld Display [3]

sul display viene visualizzata l'immagine catturata arricchita con immagini generate dal processore grafico.

I principali vantaggi degli HHDs sono rappresentati dalla semplicità d'uso, dalla compattezza e dalla limitata invasività. Tuttavia la loro limitata capacità di calcolo può causare un eccessivo ritardo nella sovrapposizione di immagini grafiche sull'immagine reale e la piccola dimensione dello schermo offre un limitato campo visivo.

1.3.1.3 Spatial Displays

A differenza dei dispositivi indossati dall'utente (come gli HMDs o HHDs), gli spatial displays separano la tecnologia dall'utente per integrarla nell'ambiente (Figura 1.12).

Esempi sono gli *Screen-Based Video See-Through* in cui le immagini arricchite dalle informazioni necessarie vengono trasmesse su un comune monitor per pc o altri tipi di monitor, oppure gli *Wall-mounted* displays che vengono integrati nell'area di lavoro

1.3.2 Tracking

Nella realtà aumentata il tracking è necessario per ottenere la posizione e l'orientamento di una persona, rispetto a un sistema di riferimento, in modo da ottenere un corretto allineamento (o *registration*) dell'immagine

(a) Monitor see-through display (b) Wall-mounted display

Figura 1.12: Spatial Displays

virtuale con quella reale. Quando l'utente cambia il suo punto di vista, l'immagine generata artificialmente deve rimanere allineata con la posizione e l'orientamento dell'oggetto reale osservato e quindi è necessario che il sistema di tracking aggiorni in real time la nuova posizione dell'utente. Il tracking può essere applicato a diverse parti del corpo, come per esempio alle mani, alla testa o anche a tutto il corpo. Nell'AR viene applicato alla testa in quanto viene utilizzato un HMD per vedere la realtà con le immagini tridimensionali sovrapposte. In questo caso si parla di *head tracking*.

Nel mercato sono presenti diversi dispositivi di tracking, le cui varie caratteristiche possono essere riassunte nelle seguenti [26]:

- Risoluzione: la misura della più piccola unità spaziale che il tracker può misurare.
- Accuratezza: intervallo entro il quale la posizione riportata può essere considerata corretta.
- Reattività del sistema:
  - Frequenza di campionamento: velocità con la quale il sensore controlla i dati, generalmente espressa come frequenza (Hz).
  - Data rate: quante volte viene rilevata la posizione in un secondo, espressa in frequenza (Hz).

  - Update rate: velocità con cui il sistema di tracking riporta la nuova posizione calcolata al computer, anche questa espressa in frequenza (Hz).
  - Latenza: ritardo tra il movimento dell'oggetto o dell'utente con il sistema di tracking e il calcolo della nuova posizione. Viene espresso in millisecondi (ms). Generalmente questo ritardo deve essere inferiore ai 60 ms.

- Range Operativo: misura dello spazio in cui il tracker riesce a funzionare.Ad esempio alcuni tracker utilizzano segnali emessi da sorgenti per calcolare la posizione. Più il sensore si allontana dalla sorgente, più il segnale ricevuto è attenuato. E' quindi chiaro che ci sarà una distanza massima oltre la quale il tracker non riuscirà a calcolare la posizione in quanto non riceve nessun segnale dalla sorgente.

- Costi: i costi variano a seconda della complessità del tracker e dell'accuratezza.

- Trasportabilità: sono importanti anche la grandezza e il peso se il tracker deve essere indossato.

La misura della posizione e dell'orientamento effettuate dai tracker è relativa ad una posizione e ad un orientamento di riferimento e viene effettuata mediante l'utilizzo di sorgenti di segnali e sensori in grado di intercettarli. A seconda di dove sono disposte queste due componenti i meccanismi di tracking vengono suddivisi in tre categorie:

- Inside-in: i sensori e le sorgenti si trovano entrambi sull'oggetto di cui si vuole misurare la posizione. Di solito questo tipo di meccanismo non è in grado di fare misurazioni assolute rispetto ad un sistema di riferimento esterno, ma solamente relativa ad uno stato iniziale.

- Inside-out: i sensori sono sul corpo mentre le sorgenti si trovano all'esterno in posizioni prefissate. Questi meccanismi sono in grado di fornire posizioni riferite ad un sistema di riferimento assoluto.

- Outside-in: le sorgenti si trovano sull'oggetto, mentre i sensori sono posti in punti esterni prefissati. Come i precedenti, anche questa tipologia di meccanismi è in grado di fornire coordinate assolute. Quest'approccio è sicuramente il meno intrusivo, poiché l'utente non deve essere equipaggiato con sensori, che possono essere voluminosi e che necessitano di alimentazione [22].

Esiste anche un altro metodo di suddivisione dei dispositivi di tracking che li raggruppa in tre categorie:

- Active-target: sistemi che includono sorgenti di segnali artificiali. Esempi di tali sistemi sono quelli che fanno uso di segnali magnetici, ottici, radio e acustici.

- Passive-target: sistemi che utilizzano l'ambiente o segnali che esistono in natura. Esempi sono bussole, girobussole e sistemi ottici che utilizzano *markers* (o *fiducials*) o caratteristiche naturali dell'ambiente (*natural features*)

- Inertial: i sistemi inerziali misurano fenomeni fisici quali l'accelerazione o il momento angolare. Esempi sono gli accelerometri e i giroscopi.

1.3.2.1 Tracker Meccanico

I tracker meccanici misurano la posizione e l'orientamento utilizzando una connessione meccanica diretta tra un punto di riferimento e l'oggetto. L'approccio tipico comporta l'uso di una serie articolata di due o più pezzi meccanici rigidi interconnessi con trasduttori elettromeccanici come potenziometri o shaft-encoder. Come l'oggetto si muove , la serie articolata cambia forma e i trasduttori di conseguenza si muovono. Conoscendo a priori i pezzi meccanici rigidi e le misurazioni real time dei trasduttori è possibile calcolare la posizione dell'oggetto rispetto al punto di riferimento.

Il ritardo introdotto dal tempo impiegato per il calcolo della misura (*lag*), è molto breve (meno di 5 msec), la frequenza di aggiornamento è abbastanza alta (300 aggiornamenti al secondo), e l'accuratezza è buona. Il principale

svantaggio consiste nel fatto che il moto dell'utente risulta vincolato dal braccio meccanico [5] [38] [34].

Figura 1.13: BOOM 3C della Fakespace [10]

Esempi commerciali di tracker meccanici sono il *Boom 3C* della *Fakespace* (Figura 1.13) e il *FaroArm* della *Faro Technologies*.

1.3.2.2 Tracker Elettromagnetico

Il tracking elettromagnetico si basa sulla misura del campo elettromagnetico generato facendo scorrere della corrente elettrica all'interno di tre cavi disposti a spirale, orientati ortogonalmente secondo gli assi x,y,z. Queste tre spirali sono inserite dentro un piccolo contenitore che viene fissato all'oggetto di cui si vuole fare il tracking (tipicamente l'utente). Inviando in sequenza corrente elettrica in ognuna delle spirali e misurando il campo magnetico generato da ognuna di esse , è possibile determinare la posizione e l'orientamento dell'oggetto.

Questi meccanismi possono riscontrare interferenze se nell'ambiente circostante sono presenti dispositivi elettronici (infatti generano campo elettromagnetico) o metalli conduttivi (i quali interrompono il campo magnetico).

Le più note aziende produttrici di tracker elettromagnetici sono la *Polhemus* e la *Ascension*. Un esempio è il *FASTRAK* della *Polhemus* (Figura 1.14) che è in grado di fornire latenze estremamente basse (4ms) ed ha la capacità di monitorare diversi oggetti contemporaneamente [5] [38].

Figura 1.14: FASTRAK della Polhemus [11]

1.3.2.3 Tracker Acustico

I tracker acustici utilizzano onde sonore ultrasoniche (cioè ad alta frequenza) per misurare la posizione e l'orientamento degli oggetti. Ci sono due modi possibili per effettuare questo tipo di tracking: il *phase-coherenct* tracking ed il *time-of-flight* tracking.

Il phase-coherence tracking consiste nel misurare la differenza di fase tra le onde sonore emesse da un trasmettitore posto sull'oggetto e quelle emesse da un trasmettitore collocato in una posizione di riferimento. La fase di un suono rappresenta la sua posizione all'interno dell'onda sonora, ed è misurata in gradi: 360 gradi è equivalente alla differenza di una lunghezza d'onda. Questo risulta chiaro se pensiamo all'onda sonora come ad un'onda sinusoidale pura. Il grafico del seno ritorna al punto di partenza dopo 360 gradi (un ciclo, o una lunghezza d'onda). Finché la distanza percorsa dall'oggetto tra una misura e la successiva è minore di una lunghezza d'onda, il sistema è in grado di aggiornare la sua posizione. Utilizzando trasmettitori multipli è possibile misurare l'orientamento.

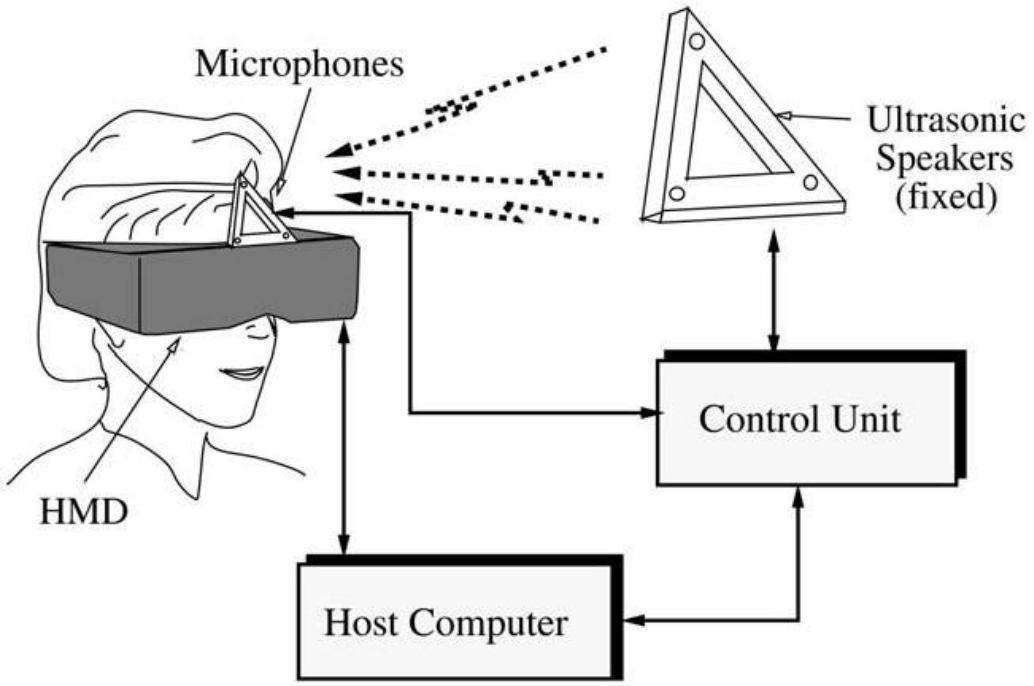


Figura 1.15: Funzionamento Tracker Acustico Logitech

Poiché questi sistemi utilizzano periodici aggiornamenti della posizione, invece di misurare la posizione assoluta ad ogni passo, i tracker phase-coherence sono soggetti all'accumulo progressivo dell'errore.

Oggi tutti i tracker acustici utilizzano il flight-of-time tracking. Questo si basa sulla misura del tempo che un suono emesso da un trasmettitore impiega a raggiungere i sensori, collocati in posizioni prefissate. I trasmettitori vengono attivati ad istanti prefissati e in ogni istante può essere attivo soltanto uno di essi. Misurando l'istante in cui i suoni raggiungono i vari sensori, il sistema riesce a calcolare il tempo impiegato dal suono per percorrere la distanza trasmettitore-sensore, e da qui la distanza percorsa. Poiché, nel volume delimitato dai sensori, può esserci un solo punto che soddisfa le equazioni per ognuna delle distanze, la posizione dell'oggetto può essere determinata univocamente.

Con l'emissione di onde sonore si può avere il problema del *multipath*, cioè l'onda ricevuta da un sensore è spesso la somma di un onda diretta e di una o più onde riflesse. Poiché le pareti e gli oggetti in una stanza sono riflettenti, le

onde riflesse differiscono dall'onda originale in ampiezza e fase, causando così una stima non corretta della posizione dell'oggetto. Questo problema viene risolto semplicemente utilizzando solo la prima onda che arriva al sensore: in questo caso siamo sicuri che la prima onda è quella che ha percorso un cammino diretto sorgente-trasmettitore in quanto le onde riflesse impiegano un maggior tempo a raggiungere il sensore.

I tracker time-of-flight sono tipicamente caratterizzati da frequenze di aggiornamento basse, causate dalla bassa velocità del suono. Un altro problema consiste nel fatto che tale velocità è influenzata da fattori ambientali quali temperatura, pressione e umidità. I vantaggi di questi sistemi consistono nelle dimensioni ridotte, noi costi contenuti costosi e possono coprire lunghe distanze [5] [34].

1.3.2.4 Tracker Inerziale

I tracker inerziali (o INS: Inertial Navigation System) sono composti da accelerometri e giroscopi. A questi tracker deve essere fornita inizialmente la posizione e la velocità (tipicamene mediante l'uso di altri sensori: un ricevitore GPS, un utente ecc..), dopodiché è in grado di calcolare la posizione e la velocità integrando le informazioni provenienti dal sistema inerziale.

Più precisamente, i tracker inerziali calcolano la variazione della velocità e dell'orientamento del corpo misurando l'accelerazione lineare e angolare applicata al corpo.

I giroscopi calcolano la velocità angolare del corpo nel sistema di riferimento inerziale: conoscendo l'orientamento iniziale e integrando la velocità angolare del corpo, sono in grado di fornire l'orientamento corrente. L'accelerometro invece calcola l'accelerazione del corpo nel sistema di riferimento inerziale. Facendo una doppia integrazione è possibile ottenere la posizione corrente. I moderni accelerometri e giroscopi sono implementati utilizzando *Micro Electro-Mechanical Systems* (MEMS) che possono essere inclusi in un circuito integrato (IC).

I tracker inerziali sono utili per calcolare variazioni rapide di posizione in quanto forniscono dati ad una frequenza elevata (circa 1KHz), sono di

ridotte dimensioni e non soggetti ad interferenze. Tuttavia l'operazione di integrazione e la calibrazione sbagliata del dispositivo portano a problemi di deriva (*drift*), ovvero ad accumulo dell'errore [5] [34].

I più noti sistemi di questo tipo sono l'*InertiaCube3* (Figura 1.16) e il *VisTracker* sviluppati dalla *InterSense*.

Figura 1.16: InertiaCube3 della Intersense [17]

1.3.2.5 Tracker Ottici

La maggior parte dei dispositivi di Tracking ottici [34] vengono utilizzati per il tracking della posizione e orientamento della testa. Questi sistemi sono composti da due componenti: sorgenti di luce e sensori ottici atti a riconoscere la posizione degli oggetti o di alcuni punti di essi e da questi ricavarne posizione ed orientamento.

Possono essere suddivisi in due categorie:

- passivi: ricavano informazione dalla luce dell'ambiente.
- attivi: effettuano uno scan per mezzo di laser o infrarossi.

Possono essere impiegati diversi tipi di strumenti ottici:

- Videocamere (es. CCD): sistemi che proiettano l'immagine tridimensionale sul proprio piano focale. Questi sistemi possono essere passivi o attivi. Esempi di sistemi passivi sono le videocamere stereo che registrano le immagini dell'ambiente e le elaborano per ricavarne informazioni 3D, oppure quelli basati sul rilevamento di marker (ad esempio marker colorati o bitonali, di differenti forme, come circolari o quadrati) posti sull'oggetto di cui si vuole effettuare il tracking.

  Sistemi attivi sono quelli basati sul rilevamento di marker luminosi (composti da LED, *Light Emitting Diode*) o quelli basati su videocamere ad infrarosso. Quest'ultimi sono composti da una videocamera, una corona di LED, filtri davanti al CCD che impediscono il passaggio di lunghezze d'onda più corte del rosso e infine marker in materiale altamente retroriflettente. Con un numero di marker pari o maggiore a quattro si riesce a ottenere la posizione più l'orientamento dell'oggetto.

  Vantaggi di questi sistemi: sono poco intrusivi, leggeri, di costi contenuti ed offrono la possibilità di effettuare il tracking di più oggetti. Tuttavia sono sensibli a riflessioni ed occlusioni.

- Emettitori laser: sistema di tipo attivo. Similmente ai sistemi acustici, misurano il tempo di volo (*time of flight*), andata e ritorno, di un raggio laser verso un oggetto. Mediante questi dispositivi si ottengono misure di distanza: con tre di queste si ricava la posizione. Se invece l'angolo del raggio è noto, allora per ricavare la posizione è sufficiente una sola distanza.

  I vantaggi di questi sistemi consistono nell'estrema accuratezza nel ricavare la posizione e nel pochissimo ingombro. Uno svantaggio è dato dal fatto che sono dispositivi costosi e soffrono del problema di shadowing.

- Four Quadrants (4Q): componenti piane in grado di generare due segnali che specificano le coordinate del centro del raggio luminoso incidente sulla superficie [29].

In questa trattazione focalizzeremo l'attenzione sui sistemi che fanno uso di videocamere che rilevano marker bitonali (nero/bianco), poiché è lo stesso tipo di sensore utilizzato nella presente tesi.

1.3.2.6 Tracker Ibridi

Sistemi di tracking commerciale utilizzano molto spesso tecniche ibride (miste) con lo scopo di sfruttare i vantaggi di ogni approccio. La scelta che di solito viene effettuata è quella di affiancare a sistemi con buone prestazioni, ma che presentano particolari svantaggi, altri sistemi che riescono a rimediare ad essi. Ad esempio è utile affiancare a tracker inerziali che, come abbiamo visto precedentemente, presentano il problema dell'accumulo dell'errore, un tracker ottico che, anche con frequenza molto bassa, fornisca una misura assoluta.

Accoppiamenti di questo tipo risultano particolarmente utili, pertanto sono ampiamente utilizzati nei sistemi commerciali.

Capitolo 2

Marker Tracking

Questo capitolo tratta del tracking ottico basato sui markers, in quanto questo è il metodo utilizzato nel presente lavoro di tesi.

I markers sono immagini bidimensionali, generalmente di forma quadrata o circolare, di dimensione nota, con all'interno un'immagine che li identifica univocamente (Figura 2.1 nella pagina seguente) I primi markers trackers sviluppati erano basati su markers di forma circolare o quadrata, tuttavia non era possibile effettuare una distinzione tra un marker ed un altro.

Il primo marker tracker sviluppato specificatamente per applicazioni di realtà aumentata è stato il *Matrix Code* sviluppato da Jun Rekimoto per la Sony Computer nel 1998 [28], nel quale viene utilizzato un marker piano quadrato per il calcolo della posizione dell'utente ed un *2D barcode* incorporato nel marker in modo da distinguerli tra di loro.

Ulteriori esempi di marker tracker sono il *Binary Square Marker*, il *Reactivision*, l'*Intersense*, il *Canon Marker* e *SCR markers*. Tutti questi tuttavia sono prototipi di ricerca oppure sono proprietari: in entrambi i casi non sono

disponibili al pubblico.

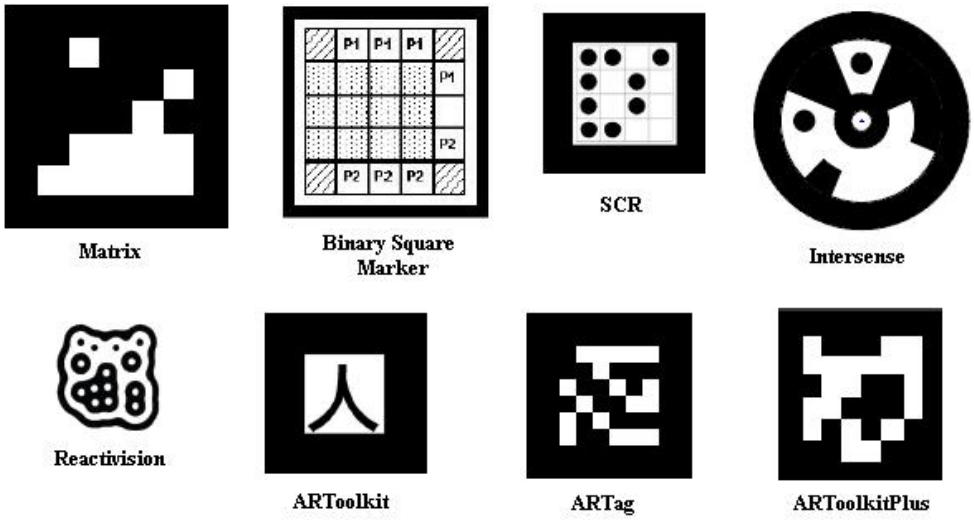


Figura 2.1: Esempi di markers

ARToolkit [15], sviluppato da Hirokazu Kato presso la *Human Interface Technology Laboratory* (HITLab) alla *University of Washington*, è stato il primo marker tracker open source, rilasciato alla comunità nel 1998. Successivamente sono stati sviluppati e rilasciati sempre in maniera opensource, *ARToolkitPlus* [33], creato da Daniel Wagner presso la *Technical University Of Vienna* ed *ARTag* [12], creato da Mark Fiala presso il *National Research Council Canada* che sono le librerie analizzate in questo capitolo.

Si basano sull'utilizzo di markers bitonali (bianco/nero) di forma quadrata e, per ricavare la posizione dell'osservatore rispetto al marker, sfruttano la proprietà per cui da quattro punti complanari è possibile ricavare univocamente la posizione.

2.1 Funzionamento Marker Tracking

Per capire quali sono i miglior markers da utilizzare in una applicazione AR, innanzitutto è necessario analizzare le varie tipologie di markers, cercando di individuare, in base alle differenti caratteristiche, quelli che si prestano in miglior modo ad essere impiegati in tali applicazioni.

Nella realtà aumentata i markers vengono utilizzati per il tracking di oggetti in un ambiente fisico: possono essere piazzati in una posizione fissa dell'ambiente in modo da ricavare la posizione della videocamera nell'ambiente stesso, oppure piazzati su persone od oggetti in movimento per ricavare la posizione della videocamera relativamente ad essi.

Attraverso varie prove è stato visto che i migliori markers da utilizzare sono quelli di forma quadrata in quanto i segmenti che identificano il quadrato permettono di calcolare con elevata precisione il punto in cui si intersecano le linee (mediante tecniche di analisi dell'immagine sub-pixels: vengono cioè ricavate informazioni su dove si intersecano le linee non a livello di pixel ma all'interno dello steso pixel) e quindi di ricavare i quattro punti complanari necessari al calcolo della posizione della videocamera rispetto al marker.

Un'altra scelta possibile sulla tipologia di markers da utilizzare, è data dall'utilizzo di markers colorati o markers bianchi/neri. Un vantaggio nell'utilizzo dei markers colorati è che aumenta il set di marker da poter utilizzare rispetto al caso dei markers bitonali. Tuttavia, con l'uso dei markers colorati, l'immagine catturata dalla videocamera deve essere elaborata a colori e non in bianco/nero e, se l'immagine è in *full resolution*, la memoria utilizzata aumenta, aumentando così anche il tempo necessario per analizzare l'immagine (di un fattore tre o quattro). C'è inoltre il problema che, a seconda dell'intensità o del tipo di luce presente nell'ambiente o a seconda del materiale utilizzato per i markers, i colori che vengono catturati dalla videocamera possono non essere esattamente quelli utilizzati, non permettendo quindi all'algoritmo che analizza l'immagine di individuare i markers.

Un sistema di marker tracking può essere valutato in base a quattro proprietà principali: usabilità, efficienza, accuratezza, affidabilità [37]. Queste quattro proprietà descrivono la performance, i vantaggi e gli svantaggi di un marker tracking system.

L'usabilità di un sistema descrive la facilità con cui l'utente può integrare il sistema di tracking nella sua applicazione di realtà aumentata. Considera anche la compatibilità del sistema con diverse piattaforme e sistemi operativi. L'usabilità può essere comparata solo qualitativamente.

L'efficienza di un sistema di marker tracking può essere valutata calcolan-

do la sua performance dal punto di vista del tempo impiegato per il tracking; viene valutato cioè il tempo necessario per il marker detection e decodifica.

L'accuratezza è definita come l'errore nell'estrazione dei bordi del marker dall'immagine 2D. Non viene considerato l'errore nel calcolo della posizione in quanto dipende anche dai parametri intrinseci della videocamera e dall'algoritmo di calibrazione.

L'affidabilità descrive se il sistema è in grado di riconoscere ed effettuare il tracking di un marker anche in condizioni di funzionamento non ideali, come, ad esempio, in caso di videocamere non a fuoco o con una elevata distorsione.

Nel seguito, si analizza il funzionamento del marker tracking nel caso in cui vengano utilizzati marker quadrati bitonali, che è il caso più utilizzato nelle applicazioni di realtà aumentata (Figura 2.2) [2]:

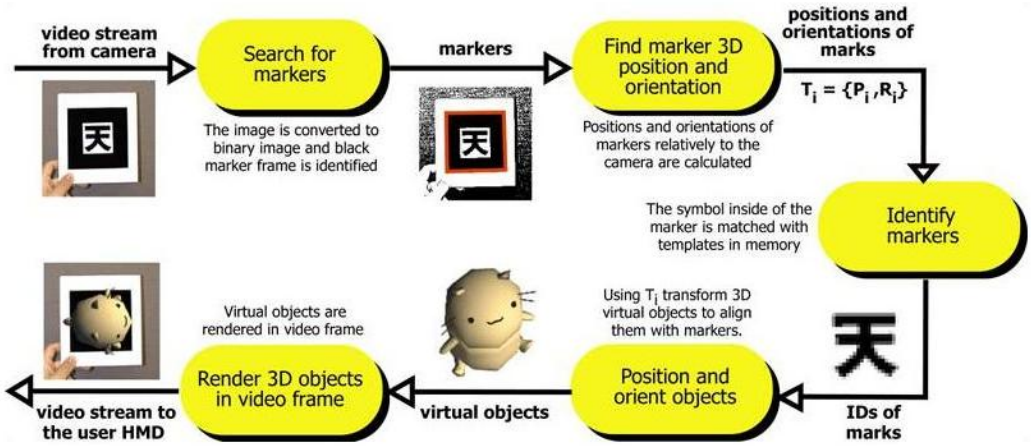


Figura 2.2: Funzionamento Marker Tracking [2]

1. Attraverso una videocamera, viene catturato il video del mondo reale ed inviato al computer.

2. Ogni singolo frame del flusso video, se a colori, viene dapprima convertito in un'immagine in scala di grigi (se l'immagine catturata dalla videocamera è già in scala di grigi ovviamente la conversione non viene fatta,

risparmiando così tempo) e quindi viene effettuata una operazione di *thresholding* che può essere statica o dinamica (Figura 2.3).

Nel caso di thresholding statica la soglia (*threshold*) è fissa mentre nel caso di thresholding dinamica il valore della soglia non è fisso e scelto dall'utente (con un valore compreso tra 0 e 255) ma viene calcolato frame per frame analizzando i valori dei livelli di grigi presenti nell'immagine. Comunque, in entrambi i casi, i valori dei pixels che stanno sotto una certa soglia (*threshold*) vengono convertiti in bianco, gli altri in nero. Un esempio di thresholding binaria può essere $dst_i = (src_i > T)?M : 0$ dove dst_i è l'i-esimo pixel nell'immagine destinataria, src_i è l'i-esimo pixel nell'immagine sorgente in scala di grigi, T è il valore della soglia da noi definito ed M il valore massimo da assegnare ai pixel il cui valore è maggiore della soglia.

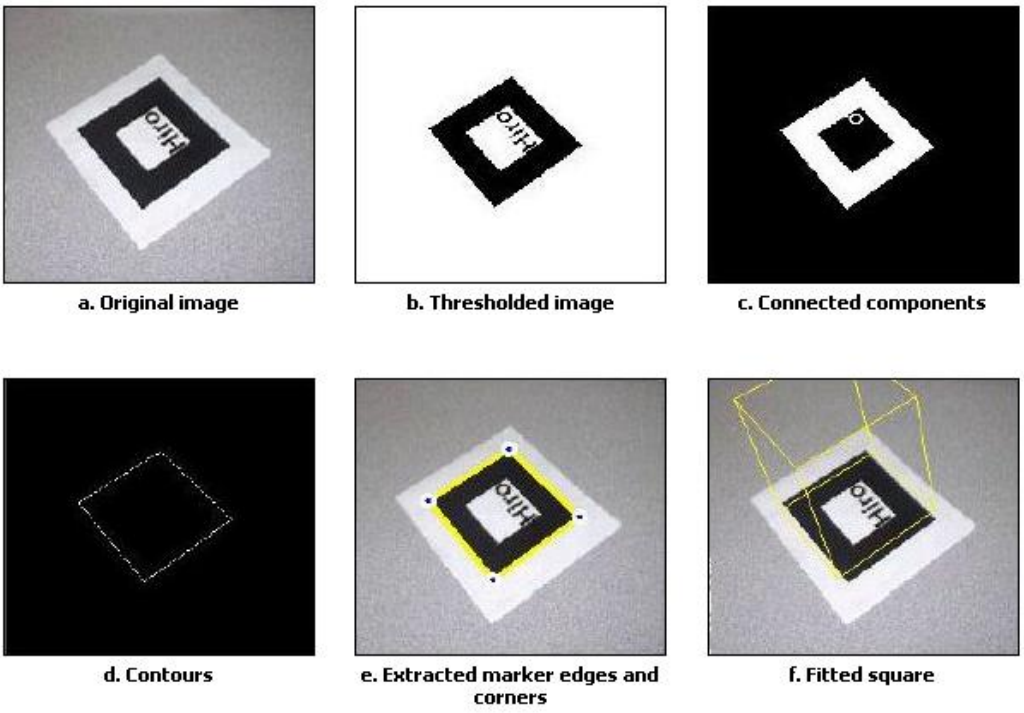


Figura 2.3: Fasi di analizzamento dell'immagine [2]

3. L'immagine binaria viene elaborata per trovare tutti i contorni. Vengono selezionati quei contorni che formano un quadrato e individua-

ti i quattro vertici (e quindi quattro punti) necessari per ricavare la posizione (Figura 2.3 nella pagina precedente).

4. Dopo aver individuato uno o più marker all'interno dell'immagine, viene calcolata la posizione e l'orientamento della videocamera relativamente al marker mediante *image analysis*[1]. L'orientamento e la posizione vengono codificati in una matrice di trasformazione 4X4 (ci si può riferire a questa anche come *camera extrinsic parameter*).

   Avendo ora diversi sistemi di riferimento come in (Figura 2.4) si deve passare dalle *marker coordinates* alla *camera coordinates* [15].

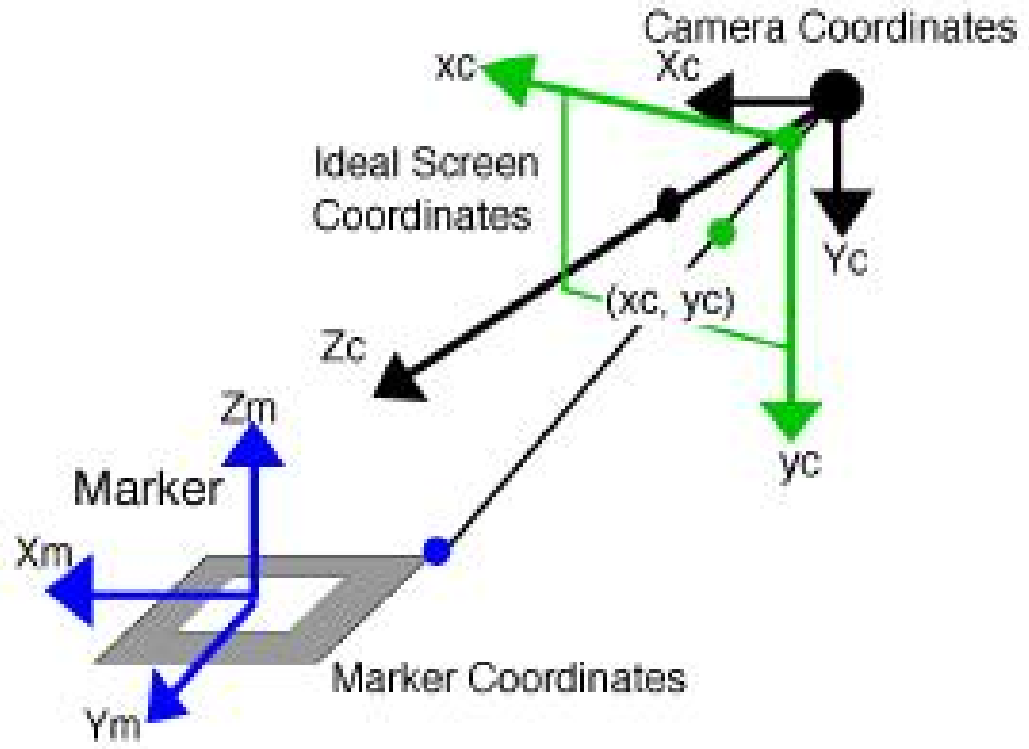


Figura 2.4: Relazione tra i vari sistemi di riferimento [15]

La matrice di trasformazione (definita anche *camera extrinsic parame-*

[1]Vedere Appendice B

ters) è la seguente:

$$\begin{bmatrix} X_c \\ Y_c \\ Zc \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3x3} & & T_{3x1} & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = T_{cm} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

dove R_{3x3} rappresenta la rotazione della videocamera rispetto al marker mentre T_{3x1} rappresenta la traslazione.

E' necessario calcolare anche la matrice di proiezione prospettica per passare dalle *camera coordinates* alle *camera screen coordinates* in cui passiamo da un sistema a tre dimensioni in uno a 2 dimensioni, rappresentato dall'*image plane* della videocamera. Il modello utilizzato è quello della *pinhole camera* (Figura 2.5 nella pagina successiva) mediante il quale possiamo notare che un punto nel mondo a tre dimensioni viene proiettato nel piano immagine a due dimensioni mediante la seguente relazione:

$$p_x = F\frac{P_x}{P_z} \quad p_y = F\frac{P_y}{P_z}$$

La matrice di proiezione prospettica viene ottenuta attraverso un processo di calibrazione della videocamera [19]:

$$P = \begin{bmatrix} s_x f & 0 & x_0 & 0 \\ 0 & s_y f & y_0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dove abbiamo i seguenti parametri intrinseci (*intrinsic parameters*):

- f rappresenta la lunghezza focale, cioè la distanza tra il centro di proiezione prospettica e l'*image plane*
- s_x e s_y rappresentano un fattore di scala [pixel/mm] nella direzione x e y degli assi. Siccome l'image plane è costituito da pixel, è

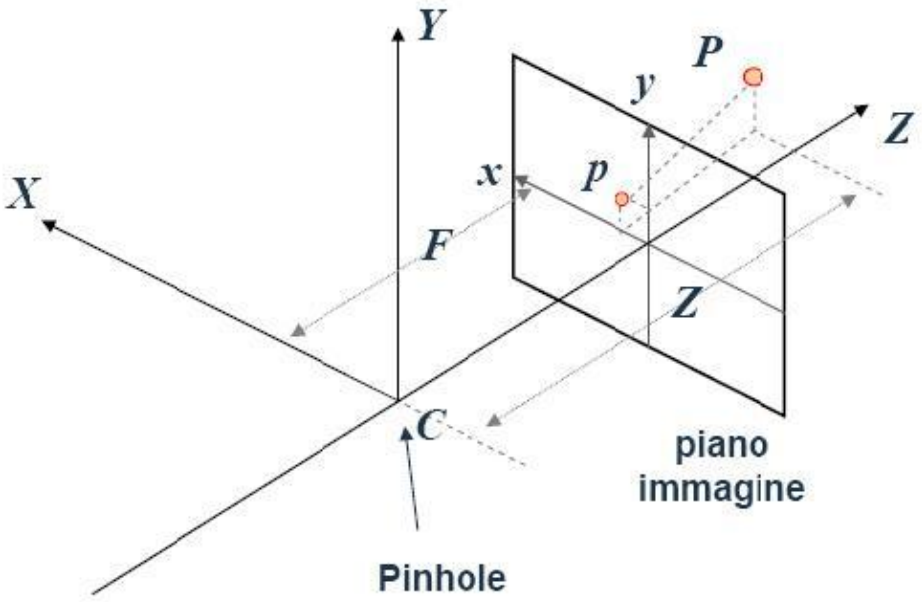


Figura 2.5: Modello Pinhole Camera

necessario questo parametro di conversione.

- x_0 e y_0 vengono definiti *principals points* e rappresentano le coordinate, nelle *camera screen coordinates*, dove l'asse ottico (retta perpendicolare al centro del piano immagine) passante per il centro di proiezione interseca il piano immagine.

I sistemi classici di calibrazione utilizzano un pattern di dimensione conosciuta, all'interno del *field of view* della videocamera: spesso sono una griglia con immagini regolari stampate (Figura 2.6 nella pagina seguente), come, ad esempio, dei cerchi neri le cui dimensioni e posizioni all'interno della griglia sono note, per permettere il calcolo della corrispondenza tra i punti 3D ed i punti 2D dell'immagine e quindi per calcolare i parametri della matrice di proiezione.

La libreria ARToolkit, ad esempio, crea la matrice di proiezione prospettica con un proprio programma di calibrazione e la matrice ottenu-

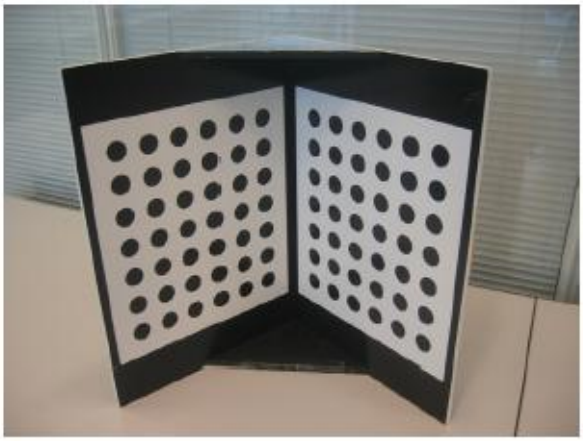


Figura 2.6: Griglia 3D di calibrazione per la stima della matrice di proiezione

ta, viene direttamente caricata in OpenGL[2] senza doversi preoccupare di dover ricavare da questa i parametri necessari alla funzione OpenGL *glFrustum()*[3] (glFrustum() ha vari parametri necessari per settare la matrice di proiezione prospettica).

5. Dopo aver ricavato la posizione viene identificato il marker attraverso l'immagine o l'ID presente nel marker stesso, in modo da poter distinguere un marker da un altro.

6. Utilizzando la matrice di trasformazione, la matrice di proiezione e l'ID del marker, viene sovrapposto un particolare modello 3D sul marker.

2.2 ARToolkit

ARToolkit [15] è una libreria utilizzata per sviluppare applicazioni di realtà aumentata ed utilizza algoritmi di computer vision per ricavare la posizione e l'orientamento della videocamera rispetto ad un marker posto nell'ambiente (Figura 2.7 nella pagina successiva). E' stata sviluppata dal Dr. Hirokazu Kato e M. Billinghurst presso lo *Human Interface Technology*

[2]vedere Appendice La grafica tridimensionale e OpenGL
[3]vedere Appendice La grafica tridimensionale e OpenGL

Laboratory (HITLab) della *University of Washington* e la prima versione è stata rilasciata nel 1999. Le caratteristiche di questa libreria sono:

Figura 2.7: Esempio di una applicazione sviluppata con ARToolkit [24]

- E' una libreria multipiattaforma che attualmente può essere utilizzata sui seguenti sistemi operativi: Windows (95/98/NT/2000/XP), SGI IRIX, PC Linux, Mac OS X.
- Fornisce delle semplici API in C.
- Sovrappone oggetti 3D su marker reali (mediante l'utilizzo di algoritmi di computer vision).
- Permette di calibrare semplicemente la videocamera.
- Offre una semplice libreria grafica basata sulle GLUT.
- Supporta il 3D VRML[4].
- Viene fornito con esempi e documentazione.
- Supporta più sorgenti di acquisizione video (USB, Firewire, capture card) e più formati (RGB/YUV420P, YUV).
- Può essere utilizato per sviluppare applicazioni video o optical see-through.

[4]VRML: Virtual Reality Modeling Language

- Utilizza *pattern marker* cioè markers con all'interno un'immagine per distinguerli tra di loro.
- OpenSource con licensa GPL per usi non commerciali.

Il processo di sviluppo di una applicazione di realtà aumentata mediante la libreria ARToolkit avviene in tre fasi:

1. Inizializzazione
    - Viene inizializzato il sistema di cattura del flusso video e vengono letti sia i parametri della videocamera derivati dalla calibrazione che il file che definisce il marker utilizzato.
2. Ciclo Principale
    - Viene catturato un frame video.
    - Nel frame video corrente vengono individuati i markers e riconosciute le immagini presenti nel marker.
    - Viene calcolata la matrice di trasformazione relativamente al marker individuato.
    - Viene disegnato l'oggetto virtuale sul marker
3. Chiusura
    - Viene chiuso il sistema di cattura del flusso video.

I passi presenti nella fase del ciclo principale vengono ripetuti continuamente mentre la fase di inizializzazione e chiusura venono eseguiti solamente all'inizio e alla fine del programma.

2.2.1 ARToolkit single marker tracking

Vediamo come viene sviluppato un programma che utilizza un solo marker mediante ARToolkit analizzando il codice sorgente del programma di esempio fornito con la libreria (`SimpleTest`) [2]: le funzioni principali sono `main`, `init`, `mainLoop`, `draw` e `cleanup`.

- funzione `main`: questa è la routune principale e provvede ad invocare le funzioni `glutInit`, `init` e `argMainLoop`. La funzione `glutInit` inizializza la libreria GLUT, necessaria per aprire e gestire la finestra grafica in un contesto OpenGL, mentre la funzione `init` contiene il codice per aprire il flusso video e per leggere i parametri di configurazione della videocamera ed il marker utilizzato per calcolare la pose estimation. La funzione `argMainLoop` contiene tutte quelle funzioni per svolgere quanto descritto nel Ciclo Principale.

- funzione `init`: inizialmente viene riconosciuta la videocamera ed individuata la dimensione dell'immagine video ed il refresh rate mediante l'utilizzo della libreria DSVideoLib[5]. Dopo è necessario inizializzare i parametri di ARToolkit che sono le caratteristiche della videocamera e il pattern che verrà usato per effettuare il *pattern template matching*. Mediante la funzione `arParamLoad` vengono settati i parametri della videocamera ottenuti attraverso la calibrazione della videocamera stessa e salvati in uno specifico file mentre, attraverso `arLoadPatt`, viene scelto quale pattern utilizzare. Questa funzione restituisce un identificatore (di tipo intero) che identifica un particolare pattern. Poiché la calibrazione della videocamera può essere stata effetuata ad una risoluzione differente rispetto a quella utilizzata in condizioni operative, con la funzione `arParamChangeSize` vengono cambiati i parametri intrinseci della videocamera (ovvero la matrice di proiezione) utilizzando la nuova risoluzione. Per ultimo viene aperta la finestra grafica con `argInit` che provvede anche a settare la matrice di proiezione prospettica.

- funzione `mainloop`: questa funzione esegue tutte le operazioni contenute nel Ciclo Principale analizzato precedentemente. Inizialmente viene catturato un frame video, memorizzato in un buffer e quindi visualizzato su schermo. Viene successivamente chiamata la funzione `arDetectMarker` con la quale viene analizzato il frame video per individuare i markers quadrati di ARToolkit: essa effettua un *thresholding statico*, *contour extraction* (vengono individuati i contorni che

[5]http://sourceforge.net/projects/dsvideolib/

costituiscono il quadrato) e line corner estimation. Il numero di markers individuati nel frame video viene memorizzato in una variabile e in una lista vengono memorizzate informazioni sui markers: ogni elemento della lista è una struttura di tipo `ARMarkerInfo` che contiene un valore di confidenza (probabilità che un marker individuato sia esattamente un marker), un *id number* che identifica univocamente un marker e il centro del marker nelle screen coordinates. Dopo che l'immagine è stata completamente analizzata, viene catturato un nuovo frame mediante la funzione `arVideoCapNext` mentre vengono eseguite altre operazioni, come quella di confrontare tutti i valori di confidenza dei marker rilevati per associare il marker id number corretto a quello con il valore di confidenza più alto. La matrice di trasformazione, ovvero la posizione e l'orientamento del marker di cui deve essere effettuato il tracking, viene calcolata mediante la funzione `arGetTransMat`. In particolare questa funzione restituisce la posizione e l'orientamento del marker nel sistema di riferimento della videocamera (*camera coordinate system*). Ogni oggetto virtuale verrà disegnato a partire dal centro del marker, infatti ogni vertice di una figura virtuale viene moltiplicato per la matrice di trasformazione che, come detto, restituisce la posizione del marker dalla videocamera. La funzione `draw` viene utilizzata per impostare la matrice di trasformazione e per effettuare il rendering degli oggetti virtuali: più specificatamente, la matrice di trasformazone calcolata (che è una matrice 3x4) viene convertita in una matrice di trasformazione compatibile con OpenGl di dimensione 4x4. Questi sedici valori rappresentano la posizione e l'orientamento della videocamera reale, così, utilizzandoli per posizionare la videocamer virtuale all'interno dell'ambiente OpenGL, ogni oggetto tridimensionale disegnato apparirà relativamente al marker fisico.

- funzione `cleanup`: questa funzione viene utilizzata per fermare il video processing e chiudere il driver video in modo che la videocamera possa essere utilizzata da altre applicazioni.

Per una trattazione più completa rimandiamo alla documentazione ufficiale di ARToolkit[6].

2.2.2 ARToolkit multi-marker tracking

Spesso un solo marker non è sufficiente per applicazioni di realtà aumentata. Questo è il caso di quando, per esempio, deve essere effettuato il tracking di una persona all'interno di un'area molto vasta come una stanza, oppure di oggetti molto grandi. E' chiaro, quindi, che, a seconda degli spostamenti della videocamera nell'area di lavoro, il marker non è più visibile. Si deve perciò ricorrere ad un *marker set* (o *marker fields*) (Figura 2.8). Il marker set contiene un insieme di markers, definiti rispetto ad un'origine predefinita (*global origin*) che può essere, ad esempio, il centro di un marker, il vertice di un marker, oppure una qualsiasi posizione all'interno del marker set.

In questo modo può essere visualizzato l'oggetto virtuale anche se uno o più marker non sono stati individuati dall'algoritmo o non sono visibili dalla videocamera.

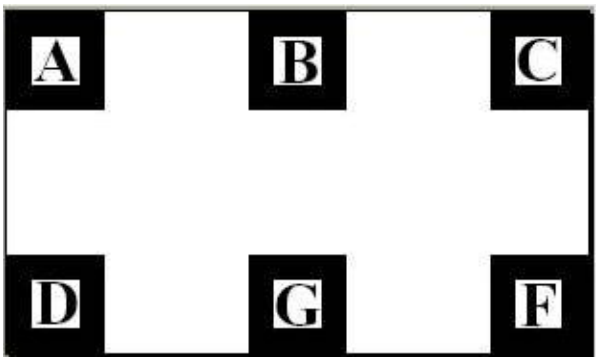


Figura 2.8: Esempio di marker set con *ARToolkit patterns*

Vediamo le funzioni più importanti che differiscono dal caso di tracking di un solo marker. Mediante la funzione `arMultiReadConfigFile` settiamo la struttura `ARMultiMarkerInfoT` con i valori presente nel file di configurazione, in questo caso `config_name`:

[6] www.hitl.washington.edu/artoolkit/documentation/

```
if((config = arMultiReadConfigFile(config_name)) == NULL ) {
        printf("config data load error !!\n");
        exit(0);
    }
```

`config_name` contiene la lista di tutti i patterns che definiscono il marker set, la loro dimensione (in mm) e la loro posizione rispetto all'origine globale (Figura 2.9).

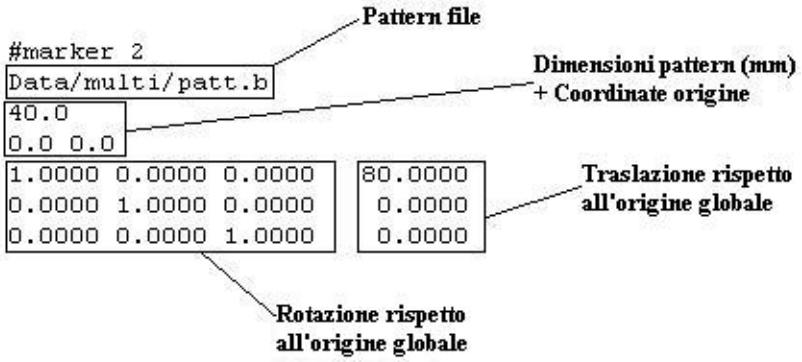


Figura 2.9: File di configurazione

Per individuare i markers presenti nel frame video viene utilizzata la stessa funzione vista precedentemente nel caso di tracking di un solo marker, ma cambia la funzione per il calcolo della matrice di trasformazione. La funzione `arMultiGetTransMat` non restituisce la posizione di ciascun marker nella *camera coordinates*, ma la posizione dell'origine globale nella *camera coordinates*.

2.3 ARToolkitPlus

ARToolkitPlus [33] [3] è una libreria open source per il marker tracking in real time sviluppata presso la *Graz University Of Technology*, rilasciata sotto licenza GPL ed ottimizzata per essere eseguita su dispositivi mobili, come smartphone. E' denominato *Plus* perchè offre delle funzionalità aggiuntive rispetto al codice delle prime versioni di questa libreria, basato su quello

di ARToolkit. La maggior parte del codice delle ultime versioni, però, non condivide quasi più niente con quello di ARToolkit, anche se molte funzioni mantengono lo stesso nome.

Le caratteristiche principali della libreria ARToolkitPlus sono:

- Scritta interamente in C++. E' presente una classe principale per il tracking, chiamata `class Tracker`, più due classi derivate: la classe `TrackerSingleMarker!`fornisce un'interfaccia ad alto livello specifica per il tracking di un solo marker mentre la classe `TrackerMultiMarker` fornisce un'interfaccia ad alto livello specifica per il tracking multi-marker (Figura 2.10).

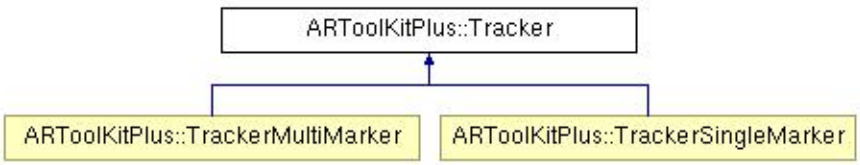


Figura 2.10: Relazioni tra le classi che costituiscono ARToolkitPlus

- Configurazione a basso livello mediante l'utilizzo di C++ Templates.

- Supporto per gli *ID-encoded markers*, mediante i quali viene velocizzato il processo di marker detection poiché non è più necessario effettuare un'*image matching* come nel caso di ARToolkit. Attualmente supporta 4096 marker con la possibilità di variare la dimensione del bordo: in questo modo è possibile utilizzare un'area maggiore per il pattern (in questo caso l'ID) all'interno del marker. Utilizzando i *simple id-encoded marker* viene codificato un numero a 9 bits (quindi abbiamo 512 marker possibili) all'interno di un pattern 6x6: questi 9 bits vengono ripetuti 4 volte per riempire i 36 bits. Per utilizzare questi marker la dimensione del marker deve essere settata a 6x6, 12x12 oppure 18x18: nel primo caso ogni pixel del pattern viene direttamente estratto. Con i *BCH id-encoded markers* possono invece essere utilizzati 4096 markers in quanto l'ID del marker è codificato in 12 bits.

- Larghezza del bordo variabile: la larghezza del bordo può essere cambiata per poter utilizzare un'area più grande del marker per il pattern (Figura 2.11). In questo modo possono essere riconosciuti anche marker di dimensioni ridotte.

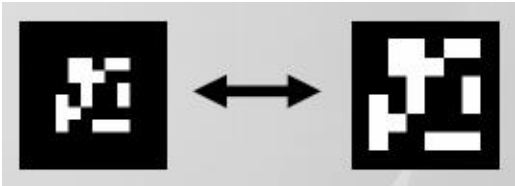


Figura 2.11: Dimensioni del bordo variabile

- Utilizza l'algoritmo RPP (*Robust Pose Estimation*) per il calcolo della posizione e dell'orientamento della videocamera rispetto ai markers. Confrontato con l'algoritmo di pose estimation utilizzato da ARToolkit, lo RPP offre un risultato più accurato.

- Può essere utilizzata una thresholding dinamica.

- Supporto per immagini 16-bit RGB565 (formato presente nelle videocamere dei PDAs) e per il formato YUV12, dove la componente di luminanza (Y) viene memorizzata alla risoluzione massima (8bit) e le due componenti di crominanza (UV) a risoluzione dimezzata (2 bit ciascuna).

- Vignetting (radial luminance falloff) compensation: alcune videocamere, soprattutto quelle degli smartphone e PDAs, presentano una *radial luminance falloff* che impedisce ai marker presenti al bordo dell'immagine di essere riconosciuti. ARToolkitPlus è in grado di rimediare a questo inconveniente (Figura 2.12 nella pagina seguente).

- Supporta la calibrazione standard di ARToolkit e il più accurato processo di calibrazione GML Matlab Camera Calibration Toolbox[7] per la calibrazione della videocamera.

[7]GML Matlab Camera Calibration Toolbox: http://research.graphicon.ru/calibration/gml-matlab-camera-calibration-toolbox.html

Figura 2.12: Vignetting Compensation [3]

Questa libreria ha anche degli svantaggi rispetto ad ARToolkit: non offre nessuna funzione per l'acquisizione del flusso video e non effettua il rendering di nessuna geometria (specialmente VRML). Non è fornito alcun aiuto per settare il proprio ambiente di sviluppo (IDE) o alcun programma di esempio.

2.3.1 ARToolkitPlus single marker tracking

La classe `TrackerSingleMarker` definisce una semplice interfaccia ad alto livello per effettuare il tracking di un solo marker.

Di seguito sarà svolto un semplice esempio di come effettuare il tracking single marker: per i dettagli sulle funzioni utilizzate rimandiamo alla documentazione sulle funzioni di ARToolkitPlus[8]. E' necessario integrare questa libreria in un ambiente OpenGL (per gestire le finestre grafiche verranno utilizzate le *GLUT*) e scrivere funzioni per l'acquisizione video in quanto, come detto precedentemente, ARToolkitPlus non effettua né il rendering né cattura flussi video.

Inizialmente deve essere invocato il costruttore della classe:

```
ARToolKitPlus::TrackerSingleMarker *tracker =
new ARToolKitPlus::TrackerSingleMarkerImpl<6,6,6,1,8>(width,height);
```

dove i primi due parametri specificano che il pattern deve essere diviso in una griglia 6x6 in cui ogni cella corrisponde ad un bit: 9 bit formano l'ID del marker che viene ripetuto quattro volte. Gli altri tre parametri non vengono presi in considerazione in quanto servono solo se i markers utilizzati sono

[8]disponibili nel pacchetto opensource ARToolkitPlus

quelli di ARToolkit. Le variabili `width` e `heigth` rappresentano le dimensioni del frame video.

Dobbiamo poi procedere a settare vari parametri per il tracker come il tipo di immagine acquisita, la dimensione del marker utilizzato e il valore della threshold da utilizzare.

Per settare il formato dei pixels dell'immagine acquisita deve essere utilizzata la funzione `setPixelFormat`: possono essere definiti vari formati come, per esempio:

- `PIXEL_FORMAT_LUM` se l'immagine acquisita che la libreria deve trattare per individuare il markers è in scala di grigio.
- `PIXEL_FORMAT_RGB` nel caso l'imagine acquisita sia a colori a 24 bit espressa nelle componenti *red, green* e *blue*.
- `PIXEL_FORMAT_RGB565` se l'immagine acquisita è a colori a 16 bit come per esempio accade nelle videocamere dei PDAs.

Per impostare la dimensione del marker, espressa in millimetri, deve essere utilizzata la funzione `setPatternWidth`, mentre con le funzioni `setBorderWidth`, `setMarkerMode` e `setThreshold` vengono impostati rispettivamente le dimensioni del bordo, il tipo di id-marker da utilizzare (se il marker standard previsto da ARToolkit oppure il BCH-id marker) e il valore della threshold.

La threshold impostata con questa funzione è di tipo statica (vedere paragrafo 2.2): nel caso in cui si volesse effettuare una *dynamic threshold* è necessario utilizzare la funzione `activateAutoThreshold`.

Per effettuare il calcolo della *pose estimation* e quindi ricavare la matrice di trasformazione, viene impiegata la funzione `setPoseEstimator`. Questa funzione, come argomento, accetta uno tra questi due valori:

- `POSE_ESTIMATOR_RPP`: per la pose estimation viene utilizzato l'algoritmo *Robust Planar Pose algorithm*.
- `POSE_ESTIMATOR_ORIGINAL`: per la pose estimation viene utilizzato l'algoritmo implementato nella libreria ARToolkit.

A questo punto è possibile inizializzare il tracker con la funzione `init` che accetta come parametri il file che contiene i parametri intrinsesci della videocamera risultati dalla calibrazione della stessa, il *near* e *far clip planes* in modo da settare il frustum (tronco di piramide che costituisce il capo visivo in una proiezione prospettica) (vedere appendice La grafica tridimensionale e OpenGL). Come per ARToolkit, la matrice di proiezione prospettica non viene definita invocando la funzione OpenGL `glFrustum`, ma viene ricavata direttamente dai parametri suddetti (compatibile con il formato OpenGL).

A questo punto le varie inizializzazioni sono state eseguite ed il tracker può cercare il marker presente nel frame immagine e calcolare la matrice di trasformazione: se più marker sono presenti, viene restituita la posizione del marker con il più alto valore di confidenza. La funzione `calc`, che ha come parametro il buffer che contiene il frame immagine, provvede ad individuare i markers in essa presenti e a calcolare la posizione del marker con il più alto valore di confidenza rispetto alla videocamera. Restituisce l'ID del marker individuato.

E' possibile ora caricare la *modelview matrix* e la *projection matrix* in OpenGL così da poter effettuare il rendering di geometrie 3D. Questa operazione viene effettuata definendo, in OpenGL, prima con quale matrice stiamo lavorando e successivamente caricando la matrice stessa: ad esempio, con la funzione `glMatrixMode(GL_MODELVIEW)` si seleziona la matrice di trasformazione e passando questa matrice come argomento alla funzione OpenGL `glLoadMatrixd` si imposta la videocamera virtuale.

2.3.2 ARToolkitPlus multi-marker tracking

Il tracking multimarker in ARToolkitPlus viene implementato attraverso la classe `TrackerMultiMarker`.

Anche in questo caso come nel caso single marker dobbiamo procedere a creare il tracker mediante la chiamata al costruttore:

```
ARToolKitPlus::TrackerMultiMarker *tracker = new
    ARToolKitPlus::TrackerMultiMarkerImpl<6,6,6, 1, 16>(width,height);
```

e a settare il formato immagine, la dimensione del marker, la tipologia di marker utilizzata, il valore della threshold e l'algoritmo di pose estimation. Nel caso in cui i markers non si trovino sullo stesso piano, l'algoritmo RPP non può essere utilizzato ma si deve ricorrere all'algoritmo impiegato da ARToolkit.

Anche in questo caso dobbiamo inizializzare il tracker utilizzando la funzione `init` che, rispetto al caso single marker, ha un parametro in più e precisamente il file di configurazione del marker set, che contiene le informazioni su tutti i marker presenti nel set, tra cui le dimensioni dei singoli markers, il loro ID e la loro posizione relativa alla *global origin*. La struttura di questo file è la medesima di quella utilizzata per ARToolkit.

Mediante la funzione `calc`, viene calcolata la posizione dell'origine globale del marker set rispetto alla videocamera e la posizione relativa di ogni marker rispetto all'origine suddetta. Come valore di ritorno restituisce il numero di marker individuati nel frame video.

A questo punto è possibile settare la matrice di trasformazione OpenGL con la matrice ricavata dalla funzione `calc` in modo da poter disegnare oggetti virtuali relativamente all'origine globale del marker set.

2.4 ARTag

ARTag [12] (Figura 2.13 nella pagina successiva) è una libreria per marker tracking sviluppata da Mark Fiala presso l'istituto NRC (*National Research Council of Canada*), disponibile dal 2004 e rilasciata come SDK (*Software Development Kit*) il cui codice sorgente non è open source. ARTag dispone di 2002 id-marker: anche in questo caso, come per ARToolkitPlus, non viene fatto il template-matching garantendo così maggiore velocità nel marker detection. I markers sono bitonali (bianchi e neri), ognuno dei quali consiste di un bordo quadrato e di una regione interna formata da una griglia 6x6 di celle bianche e nere: 1001 markers hanno un bordo nero su uno sfondo bianco e viceversa per gli altri 1001.

La libreria ARTag inizialmente, dopo che è stato catturato un frame video, individua i contorni mediante un metodo *edge based*: gli edge pixels (i

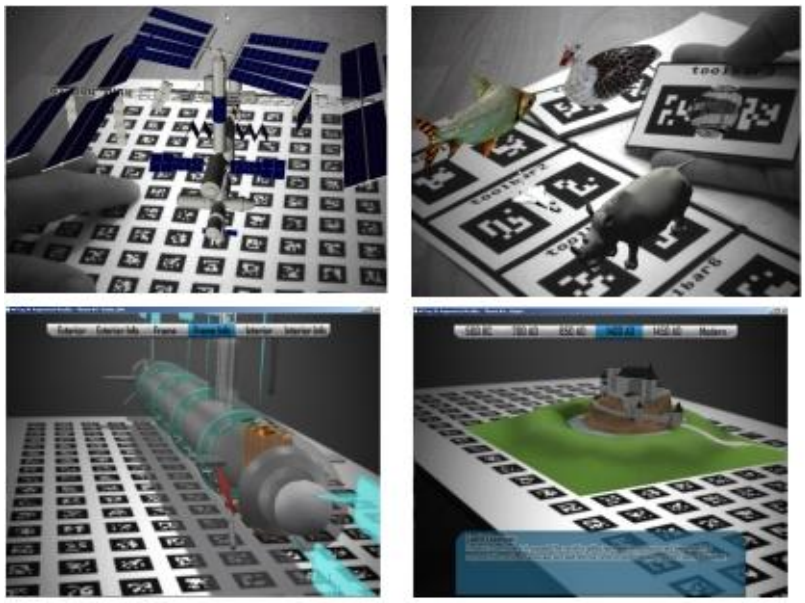


Figura 2.13: Esempio di applicazione sviluppata con ARTag [1]

pixels che costituiscono il contorno di una immagine e nel caso specifico che individuano il contorno del marker) vengono collegati tra di loro in segmenti che poi, se possibile, vengono raggruppati in quadrati. ARTag è in grado di rilevare i markers anche se il bordo è parzialmente ostruito (Figura 2.14) e in condizioni di variazione dell'intensità della luce.

Figura 2.14: ARTag può riconoscere i markers anche se parzialmente ostruiti

Dopo che i contorni del marker sono stati individuati, l'immagine interna viene campionata con una griglia 6x6 e ad ogni cella viene assegnato il valore '1' o '0' a seconda del suo colore. Dei 36 bit così ottenuti, 10 sono utilizzati per ottenere l'ID, mentre gli altri 26 sono bit di ridondanza: rispetto ad ARToolkit vengono ridotti notevolmente i falsi positivi. Il *CRC* ed il *forward*

error correction sono i metodi utilizzati per vedere se i 36 bit fanno parte dell'insieme di marker ARTag e per estrarre l'ID. Rispetto ad ARToolkit, ARTag riduce anche *l'inter-marker confusion*: la probabilità che l'algoritmo di marker detection confonda un marker per un altro è molto bassa.

Anche questa libreria può effettuare il single-marker tracking o il multi-marker tracking: può essere ricavata, cioè, la posizione della videocamera attraverso un solo marker oppure da un marker set. Nell'ultimo caso la posizione è definita rispetto ad una *global origin* del marker set.

E' da notare che l'SDK non offre alcuno strumento per effettuare una calibrazione automatica della videocamera: la scarsa documentazione spiega come poter effettuare una calibrazione manuale della stessa per ricavare così i parametri da poter inserire nella funzione OpenGL `glFrustum` in modo da poter settare la matrice di proiezione prospettica. Non offre nemmeno una funzione per la gestione della finestra grafica: anche in questo caso verrà usata la libreria *GLUT* per aprire una finestra grafica in un contesto OpenGL. In compenso però, al contrario di ARToolkitPlus, offre funzioni per l'acquisizione del flusso video.

L'esecuzione di un'applicazione ARTag avviene in 3 fasi [8]:

1. Inizializzazione

   - Viene chiamata la funzione `init_artag()`.
   - Se vogliamo usare un marker set deve essere invocata la funzione `load_array_file()`.
   - Si deve associare il marker o il single marker ad un *handle* (o identificatore) mediante la funzioni `artag_associate_array()` o `associate_single_marker()`. Il valore dell'identificatore assegnato da queste funzioni sarà poi necessario ad altre funzioni.

2. Ciclo Principale

   - Mediante la funzione `artag_find_objects()` l'algoritmo di marker detection individua i markers presenti nell'immagine.

- Per ogni marker o marker set associato ad un identificatore, deve essere chiamata la funzione `artag_is_object_found(#handle)` che restituisce `true` se quel particolare marker o marker set è presente nell'immagine.
- Se è stato restituito `true` dalla funzione precedente, deve essere chiamata la funzione `artag_set_object_opengl_matrix()` che provvede a settare la *modelview matrix* in OpenGL per poter effettuare il rendering.

3. Chiusura

   - Tutte le risorse allocate dalla libreria ARTag possono essere deallocate mediante la funzione `close_artag()`.

Di seguito verranno svolti dei semplici esempi su come sviluppare un'applicazione ARTag single o multi-marker tracking, analizzando le principali funzioni disponibili[9].

2.4.1 ARTag single-marker tracking

Nella funzione `main()` deve essere inizializzata la libreria per effettuare il tracking e creare la finestra grafica mediante le *GLUT*. La funzione `init_camera` restituisce la dimensione dell'immagine che verrà catturata dalla videocamera, mentre con la funzione `artag_init` ARTag alloca internamente un buffer con la dimensione esatta per contenere l'immagine acquisita. Devono essere forniti a questa funzione l'altezza, la larghezza dell'immagine ed anche i bit per pixel che, nel caso di una immagine RGB, sono 24:

A questo punto devono essere forniti i parametri intrinseci della videocamera per settare il frustum, in modo che il campo di vista della videocamera virtuale e di quella reale sia lo stesso: questo può essere fatto mediante la funzione `artag_set_camera_params` che accetta quattro parametri, dove i primi due parametri sono la distanza focale lungo x e y in pixels mentre gli altri due rappresentano il centro del piano immagine della videocamera.

[9]Per approfondimenti rimandiamo agli esempi forniti con l'ARTag SDK

A questo punto è possibile settare il frustum mediante la funzione OpenGL `glfrustum` (vedere appendice La grafica tridimensionale e OpenGL).

I parametri della funzione `glFrustum` possono essere ottenuti dalla seguente relazione (Figura 2.15):

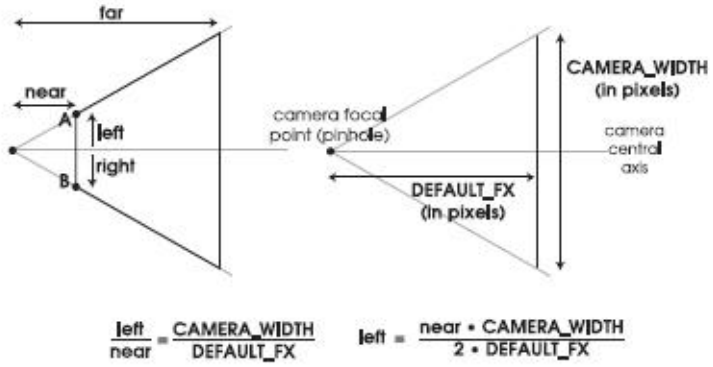


Figura 2.15: Relazione per trovare i parametri del frustum

A questo punto possiamo informare la libreria qual è il marker di cui vogliamo effettuare il tracking, mediante la funzione `artag_object_id` che come argomento prevede l'identificatore del marker e come valore di ritorno restituisce un handle che verrà successivamente utilizzato.

Ciclicamente dobbiamo catturare un frame video e per ogni frame acquisito, ARTag deve analizzare l'immagine per trovare eventuali marker: la funzione `camera_grab_bgr_blocking` viene utilizzata per acquisire frame video mentre la funzione `artag_find_objects`, che ha come argomenti il buffer che contiene il frame corrente e se l'immagine è a colori o in scala di grigi.

A questo punto, dopo aver visualizzato nella finestra grafica l'immagine acquisita dalla videocamera come texture, possiamo far settare la *model-view_matrix* ad ARTag se il marker è presente nell'immagine: la funzione `artag_is_object_found` restituisce il valore 1 se il marker di cui deve essere effettuato il tracking è presente nell'immagine e 0 altrimenti. La funzione `artag_set_object_opengl_matrix` imposta la matrice di trasformazione.

2.4.2 ARTag multi-marker tracking

Anche con questa libreria, come per le altre due precedentemente analizzate, è possibile ricavare univocamente la posizione della videocamera da un set di markers. Rispetto al caso precedentemente analizzato, ora è necessario fornire ad ARTag un file che contiene la posizione di tutti i markers rispetto ad un'origine, la quale è il punto da cui verrà ricavata poi la posizione della videocamera. Nella funzione `main()` è quindi necessario passare alla libreria il file contenente le informazioni sui marker e associare un handle a tale marker set: con la funzione `load_array_file` informiamo la libreria ARTag quale è il file che contiene le informazioni sui markers e con la funzione `artag_associate_array` associamo ad una variabile un handle per tale marker set.

Un esempio di marker set è il seguente (Figura 2.16):

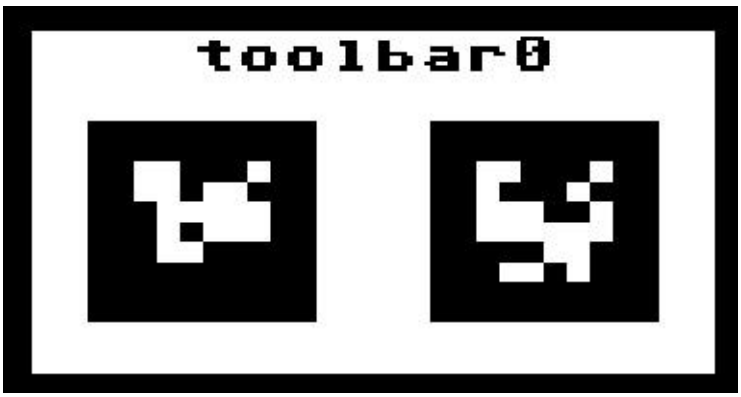


Figura 2.16: Esempio di marker set

il cui file di configurazione, ad esempio, può essere il seguente:

```
/coordframe
coordframe
name="toolbar0"
min_points=4
   marker
     id=170
     //-list of corners cw from top left
```

```
    vtx=0,0,0 //upper left
    vtx=0,40,0 //upper right
    vtx=40,40,0 //lower right
    vtx=40,0,0 //lower left
  /marker

  marker
    id=172
    //-list of corners cw from top left
    vtx=0,60,0 //upper left
    vtx=0,100,0 //upper right
    vtx=40,100,0 //lower right
    vtx=40,60,0 //lower left
  /marker
/coordframe
```

In questo caso possiamo notare che l'origine global del marker set si trova nel vertice in alto a sinistra del marker 170.

Capitolo 3

Marker Field

Il tracking ottico basato su marker è diventato una delle tecniche di tracking più popolari poiché è caratterizzato da un funzionamento in real time, da un equipaggiamento a basso costo (richiede solo una videocamera ed alcuni markers in aggiunta ad un comune PC) e dall'utilizzo di poche risorse hardware. Per utilizzare efficacemente ed efficientemente questa tecnologia, è indispensabile calibrare con accuratezza l'intero sistema. Questi processi di calibrazione sono [6]:

- Calibrazione della videocamera.
- Calibrazione dei singoli marker.
- Calibrazione di più marker (marker field).

La calibrazione della videocamera è utilizzata per ottenere i parametri intrinseci della videocamera, parametri fondamentali per definire la matrice di proiezione prospettica. ARToolkit fornisce un proprio programma di calibrazione, ma ne esistono altri, come il *Matlbab Calibration Toolbox*.

La calibrazione dei singoli marker consiste nel definirne la dimensione, l'ID del marker utilizzato e la posizione del centro del marker.

La calibrazione di più marker è fondamentale per formare un singolo sistema di coordinate per effettuare l'head tracking.

In semplici applicazioni di realtà aumentata, ogni singolo oggetto virtuale è associato ad uno specifico marker, precedentemente calibrato (cioè definito in dimensione), e viene sovrapposto al relativo marker quando viene individuato dall'algoritmo di marker detection. In questo scenario, poiché ogni oggetto virtuale è posizionato in accordo ad un marker specifico, non c'è necessità di conoscere la posizione relativa di diversi markers tra di loro.

In applicazioni di realtà aumentata più complesse, un singolo marker non è sufficiente, a causa, per esempio, dei continui spostamenti dell'utente (e quindi della videocamera) che porta inevitabilmente il marker fuori dall'area di vista della videocamera, oppure nel caso in cui l'area di visualizzazione sia troppo vasta come in applicazioni *wide-area indoors*.

In questi casi viene impiegato il cosiddetto *marker field* (o *marker set*), che contiene un insieme di markers ed un'origine globale. In un marker field, la posizione relativa di ciascun marker rispetto all'origine è conosciuta e l'origine globale può essere, ad esempio, il centro di un marker, il vertice di un marker oppure una qualsiasi posizione all'interno del marker field. In questo modo gli oggetti virtuali possono essere visualizzati nell'area del marker field anche quando alcuni marker non sono riconosciuti dall'applicazione software.

Per definire il marker field, deve essere effettuata la *multi-markers calibration*, necessaria per formare un singolo sistema di coordinate la cui origine viene definita dall'utente. La calibrazione consiste nello stimare la posizione relativa, traslazione più rotazione, tra i vari markers. Nel caso in cui tutti i markers siano molto vicini tra loro e situati sullo stesso piano, come, ad esempio, nel caso in cui siano stampati su un foglio o poster, la calibrazione può essere effettuata mediante righello, goniometro, tacheometro o altri strumenti simili in grado di misurare la posizione [23] (Figura 3.1 nella pagina successiva).

Questo approccio viene utilizzato dalle librerie precedentemente analizzate in cui, nel caso del tracking multi markers, deve essere fornito al pro-

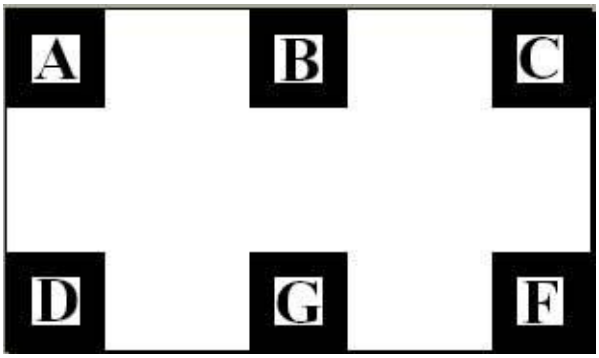


Figura 3.1: Esempio di un semplice marker field stampato su foglio

gramma un marker field con tutte le posizioni dei vari markers relative ad una origine globale misurate con appositi strumenti.

Questo metodo, tuttavia, non è più sufficiente ed efficiente nel caso in cui i markers siano posizionati molto distanti tra di loro e con angolazioni arbitrarie (Figura 3.2 nella pagina seguente), poiché poco accurato e troppo dispendioso dal punto di vista del tempo necessario ad effettuare la calibrazione.

Un secondo metodo, molto accurato, consiste nell'utilizzare un *digitizer*: un dispositivo che può misurare la posizione nello spazio 3D, come un sistema di tracking convenzionale o un laser scanner. Questo metodo tuttavia richiede attrezzature ingombranti, che non sempre possono essere utilizzati ed il tempo necessario per effettuare l'installazione e la calibrazione di questi dispositivi è abbastanza lungo.

Un metodo più efficiente che permette di ottenere una calibrazione di un marker field impiegato in ambienti di grandi dimensioni, è ottenuto attraverso una calibrazione automatica: la stessa videocamera utilizzata per il video see-through augmentation e per il marker detection, viene impiegata per catturare le immagini dell'ambiente che vengono successivamente utilizzate per calcolare la posizione relativa dei markers tra di loro.

Uematsu e Saito [36] hanno sviluppato un sistema di tracking puramente ottico, che utilizza più markers: il loro sistema impiega un insieme di immagini chiave, riprese con la videocamera, per stimare gli *extrinsic camera parameters* durante un processo di calibrazione.

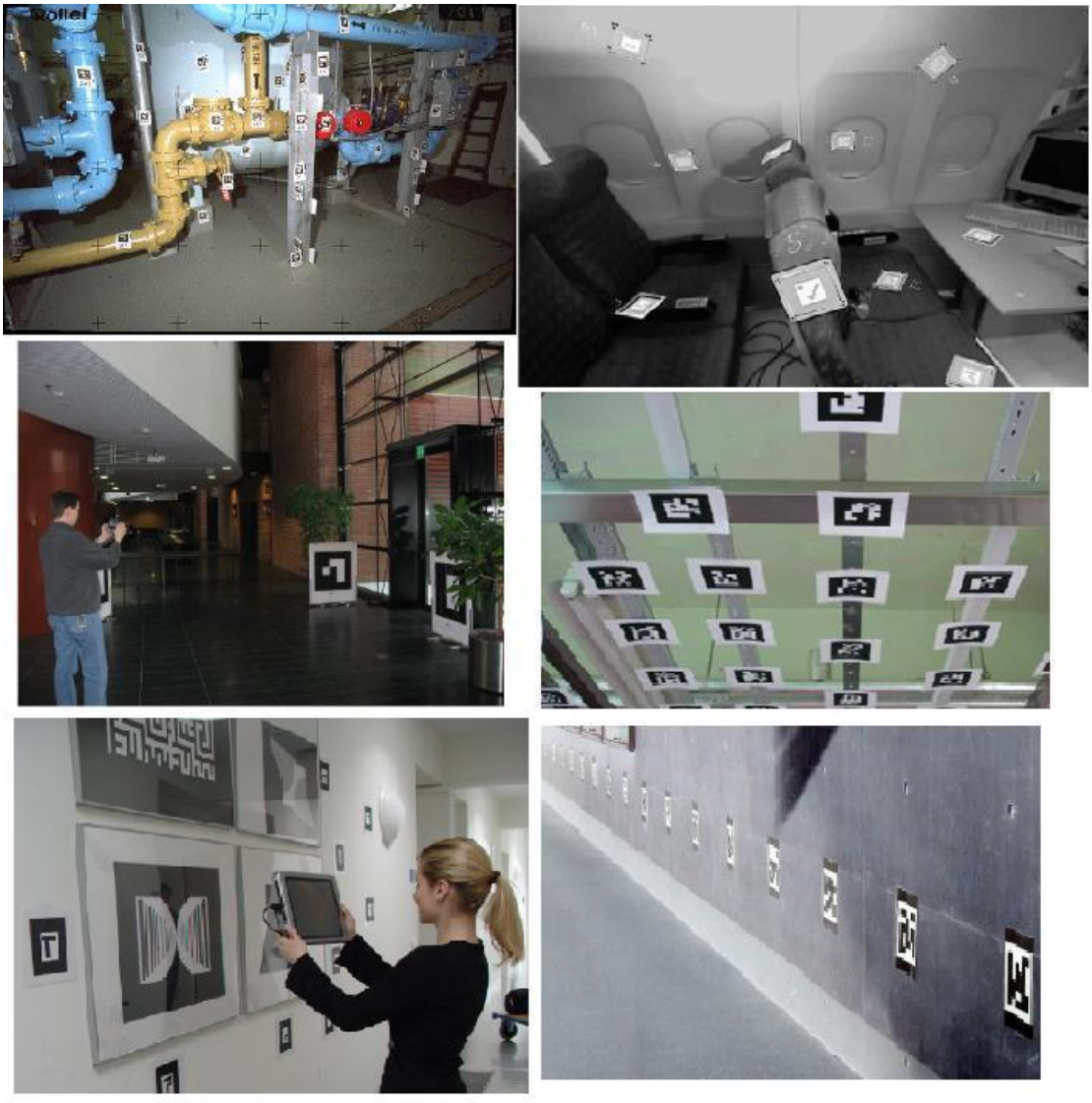


Figura 3.2: Esempi di marker field complessi [30] [18]

In [6] e [9], per calcolare rapidamente un sistema di riferimento comune per il marker field, viene impiegato il *bundle adjustment method*, una tecnica di fotogrammetria che calcola la posizione dei markers da un insieme di immagini registrate con la videocamera; con questa tecnica sono necessarie però delle conoscenze note a priori sui markers, come, ad esempio, se sono posizionati tutti sul medesimo piano ed il processo di *photogrammetric computation* è molto dispendioso dal punto di vista del tempo necessario a

calcolare la posizione dei markers.

In [20] per ottenere la calibrazione del marker field, viene impiegato un modello CAD dell'ambiente in cui sono posizionati i markers costituenti il marker field, con la loro posizione e, combinando questi dati con la posizione dei markers ottenuta attraverso il tracking ottico, è possibile ottenere una calibrazione più accurata.

Le librerie analizzate precedentemente, e cioè ARToolkit, ARToolkitPlus ed ARTag, non forniscono alcun processo di calibrazione automatico del marker field: esse assumono che l'utente abbia misurato precedentemente, in qualche modo, la posizione dei markers.

In questa tesi viene implementato l'algoritmo proposto da *Siltanen, Hakkarainen, Honkamaa* in *Automatic Marker Field Calibration* [30] (con alcune aggiunte per migliorarne il funzionamento con la libreria utilizzata), che permette di ottenere una calibrazione automatica del marker field e può essere integrato nelle librerie suddette.

Con questo algoritmo i markers possono essere posti ad una qualsiasi distanza tra di loro, ruotati di angoli arbitrari, non devono essere necessariamente posizionati sul medesimo piano e l'origine globale viene definita rispetto ad un *base marker*. Il solo requisito che il marker field deve rispettare è che i markers devono rimanere statici tra di loro e l'applicazione deve essere in grado di riconoscere almeno due markers nello stesso istante in modo che ogni marker venga relazionato con il *base marker*. Per chiarire il concetto supponiamo di avere il marker field di figura 3.3 nella pagina successiva e supponiamo che nell'area di visualizzazione della videocamera prima ci siano il base marker e il marker 1 e successivamente il marker 1 ed il marker 2. In questo caso prima viene calcolata T1 e cioè la matrice di trasformazione del marker 1 relativamente al base marker e poi T2, cioè la matrice di trasformazione del marker 2 rispetto al marker 1 e avendo a disposizione T1, è possibile ricavare la posizione del marker 2 rispetto al base marker.

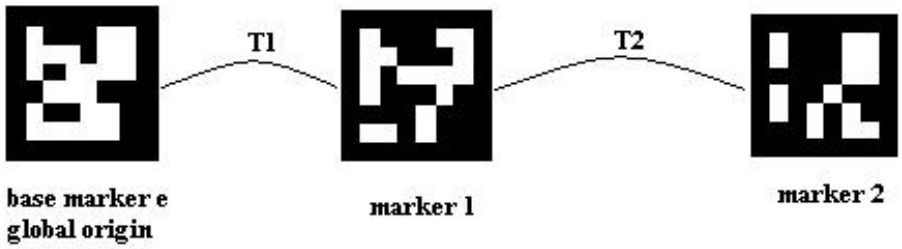


Figura 3.3: Esempio marker field

3.1 Creazione del marker field

Il processo automatico di creazione del marker field può essere suddiviso in tre sottoprocessi:

1. Individuazione e riconoscimento dei markers
2. Calcolo della matrice di trasformazione tra coppie di markers
3. Calcolo della matrice di trasformazione tra ogni markers e l'origine globale.

3.1.1 Individuazione e riconoscimento dei markers

In questa fase, l'utente si muove all'interno dell'area del marker field per cercare i markers che lo compongono. I markers che lo costituiscono sono forniti all'applicazione di *marker detection* attraverso un file di configurazione (come visto, ad esempio, per ARToolkit e ARToolkitPlus) che fornisce l'ID dei markers e la loro dimensione in una determinata unità di misura che dipende dalla libreria utilizzata. Vengono fornite anche informazioni sui markers che sono posizionati sullo stesso piano, in modo da migliorare la precisione con cui viene calcolata la matrice di trasformazione relativa tra due markers presenti sul medesimo piano.

Per ogni marker individuato e riconosciuto, viene memorizzato in una specifica struttura l'ID corrispondente, viene calcolata la matrice di trasformazione T_{ID} del marker rispetto alla videocamera (che rappresenta quindi la

posizione del marker relativamente alla videocamera) ed assegnata ed essa un valore di confidenza c_{ID} che rappresenta la qualità con la quale viene effettuato il riconoscimento del marker o quanto possiamo confidare sull'accuratezza della matrice di trasformazione T_{ID}.

3.1.2 Calcolo della matrice di trasformazione tra coppie di markers

Quando vengono riconosciuti per la prima volta i markers m1 ed m2, viene calcolata la corrispondente matrice di trasformazione T_{m1m2} e cioè

$$T_{m1m2} = T_{m1}^{-1} T_{m2}$$

che fornisce la posizione del marker m2 relativamente al marker m1.

Consideriamo ad esempio la Figura 3.4 nella pagina seguente: l'ultima colonna rappresenta la posizione del marker 2 (ovvero il marker sul quale è sovrapposto il cono virtuale) rispetto al marker 1 (quello con la sfera). Il marker 2 è posizionato, rispetto al marker 1, 142.65 mm lungo l'asse X, 27.65 mm lungo l'asse Y ed è situato all'incirca sullo stesso piano, 7.2 mm lungo l'asse Z. La prime tre colonne, che costituiscono una matrice 3x3, rappresentano la rotazione del marker 2 rispetto al marker 1.

Questa matrice viene memorizzata in memoria dinamica, in una struttura che descrive la relazione tra il marker m1 ed il marker m2. Per avere la posizione del marker m1 relativamente al marker m2 è sufficiente calcolare la seguente matrice di trasformazione:

$$T_{m2m1} = T_{m2}^{-1} T_{m1}$$

Viene anche calcolato un valore di confidenza c_{m1m2} per la matrice di trasformazione relativa T_{m1m2} che si basa sui singoli valori di confidenza c_{m1} e c_{m2} delle matrici T_{m1} e T_{m2}: una possibile scelta è, ad esempio, la media aritmetica di c_{m1} e c_{m2} oppure il valore minimo tra i due.

Una sola matrice non è sufficiente per fornire una posizione accurata del marker m2 rispetto al marker m1, poiché il calcolo della matrice e quindi della

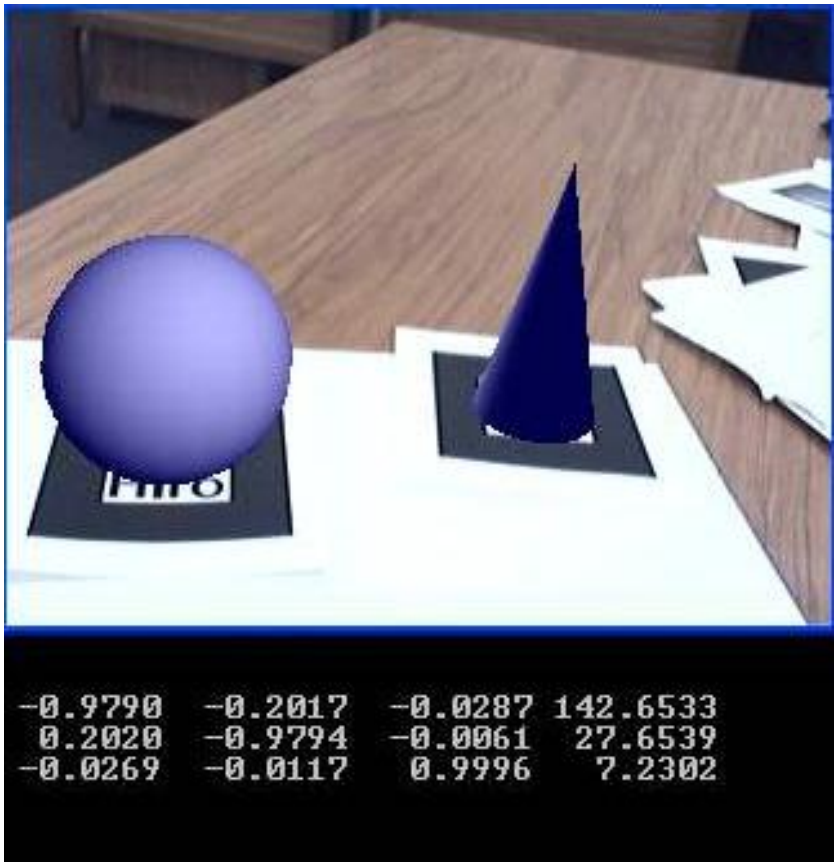


Figura 3.4: Relazione tra due marker

posizione è influenzato da diversi fattori come, ad esempio, rapidi spostamenti della videocamera o variazioni della luminosità dell'ambiente: risulta quindi necessario memorizzare N matrici di trasformazione e applicare un filtro per eliminare quelle che forniscono una posizione che si discosta eccessivamente dal valore reale. Come agisce il filtro, applicato solamente alle componenti di traslazione x, y, z della matrice, viene schematizzato nei seguenti passi:

1. Dopo aver memorizzato N matrici, esegue una somma tra tutte le componenti di traslazione x, y, z di tutte le matrici e divide il risultato finale per il numero di matrici memorizzate (N) ottenendo così una media $\bar{x}, \bar{y}, \bar{z}$:

$$\bar{x} = \frac{\sum_{i=0}^{i=N-1} x_i}{N}, \quad \bar{y} = \frac{\sum_{i=0}^{i=N-1} y_i}{N}, \quad \bar{z} = \frac{\sum_{i=0}^{i=N-1} z_i}{N}$$

2. Esegue una scansione delle matrici, eliminando quelle le cui componenti x, y, z si discostano di un certo valore, predefinito dall'utente, dalla media calcolata nel precedente punto.

3. In totale ho S matrici rimanenti (ovvero considero che ho individuato correttamente la stessa coppia per S volte). Per la Q^{esima} matrice delle S matrici, il filtro aggiorna T_{m1m2} eseguendo una media pesata:

$$T_{m1m2} = \frac{\sum_{i=1}^{Q-1} c_{12}^i}{\sum_{i=1}^{Q} c_{12}^i} T_{m1m2}^{Q-1} + \frac{c_{12}^Q}{\sum_{i=1}^{Q} c_{12}^i} T_{m1m2}^{Q} \quad con\ Q = 1...S$$

Ottenuto T_{m1m2} finale, le N matrici, precedentemente memorizzate, possono essere eliminate dalla memoria.

Questo algoritmo funziona fino a quando le relazioni tra i markers rimangono statiche, ovvero fino a quando i markers mantengono le stesse posizioni tra di loro ed è possibile vedere contemporaneamente almeno una coppia di markers per permettere il processo di calibrazione.

3.1.3 Calcolo della matrice di trasformazione tra ogni markers e l'origine globale

Ad un certo punto abbiamo abbastanza matrici di trasformazioni tra coppie di marker per calcolare la trasformazione tra ogni marker ed il base marker. Per questo scopo i marker vengono organizzati come un grafo: ogni marker è un nodo e se esiste la matrice di trasformazione tra due markers, allora i due nodi vengono connessi da un arco (Figura 3.5 nella pagina successivaa). Ad ogni arco poi viene associato un peso, che rappresenta il costo del collegamento e viene utilizzato per trovare il cammino minimo. Un possibile modello di costo può essere, ad esempio, il valore di confidenza nel riconoscimento di un marker, che tiene conto dell'accuratezza del riconoscimento dovuto alla distanza del marker dalla videocamera, o dalla risoluzione

dell'immagine acquisita. E' ragionevole anche assegnare ad ogni percorso un costo pari ad "1": questo significa che viene cercato il percorso con il minimo numero di nodi.

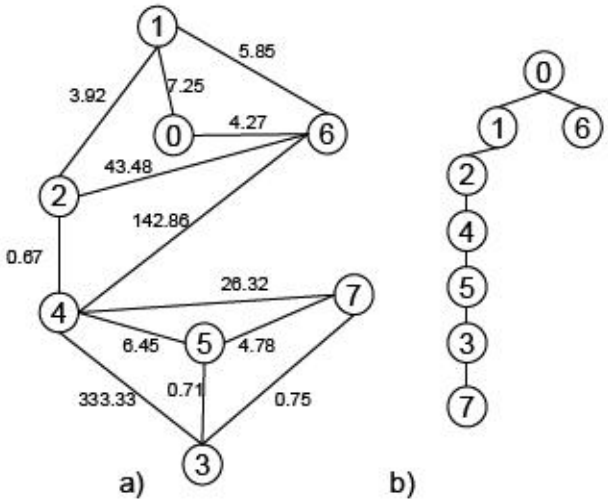


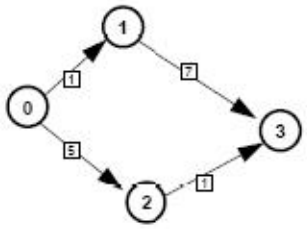
Figura 3.5: a) Grafo che rappresenta la relazione tra i markers. b) Albero ottenuto dopo l'applicazione di Dijkstra

Il prossimo passo è quello di trovare il cammino minimo da ogni marker al base marker e per questo, viene utilizzato l'algoritmo di Dijkstra. Dopo l'applicazione di questo algoritmo abbiamo un albero che rappresenta il marker field (Figura 3.5b).

Per spiegare l'algoritmo, è utile utilizzare due insiemi disgiunti di nodi che chiameremo S e Q, tali che in ogni momento $S \bigcup Q = N$ con N insieme totale dei nodi: durante l'esecuzione dell'algoritmo, un nodo p appartiene all'insieme S se abbiamo già trovato il cammino minimo da p_0 a p. Per ogni nodo p appartenente ad S o a Q, manteniamo la distanza minima da p_0 a p, che chiameremo $dist(p)$: se p appartiene ad S, $dist(p)$ è la distanza minima effettiva, se $p \in Q$, $dist(p)$ è la distanza minima stimata e può essere modificata. Per ogni nodo p, teniamo anche un puntatore, $pred(p)$, al nodo che lo precede nel cammino minimo (definitivo o stimato) da p_0 a p [14]. L'algoritmo di Dijkstra, espresso in modo informale, è il seguente:

```
1 {Q=N;
2 per ogni nodo p diverso da p0,{dist(p)=infinito,
3                                    pred(p)=vuoto;}
4 dist(p0)=0;
5 while (Q non vuoto){
6         p=estrai da Q il nodo con minima distanza;
7         per ogni nodo q successore di p{
8           lpq=lunghezza dell'arco (p,q);
9           if (dist(p) + lpq < dist(p)){
10            dist(q)=dist(p) + lpq} ;
11            pred(q)=p;
12            re-inserisci in Q il nodo q modificato;
13                 }
14         }
15  }
16 }
```

Nella fase di inizializzazione (linee 1..4) l'insieme Q viene inizializzato con tutti i nodi: le distanze cioè, sono tutte stimate e non definitive. Tali distanze vengono poste a ∞ per i nodi diversi da quello di partenza e a 0 per p_0. Ad ogni iterazione del ciclo *while* (linee 5..15), il nodo con minima distanza stimata viene tolto dall'insieme Q e considerato stimato. Successivamente, se p è il nome di tale nodo, vengono esaminati tutti i successori di p, modificando le loro distanze stimate e predecessore nel modo seguente: per ogni nodo q successore di p, connesso a p dall'arco (p, q), con lunghezza lpq, se la distanza stimata di q è minore della distanza di p più *lpq*, allora non si modifica nulla; altrimenti vuol dire che abbiamo trovato un cammino (da p_0 a q), passante per p, di lunghezza più corta e quindi $dist(q)$ deve essere aggiornato e così pure il predecessore di q, che diventa p. Quando l'insieme Q è vuoto, $dist(p)$ contiene, per ogni nodo p, la sua minima distanza da p_0, mentre il minimo cammino da p_0 a p è ricostruibile utilizzando $pred(p)$. Un esempio dell'applicazione dell'algoritmo di Dijkstra è dato dalla figura 3.6 nella pagina successiva.



S	Q	dist				pred			
		0	1	2	3	0	1	2	3
{}	{0, 1, 2, 3}	0	inft	inft	inft	-	-	-	-
{0}	{1, 2, 3}	0	1	5	inft	-	0	0	-
{0, 1}	{2, 3}	0	1	5	8	-	0	0	1
{0, 1, 2}	{3}	0	1	5	6	-	0	0	2
{0, 1, 2, 3}	{}	0	1	5	6	-	0	0	2

Figura 3.6: Passi dell'algoritmo di Dijkstra

A questo punto, possiamo ottenere la matrice di trasformazione tra il base marker e il marker, moltiplicando le matrici di trasformazione presenti lungo il cammino minimo che li connette. Quindi, la matrice di trasformazione tra il base marker b e il marker m è:

$$T_{bm} = T_{bm1}T_{m1m2}T_{m2m3}...T_{m_nm}$$

dove $m1.....m_n$ appartengono al cammino minimo dal base marker al marker m. L'origine è definita relativamente al base marker, quindi se con T_{ob} indichiamo la trasformazione dall'origine al base marker, abbiamo che la matrice di trasformazione dall'origine al marker m è:

$$T_{om} = T_{bm}T_{ob}$$

L'origine può essere anche il base marker stesso: in questo caso

$$T_{ob} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

L'insieme di tutte le matrici di trasformazioni, dal base marker ad ogni marker, definisce il marker field, che può essere memorizzato in un file per poter essere utilizzato da applicazioni di realtà aumentata nel caso di multi marker tracking.

3.1.4 Markers temporanei

Per la calibrazione del marker field a volte può essere necessario ricorrere a dei markers temporanei, cioè markers che vengono impiegati durante il processo di calibrazione e che vengono rimossi dal marker field durante il normale funzionamento dell'applicazione di realtà aumentata.

Un tipico utilizzo dei markers temporanei è dato quando il marker field deve ricoprire vasti spazi, come in applicazioni outdoor: in questo caso, se nell'ambiente non devono essere posizionati un elevato numero di markers, è possibile, in prima istanza, utilizzare dei markers temporanei per la calibrazione, in modo da ottenere matrici di trasformazioni relative più accurate tra i markers costituenti il marker field e, successivamente, possono essere rimossi (Figura 3.7)

Figura 3.7: Utilizzo di un marker temporaneo durante la calibrazione e normale funzionamento [30]

Capitolo 4

Implementazione delle tecniche presentate

In questo capitolo vengono presentati i moduli sviluppati per la creazione in modo automatico del marker field e per la visualizzazione dei modelli virtuali.

In entrambi i casi, come libreria per il marker tracking, è stata scelta, tra quelle precedentemente analizzate, ARToolkitPlus, in quanto dispone di un elevato numero di markers che vengono identificati tramite ID ed inoltre ha una bassa occupazione di memoria e di CPU, caratteristica non indifferente quando una applicazione di realtà aumentata deve essere eseguita su piccoli computer portatili, che dispongono di risorse hardware limitate, adatti ad essere trasportati da una persona.

Anche ARTag dispone di un elevato numero di markers ed è in grado di rilevarli anche se il bordo è parzialmente ostruito, tuttavia una applicazione di realtà aumentata basata su questa libreria ha una occupazione di risorse

hardware abbastanza elevata. Inoltre il codice sorgente della libreria non è open source e non offre nessuna funzione che permetta di accedere alla matrice di trasformazione tra marker e videocamera (matrice che identifica la posizione e l'orientamento del marker rispetto alla videocamera) necessaria per l'implementazione dell'algoritmo di creazione automatica del marker field.

4.1 Modulo per la creazione del marker field

L'algoritmo per la creazione automatica del marker field è stato implementato utilizzando il linguaggio C++ e la libreria ARToolkitPlus per quanto riguarda le operazioni di *marker detection* e *pose estimation* (viene utilizzata la classe `TrackerSingleMarker`, classe derivata dalla classe base `Tracker`) Il modulo per la creazione del marker field è costituito da due classi:

- `class camera_image`: classe che implementa l'interfacciamento della videocamera e che provvede ad acquisire real-time un frame video ed a visualizzarlo su schermo.
- `class marker relation`: classe che implementa l'algoritmo di creazione del marker field ed al suo salvataggio su file in modo da poter essere utilizzato successivamente da una applicazione di realtà aumentata che utilizza la libreria ARToolkitPlus nel caso multi markers.

4.1.1 Marker Detection e Pose Estimation

Per effettuare l'operazione di marker detection e pose estimation, come già detto, viene utilizzata la classe `TrackerSingleMarker` disponibile nella libreria ARToolkitPlus, in quanto, per la creazione del marker field, ogni volta che viene individuata una coppia di markers, viene calcolata la matrice di trasformazione tra la videocamera ed ogni marker della coppia e successivamente viene calcolata la matrice di trasformazione relativa tra i due markers (vedere paragrafi 3.2.1 e 3.2.2).

Poiché devono essere riconosciuti più markers, è necessario che il tracker conosca tutti i markers che costituiranno successivamente il marker field: viene quindi utilizzato un file di configurazione dove sono elencati tutti i markers, la loro dimensione, il loro ID e l'ID del base marker. Queste informazioni vengono memorizzate in una struttura di tipo `ObjectData_T` in cui è presente anche un membro dove successivamente verrà memorizzata la matrice di trasformazione tra marker e videocamera.

Inizialmente viene invocato il costruttore della classe e quindi devono essere impostati vari parametri per il tracker che sono (per una trattazione più dettagliata vedere paragrafo 2.4.1):

- formato dei pixel dell'immagine che il tracker deve trattare: l'immagine può essere in formato RGB a 24 bit, RGB565 a 16 bit o in scala di grigi a 8 bit.
- dimensione del marker espressa in millimetri.
- tipologia di marker utilizzato: se viene utilizzato un marker BCH o un marker standard di ARToolkitPlus.
- valore della *threshold*.
- algoritmo da utilizzare per la pose estimation: in questo caso viene utilizzato il *Robust Planar Pose algorithm*.

Per effettuare il marker detection viene utilizzata la funzione `arDetectMarker` che memorizza in una struttura di tipo `ARMarkerInfo` diverse informazioni sui marker individuati i cui membri principali sono:

- int id: identificatore del marker riconosciuto.
- double cf: valore di confidenza del marker riconosciuto.
- double pos: centro del marker (nelle *screen coordinates*.

Per effettuare la pose estimation tra marker e videocamera viene invece utilizzata la funzione `executeSingleMarkerPoseEstimator` che memorizza in una matrice 3x4 la posizione e l'orientamento del marker rispetto alla

videocamera. A causa di un errore intrinseco della libreria ARToolkitPlus, a volte la matrice di trasformazione risulta essere nulla e quindi il marker deve essere considerato come non riconosciuto anche se in realtà precedentemente era stato individuato.

4.1.2 Classe camera_image

Per implementare l'interfacciamento della webcam è stato scelto l'utilizzo della *DirectShow Video Processing Library*[1] (*DSVideoLib*), wrapper delle DirectShow della Microsoft, una libreria opensource e gratuita che supporta l'accesso concorrente ai buffers di acquisizione da parte di più thread.

La scelta è basata fondamentalmente sull'esigenza di dover utilizzare nell'applicazione qualsiasi tipo i webcam in commercio (i cui drivers, naturalmente, devono essere compatibili con Windows). DSVideoLib, e quindi le DirectShow, sono state utilizzate poiché consentono un'efficiente astrazione dell'hardware effettivo. Un'altra scelta possibile poteva ricadere sull'utilizzo della libreria *OpenCV*[2] (libreria di computer vision sviluppata da Intel a partire dal 1999), tuttavia il numero di webcam supportate su Windows è limitata.

Dopo che la webcam ha acquisito l'immagine, questa viene visualizzata in una finestra grafica attraverso una texture, mediante l'utilizzo della libreria OpenGL.

Segue un elenco dei principali campi e funzioni membro della classe.

- **unsigned char *cam_image**: puntatore al buffer di memoria contenente l'immagine acquisita dalla webcam.
- **unsigned char *cam_tex_img**: puntatore al buffer contenente i dati della texture.
- **int cam_height, int cam_height**: variabili che contengono le dimensioni dell'immagine acquisita.

[1] http://sourceforge.net/projects/dsvideolib/
[2] sourceforge.net/projects/opencvlibrary/

- **GLuint camera_texID**: identificatore della texture che contiene l'immagine acquisita con la webcam.
- **DSVL_VideoSource* dsvl_vs**: oggetto che si occupa della gestione dello stato della webcam.
- **MemoryBufferHandle g_mbHandle**: handler al buffer di memoria utilizato dalla webcam per l'acquisizione delle immagini.
- **BYTE* g_pPixelBuffer**: buffer di memoria contenente l'immagine acquisita.
- **camera_image()**: quando viene creato l'oggetto appartenente alla classe camera_image, viene automaticamente invocato il suo costruttore default che inizializza al valore `NULL` l'oggetto `!DSVL_VideoSource* dsvl_vs` ed il buffer `BYTE* g_pPixelBuffer`.
- **void camera_init()**: inizialmente, con la funzione propria della libreria DSVideoLib `BuildGraphFromXMLString`, si specifica all'utente che all'avvio del programma venga mostrata una schermata che consenta di impostare alcuni parametri propri della webcam, come la dimensione dell'immagine da acquisire, il *refresh rate* della webcam e il formato dei pixels (ad esempio RGB o YUV).

  Successivamente, le variabili `cam_width` e `cam_height` vengono settate con il valore rispettivamente della larghezza e dell'altezza dell'immagine acquisita e quindi viene avviato il processo di acquisizione video mediante la funzione `dsvl_vs->Run()`.

  Viene anche inizializzata la texture che conterrà l'immagine catturata dall webcam: mediante la chiamata della funzione glGenTextures(1, camera_texID) viene creato un nome di una texture memorizzandolo in `camera_texID` e attraverso `glBindTexture(GL_TEXTURE_2D, camera_texID)` si specifica che le operazioni successive che coinvolgono le texture, avvengono sulla texture bidimensionale `camera_texID`.
- **void memory_allocation()**: questa funzione provvede ad inizializzare il buffer contenente l'immagine video acquisita con la webcam e

il buffer contenente la texture. La dimensione del buffer contenente l'immagine video, essendo questa in formato RGB, ha dimensioni pari a `cam_width`*`cam_height`*3, mentre il buffer contenente la texture ha dimensione 1024*512*3 in quanto l'immagine che viene acquisita con la webcam è in formato 640*480. Il motivo di questo è che la dimensione della texture deve essere una potenza di 2 e quindi la potenza di 2 successiva a 640 equivale a 1024 mentre di 480 equivale a 512.

- **void camera_grab()**: utilizzata per effettuare l'acquisizione dell'immagine mediante la webcam che viene memorizzata nel buffer BYTE* g_pPixelBuffer. Poiché l'immagine catturata mediante DSVideoLib è ruotata di 180° rispetto alla realtà, nel buffer individuato da cam_image viene memorizzata l'immagine ruotata.

- **void copy_img_into_texture()**: per poter visualizzare l'immagine acquisita mediante la webcam in una finestra grafica con contesto OpenGL, è necessario che l'immagine venga convertita in una texture bidimensionale e sovrapposta ad un quadrato di dimensione pari alla finestra grafica.

  Mediante la funzione `glTexImage2D()` viene generata la texture (di dimensione 1024*512) con l'immagine contenuta nel buffer `cam_tex_img`. Creare una texture è un'operazione dispendiosa dal punto di vista computazionale: dopo che è stata creata la texture per il primo frame video, per le immagini successive, invece di creare una nuova texture, viene effettuata un'operazione di modifica mediante la funzione `glTexSubImage2D` che ripetutamente rimpiazza la texture creata con le immagini acquisite dalla webcam.

4.1.3 Classe marker_relation

Come detto precedentemente, questa classe implementa l'algoritmo di creazione automatica del marker field come descritto nel paragrafo 3.2 ed il relativo salvataggio su file per poter essere utilizzato successivamente dalle applicazioni di realtà aumentata.

Le operazioni che coinvolgono le matrici, come inversione di matrici o moltiplicazioni tra matrici, vengono effettuate tramite Eigen2[3], una libreria C++ template per algebra lineare.

Segue la descrizione dei principali campi e funzioni membro della classe.

- **element elem**: oggetto di tipo struttura. Ogni volta che l'algoritmo di *marker detection* individua, all'interno dell'immagine acquisita, una o più coppie di markers, viene creato un nuovo elemento di questa struttura, uno per ciascuna coppia individuata. Più dettagliatamente la struttura `element` contiene i seguenti membri:
  - int m1,m2: variabili che contengono rispettivamente gli identificatori (ID) dei marker che formano la coppia identificata. Alla variabile m1 viene assegnato sempre l'identificatore del marker (costituente la coppia) con ID minore.
  - ARFloat matrixTm1m2[3][4]: matrice 3x4 che descrive la posizione del marker m2 relativamente al marker m1 dopo che è stata eseguita l'operazione di filtraggio (vedere paragrafo 3.2.2).
  - ARFloat matrixTm2m1[3][4]: matrice 3x4 che descrive la posizione del marker m1 relativamente al marker m2 dopo che è stata eseguita l'operazione di filtraggio (vedere paragrafo 3.2.2).
  - double confidence: variabile che memorizza la sommatoria del valore di confidenza (vedere formula pag 62)
  - dyn_matrix *punt: puntatore a una struttura di tipo dyn_matrix. La prima volta che viene individuata, all'interno dell'immagine, una coppia di markers, viene allocato dinamicamente un array di questo tipo di dimensione N, pari ad un valore definito dall'utente: questo valore indica quante volte deve essere riconosciuta la stessa coppia di marker e per N volte viene memorizzata la matrice di trasformazione del marker m2 rispetto al marker m1 (ovvero la posizione del marker con ID maggiore rispetto al marker con ID minore). Questa struttura contiene anche il valore

[3]http://eigen.tuxfamily.org/index.php

di confidenza per la matrice `mat_Tm1m2` (variabile membro della struttura `dyn_matrix`): il valore è calcolato come media dei due valori di confidenza per le matrici di trasformazione dei due markers costituenti la coppia.

- int count_visibility: numero di volte che la stessa coppia di markers è stata individuata nell'immagine acquisita dalla videocamera.
- int cost: costo (o peso) associato alla coppia di markers. Questo valore è necessario per il calcolo del cammino minimo tra il base marker e ogni marker costituente il marker field.

- **shortest_path path**: oggetto di tipo struttura. La struttura dati `shortest_path`, memorizza i markers come nodi per essere organizzati come grafo ed implementa gli insiemi S, Q ed i vettori *dist* e *pred* (vedere paragrafo 3.2.3). Questi ultimi contengono, rispettivamente, le distanze effettive tra il base marker ed ogni marker del marker field e i predecessori di ogni marker come risultato dell'applicazione dell'algoritmo di Dijkstra

- **list<element>L**: contenitore sequenziale STL (*Standard Template Library*, parte integrante della libreria standard C++) di tipo `element`.

- **list<element>::iterator iter**: iteratore del contenitore sequenziale. E'un oggetto capace di puntare ad un elemento del contenitore STL e di passare efficientemente da un elemento all'altro.

- **marker_relation(ObjectData_T, ObjectData_T)**: costruttore della classe marker_relation che viene invocato quando viene individuata per la prima volta una coppia di markers. Provvede a creare un elemento della struttura `element` e quindi a calcolare per la prima volta $T_{m1m2} = T_{m1}^{-1} T_{m2}$, il valore di confidenza e ad allocare dinamicamente un array di tipo `dyn_matrix` che conterrà le matrici di trasformazioni, per la medesima coppia di markers, che verranno poi analizzate dal filtro. Una volta che l'elemento è creato, viene inserito nel contenitore sequenziale STL mediante una `L.push_back(elem)`.

- **void create_element(ObjectData_T, ObjectData_T)**: funzione che crea un nuovo elemento di tipo `element` e lo inserisce nel contenitore sequenziale STL. Alla fine, nel contenitore sequenziale sono presenti tanti elementi quante sono le coppie di markers individuate durante il processo di calibrazione.

- **void update_element(ObjectData_T, ObjectData_T)**: se una coppia di markers è già stata individuata dall'algoritmo di marker detection (e quindi è già presente nel contenitore STL), allora è necessario memorizzare le nuove matrici di trasformazione (ovvero la posizione del marker m2 rispetto al marker m1) nella struttura di tipo **dyn_matrix**. Il numero di volte N che deve essere memorizzata la matrice di trasformazione è un parametro definito dall'utente.

- **list<element>::iterator find_element(int,int,int *)**: funzione utilizzata per individuare se all'interno del contenitore sequenziale STL è presente una coppia di markers (cioè se la coppia di markers è già stata individuata precedentemente dall'algoritmo di marker detection). Restituisce la posizione dell'elemento all'interno del contenitore se l'elemento è stato trovato.

- **void apply_filter()**: Il filtro opera come descritto nel paragrafo 3.2.2. e viene eseguito quando la medesima coppia di markers è stata individuata e riconosciuta per la *N-esima* volta. Utilizzando le matrici di trasformazioni, relative ad una stessa coppia di markers, presenti nella struttura **dyn_matrix**, calcola la matrice di trasformazione finale secondo la formula di pagina 62 dopo che sono state eliminate quelle matrici che non soddisfano le condizioni imposte dal filtro.

  Può accadere che ARTookitPlus in alcuni casi non sia in grado di calcolare correttamente la matrice di trasformazione relativa tra due markers: questo succede quando, ad esempio, l'utente è posizionato lontano dai markers (lontano rispetto alla loro dimensione) oppure quando un marker viene illuminato direttamente e da vicino da luce artificiale. In questo caso si può presentare un caso come nella figura 4.1 in cui

possiamo notare che, se anche i due markers sono nello stesso piano, il marker 2 viene a trovarsi 178mm sopra il marker 1. Spesso, un errore di stima della posizione sull'asse z si riflette anche sull'asse x portando anche in questo caso ad un calcolo errato della posizione lungo questo asse di un marker relativamente all'altro.

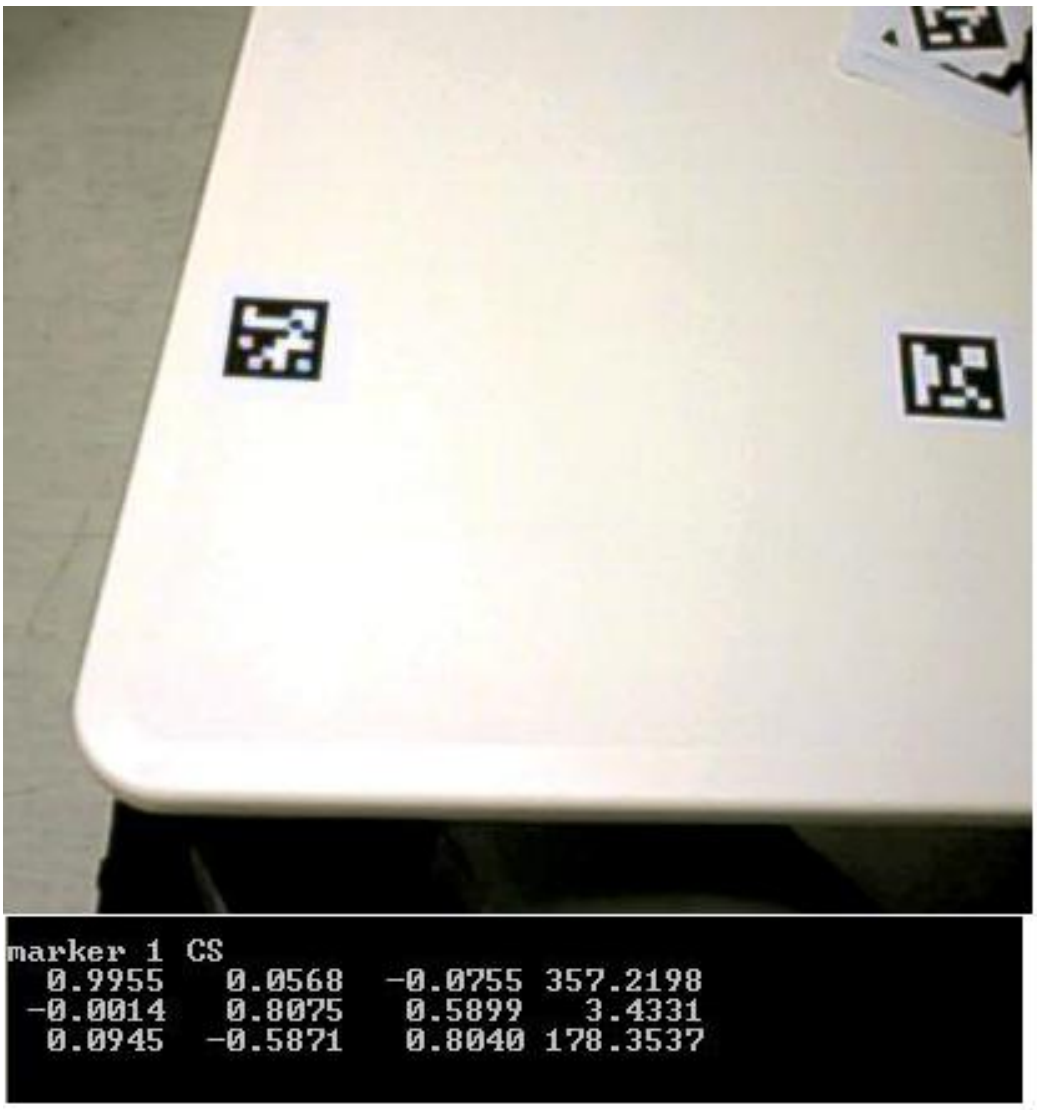


Figura 4.1: Problema nel calcolo della matrice di trasformazione relativa tra due markers

Test preliminari eseguiti in questa sede nello studio della libreria AR-

ToolkitPlus hanno messo in evidenza che un errore sull'asse z superiore a 40mm porta ad errori di posizione anche sull'asse x.

Possiamo sfruttare quindi la conoscenza che due markers sono posizionati sullo stesso piano, per istruire l'algoritmo di creazione automatica del marker field ad eliminare quelle matrici di trasformazione relativa che presentano una $z > 40$. Non possono essere invece fatte considerazioni su markers che si trovano su piani diversi.

Il filtro calcola quindi la media della componente x, y e z (paragrafo 3.2.2) e successivamente elimina le matrici le cui componenti x, y o z distano di una certa quantità in valore assoluto (che può essere predefinita o scelta dall'utente) dalle medie precedentemente calcolate.

Infine, dopo che per una stessa coppia di markers è stata calcolata la matrice di trasformazione con le matrici rimanenti dall'operazione di filtraggio, la componente z, per quelle coppie di markers che si trovano sul medesimo piano, viene posta a 0.

- **void build_graph_dijkstra_init(int, int, int)**: crea ed inizializza gli elementi necessari per il calcolo dell'algoritmo di Dijkstra: in memoria dinamica vengono allocati gli array *S*, *Q*, *pred* e *dist* di dimensione pari al numero dei marker costituenti il marker field e la rappresentazione dei markers come grafo viene effettuata mediante una *matrice di adiacenza* (vedere paragrafo 3.2.3).

- **void dijkstra_algorithm(int)**: codice che implementa l'algoritmo di Dijkstra (vedere paragreafo 3.2.3 per lo pseudocodice dell'algoritmo) per il calcolo dei cammini a costo minimo tra il base marker ed ogni marker facente parte del marker field.

- **void save_marker_field(ObjectData_T*, int, int, int, int*)**: calcola la matrice di trasformazione tra l'origine ed ogni marker secondo la seguente formula: $T_{om} = T_{bm}T_{ob}$ dove $T_{bm} = T_{bm1}T_{m1m2}T_{m2m3}...T_{m_nm}$ e salva il marker field in un file secondo la formattazione prevista da ARToolkit per il tracking multi markers.

4.2 Modulo per la visualizzazione dei modelli virtuali

Anche in questo caso per effettuare il tracking dei markers viene sfruttata la libreria ARToolkitPlus e per visualizzare i modelli virtuali viene utilizzato XVR[4].

4.2.1 XVR: eXtreme Virtual Reality

XVR è un framework integrato per lo sviluppo rapido di applicazioni complesse di realtà virtuale, sviluppato a partire dal 2001 presso il laboratorio PERCRO. Originariamente creato per lo sviluppo di applicazioni 3D web-oriented, oggi supporta un'ampia gamma di dispositivi per la realtà virtuale come, ad esempio, mouse 3D, dispositivi per la motion capture, sistemi di proiezione stereo e HMD ed utilizza una engine grafica avanzata per la visualizzazione in tempo reale di modelli tridimensionali complessi. L'engine si basa sulla libreria grafica OpenGL.

Le applicazioni XVR sono sviluppate attraverso un linguaggio di scripting dedicato, S3D, la cui sintassi è molto simile al C++ e al Java, con il quale i programmatori hanno a disposizione una serie di comandi e costrutti che permettono di controllare ogni aspetto del sistema, dalle animazioni 3D alle interfacce aptiche.

Attualmente XVR è suddiviso in due moduli principali: l'*ActiveX Control module* che comprende le componenti più basilari della tecnologia, come la verifica delle versioni e le interfacce dei plug-in, e realizza l'interfaccia web e lo *XVR Virtual Machine (VM)*[5] *module* che contiene il nocciolo della tecnologia, come l'engine grafico 3D, l'engine multimediale e tutti i moduli software che gestiscono le altre funzionalità proprie di XVR. E' anche possibile caricare moduli aggiuntivi che forniscano funzionalità avanzate non direttamente disponibili, attraverso librerie esterne (DLLs[6]) personalizzate.

[4]http://www.vrmedia.it/

[5]per una definizione di Virtual Machine (macchina virtuale) si veda ad esempio http://it.wikipedia.org/wiki/Macchina_virtuale

[6]Dynamic Linked Library

La XVR-VM, come molte altre VMs, possiede un proprio set di istruzioni *byte code*, un set di *registri*, uno *stack* ed infine un'area di memoria in cui memorzzare i metodi. Il linguaggio di scripting S3D permette di specificare il comportamento dell'applicazione mettendo a disposizione le funzionalità basilari ed i metodi VR-oriented, disponibili come funzioni o classi. Gli script S3D sono compilati in byte code in modo da poter essere eseguiti dalla XVR-VM (Figura 4.2).

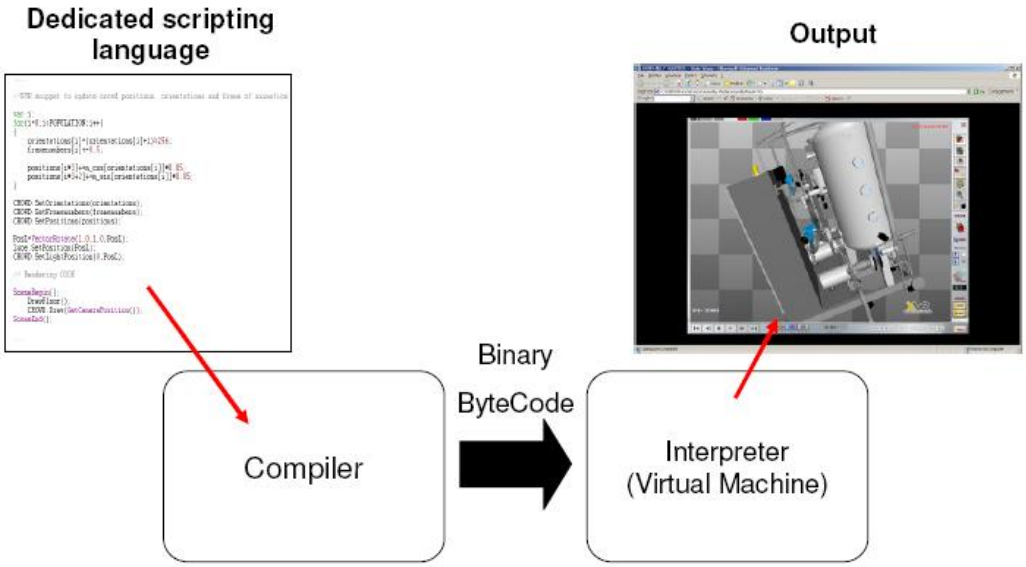


Figura 4.2: Flusso di sviluppo di una applicazione con XVR

Il bytecode prodotto dalla compilazione (.bin) può essere eseguito dalla VM mediante un'applicazione ad hoc oppure inserito in una pagina HTML. Quando si accede alla pagina HTML che contiene l'applicazione XVR, viene controllata la versione dell'XVR engine, viene scaricato il bytecode dell'applicazione, vengono scaricati i file di dati associati all'applicazione (textures, modelli 3D) ed infine viene eseguito il bytecode sulla XVR-VM.

4.2.1.1 L'ambiente di sviluppo XVR

La piattaforma XVR è distribuita con alcuni programmi di utilità per la progettazion e l'implementazione di applicazioni di realtà virtuale. Nella figura 4.3 è possibile vedere l'IDE del framework, denominato XVR Studio.

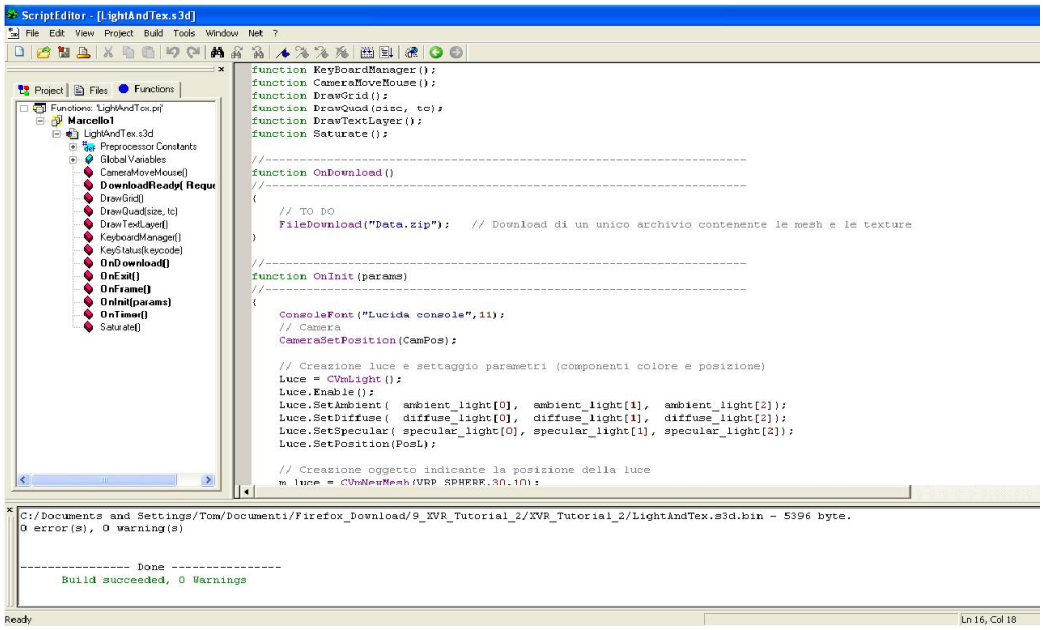


Figura 4.3: XVR Studio

Tra le sue funzionalità vi sono:

- un project manager
- un compilatore
- un editor per il linguaggio S3D
- un modulo per pubblicare il progetto su web
- una finestra dove vengono visualizzate informazioni utili alla fase di *debug*

Al fine di velocizzare ulteriormente la preparazione di una applicazione di VR, XVR Studio offre anche uno *wizard* che guida il programmatore durante il setup dell'applicazione con una serie di passi. Con tale wizard è possibile scegliere i parametri di base relativi alla scena virtuale come, ad esempio, il numero e la posizione delle luci e del punto di vista.

Come risultato di questa fase di inizializzazione, XVR Studio produce uno script S3D completo su cui il programmatore può intervenire per personalizzare l'applicazione inserendo il proprio codice.

4.2.1.2 Il linguaggio S3D

In generale una applicazione XVR si basa su sei funzioni fondamentali che costituiscono la base di ogni progetto:

- OnDownload()
- OnInit ()
- OnFrame()
- OnUpdate()
- OnEvent()
- OnExit()

La funzione OnDownload() è la prima ad essere eseguita dal framework e provvede a reperire tutte le risorse necessarie (come le librerie personalizzate o modelli 3D) al corretto funzionamento dell'applicazione.

Nella funzione OnInit() si trova invece il codice necessario per l'inizializzazione dell'applicazione. Tutti i comandi sono eseguiti sequenzialmente e nessun'altra funzione diventa attiva fino a quando la OnInit() non termina la sua esecuzione.

La funzione OnFrame() è responsabile della produzione dell'output grafico. Questa funzione è l'unica che può contenere comandi di grafica in quanto il contesto grafico è accessibile solamente durante la sua esecuzione: se i comandi di grafica vengono richiamati dal corpo di altre funzioni, essi non produrranno nessun effetto. Di default viene chiamata ad una frequenza di 60 Hz

Un'altra funzione non elencata, ma che può risultare utile in molte situazioni è la OnTimer() che viene eseguita ad intervalli regolari indipendentemente dalla OnFrame(). In questa funzione possono essere inseriti quei comandi che devono essere eseguiti ad intervalli regolari ma che non sono correlati con il rendering grafico. E' possibile impostare alcuni parametri per selezionare la frequenza di esecuzione di questa funzione che, grazie all'alta risoluzione del timer, può essere eseguita fino a mille volte al secondo.

La funzione OnEvent() è indipendente sia dalla OnFrame() che dalla OnTimer(). Viene richiamata ogni volta che l'applicazione riceve dei *messaggi*. Questi messaggi possono essere generati esternamente dal sistema operativo, oppure internamente dal programma XVR stesso.

La funzione OnExit() viene eseguita nel momento in cui l'applicazione termina la propria esecuzione o quando l'utente chiude la finestra che la contiene.

4.2.1.3 Importazione di funzioni tramite librerie esterne

Le funzioni offerte da XVR possono essere aggiunte e personalizzate utilizzando delle librerie DLL esterne. XVR dispone di una classe, la `CVmExternDLL`, tramite la quale viene caricata dinamicamente una DLL specificandone il percorso all'interno del file system:

```
var library = CVmExternDLL("myLib.dll");
```

Una volta che l'oggetto `CVmExternDLL` è stato creato, questo riferisce la DLL, ma è sostanzialmente vuoto ed è necessario importare una ad una le funzioni della libreria che si intendono utilizzare nell'applicazione utilizzando `__AddFunction()`:

```
library.__AddFunction(C_FLOAT, "myFunction", C_PFLOAT, C_INT);
```

dove il primo parametro è il tipo del valore di ritorno della funzione, il secondo parametro è il nome della funzione importata e i parametri successivi rappresentano i tipi dei parametri che ha come argomento la funzione importata.

A questo punto è possibile richiamare la funzione importata come metodo dell'oggetto `CVmExternDLL`:

```
var vet = [1.0, 2.0, 3.0];
var res = library.myFunction(a, 3);
```

4.2.1.4 Esportazione di funzioni personalizzate

La programmazione di un modulo di XVR prevede l'esportazione di un insieme di funzioni affinché possano essere importate come descritto precedentemente.

Se viene creato un modulo con il linguaggio C/C++, le funzioni da esportare devono essere dichiarate utilizzando le parole chiave:

```
extern "C" __declspec(dllexport)
```

che servono ad indicare al *linker* che tali funzioni devono essere rese pubbliche ed accessibili tramite un identificatore che coincide con il nome dato alla funzione stessa.

La funzione `myFunction` dell'esempio precedente può essere:

```
extern "C" __declspec(dllexport)
float myFunction(float *f, int n){
   float a = 0;
   for (int i = 0; i < n; i++)
      a += f[i];
   return a;
}
```

4.2.2 Integrazione con XVR

Affinchè XVR possa effettuare il tracking in un ambiente multi-markers è stato necessario implementare una DLL ed integrarla nel framework. Anche in questo caso viene utilizzata la libreria ARToolkitPlus per effettuare il marker detection e pose estimation.

Le funzioni esportate dalla DLL sono:

- **extern C __declspec(dllexport)void artkplus_init(char* file_name, char* markerboard)**: funzione che inizializza l'ambiente ARToolkitPlus per effettuare il tracking mult-markers ed avvia la videocamera.

- **extern C __declspec(dllexport) void opengl_mem_init()**: funzione che alloca i buffers per contenere sia l'immagine catturata tramite la videocamera che la texture.

- **extern C __declspec(dllexport) int artkplus_grab_draw()**: funzione che visualizza, nella finestra con contesto OpenGL, l'immagine catturata con la videocamera mediante texture, esegue l'algoritmo di marker detection e pose estimation ed infine imposta la matrice di proiezione prospettica. Se i markers che formano il marker field sono posizionati tutti sul medesimo piano è possibile utilizzare, come algoritmo di pose estimation, lo RPP (Robust Planar Pose algorithm), altrimenti viene utilizzato l'algoritmo di default previsto da ARToolkit.

- **extern C __declspec(dllexport) void artkplus_set_modelview_matrix()**: funzione che imposta la matrice di trasformazione ottenuta mediante l'algoritmo di pose estimation.

- **extern C __declspec(dllexport) void close_camera()**: dealloca le risorse utilizzate dalla videocamera.

Capitolo 5

Test e Analisi della creazione del Markerfield

In questo capitolo si analizza l'accuratezza con cui viene creato il marker field mediante l'algoritmo proposto precedentemente, in varie condizioni operative.

La posizione dei markers all'interno del marker field, ottenuta attraverso la calibrazione automatica, ovviamente differisce dalla posizione reale sia a causa dell'algoritmo stesso che calcola il marker field sia a causa dell'algoritmo di marker detection (dovuto alla threshold utilizzata, alla non corretta estrazione dei contorni del marker, agli errori di approssimazione nel calcolo delle matrici).

Per ogni marker presente nel marker field calcoliamo la distanza geometrica (considerando le componenti x,y e z) tra il marker nel marker field reale e lo stesso marker prodotto automaticamente (Figura 5.1 nella pagina successiva) dopodiché, considerando la totalità dei markers presenti nel marker

field, calcoliamo la distanza geometrica media:

$$\frac{\sum_{i=1}^{N} \sqrt{(X_R^i - X_A^i)^2 + (Y_R^i - Y_A^i)^2 + (Z_R^i - Z_A^i)^2}}{N}$$

dove N rappresenta il numero di markers mentre i pedici R ed A indicano rispettivamente la posizione reale e quella ricavata automaticamente del marker all'interno del marker field rispetto al base marker.

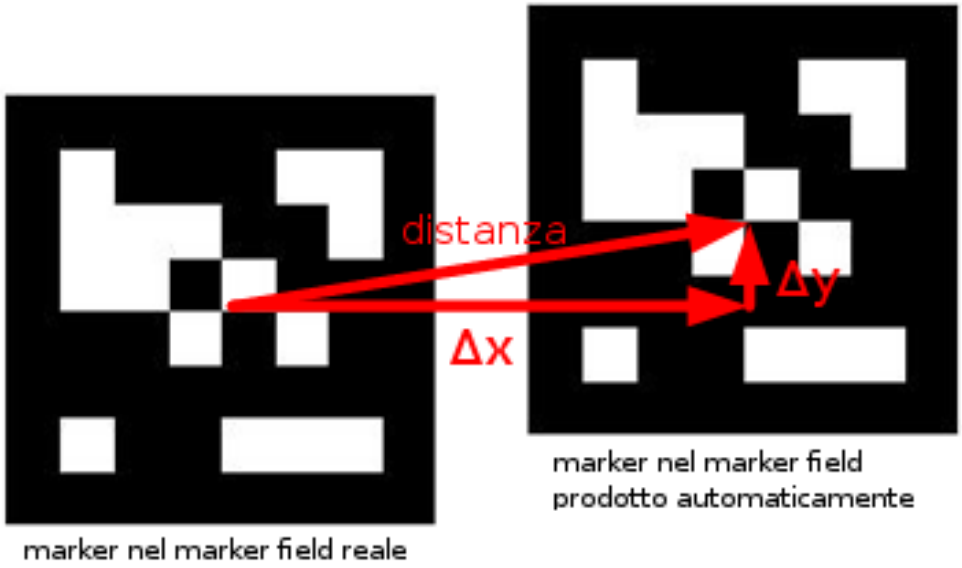


Figura 5.1: Distanza gemetrica tra il marker nel marker field reale e lo stesso marker nel marker field prodotto automaticamente nel caso a due dimensioni

La distanza può essere assunta come l'errore con cui è stato prodotto il marker field: più l'errore ottenuto è basso e più la posizione dei markers all'interno del marker field ottenuto automaticamente si avvicina a quella reale.

I test sono stati eseguiti registrando un file video dell'ambiente di lavoro: l'utente si sposta all'interno del marker field inquadrando coppie di markers sufficienti affinché possa essere trovato un percorso tra il base marker ed ogni marker (vedere capitolo 3 per l'esatto funzionamento dell'algoritmo). Successivamente, variando opportuni parametri (ambientali o dell'algoritmo stesso), è possibile confrontare i markers fields prodotti a partire dallo stesso video.

Considerata la molteplicità delle configurazioni in cui può essere prodotto un marker field, i test effettuati si limitano ad un sottoinsieme selezionato, cercando di mettere in evidenza le condizioni ottimali in cui eseguire l'algoritmo di creazione automatica.

Alcuni dei parametri di cui possiamo tenere conto nella creazione del marker field sono:

- Illuminazione: all'interno del marker field, l'illuminazione può essere naturale o artificiale.
- Video a *full resolution* o *half resolution*: il video registrato può essere analizzato alla risoluzione con cui è stato acquisito dalla webcam oppure a metà risoluzione.
- Distanza: l'utente che si sposta all'interno del marker field può inquadrare coppie di markers a differenti distanze.
- Costo: quando viene inividuata una coppia di markers, il costo associato al cammino che li connette può essere opportunamente scelto considerando diversi fattori.

5.1 Test 1

Il primo test è stato effettuato in un ambiente illuminato uniformemente (non sono presenti evidenti zone d'ombra) con luce artificiale con 10 markers di dimensione 120 millimetri che costituiscono il marker field.

Il video è stato registrato con una webcam ad una risoluzione di 640x480 pixels, inquadrando due markers per volta ad una distanza di circa 1 metro da essi. Gli screenshots estratti dal file video, che mostrano le coppie di markers considerate per produrre il marker field, sono mostrati in figura 5.2 nella pagina seguente

Il video registrato, viene quindi analizzato da ARToolkitPlus per effettuare il marker detection e la pose estimation.

Figura 5.2: Screenshots delle coppie di markers considerate nell'ambiente di riferimento

La scelta del base marker può essere arbitraria e fornita come input all'algoritmo ma una buona scelta è quella di prendere un marker che sia posizionato nella parte centrale del marker field: in questo modo il percorso dal base marker agli altri markers non diventa troppo lungo. Inoltre, come metrica di costo, è stato scelto un valore unitario: quando viene eseguito l'algoritmo di Dijkstra, il percorso tra il base marker ed ogni marker è quello con il minore numero di markers. In questo test, considerare come costo di un collegamento tra una coppia di markers un valore unitario o un qualsiasi altro valore è indifferente, poiché esiste un cammino minimo univoco che collega il base marker a tutti gli altri markers.

Come descritto nel paragrafo 3.2.3 ogni marker può essere rappresentato come un nodo e quando l'algoritmo di marker detection individua nel frame

video una coppia di markers, questi vengono connessi da un arco a cui è associato il relativo costo (o peso): il grafo che rappresenta le relazioni tra i markers nello scenario in questione è rappresentato nella figura 5.3.

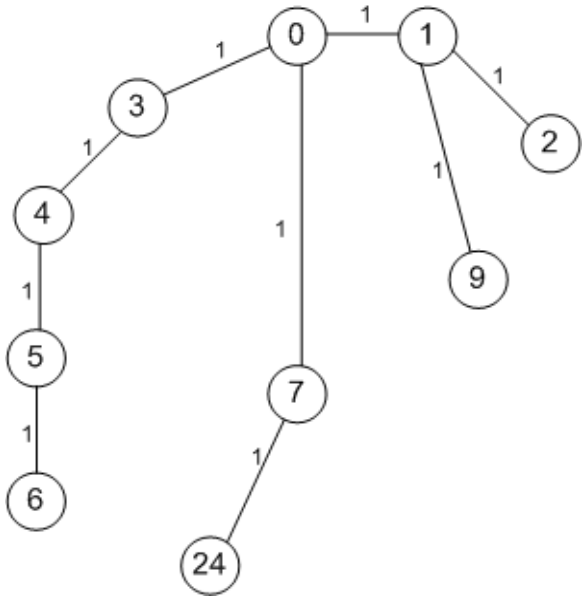


Figura 5.3: Grafo che rappresenta le relazioni tra i markers (1 metro)

Dopo l'esecuzione dell'algoritmo di Dijkstra otteniamo la rappresentazione ad albero del marker field di figura 5.4

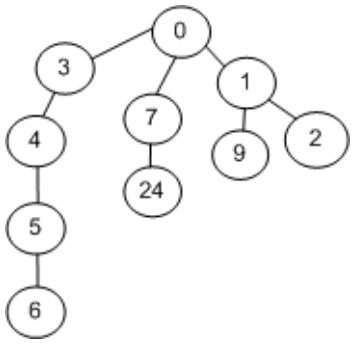


Figura 5.4: Albero ottenuto dopo l'applicazione di Dijkstra al grafo di figura 5.3

Un altro parametro, intrinseco all'algoritmo di costruzione del marker field, che deve essere considerato è il valore da assegnare al filtro dell'algo-

ritmo, al fine di eliminare le matrici le cui componenti distano di una certa quantità dalla media calcolata[1].

In un primo momento sono stati registrati venti video nelle condizioni precedentemente descritte, inquadrando le stesse coppie di markers in ogni video. Successivamente, per ogni video registrato, è stato passato al filtro dell'algoritmo di creazione automatica un valore di 5, 10, 15, 20, 30, 40 e 50 millimetri (cioè vengono eliminate le matrici le cui componenti x o y o z distano di una quantità maggiore di 5, 10,....., 50 mm) per produrre il marker field ed è stata calcolata la distanza media[2] di un marker presente nel marker field prodotto automaticamente rispetto al medesimo marker nel marker field reale, per ogni valore passato al filtro. Infine, ne è stata calcolata una media considerando il numero di video registrati (Figura 5.5)

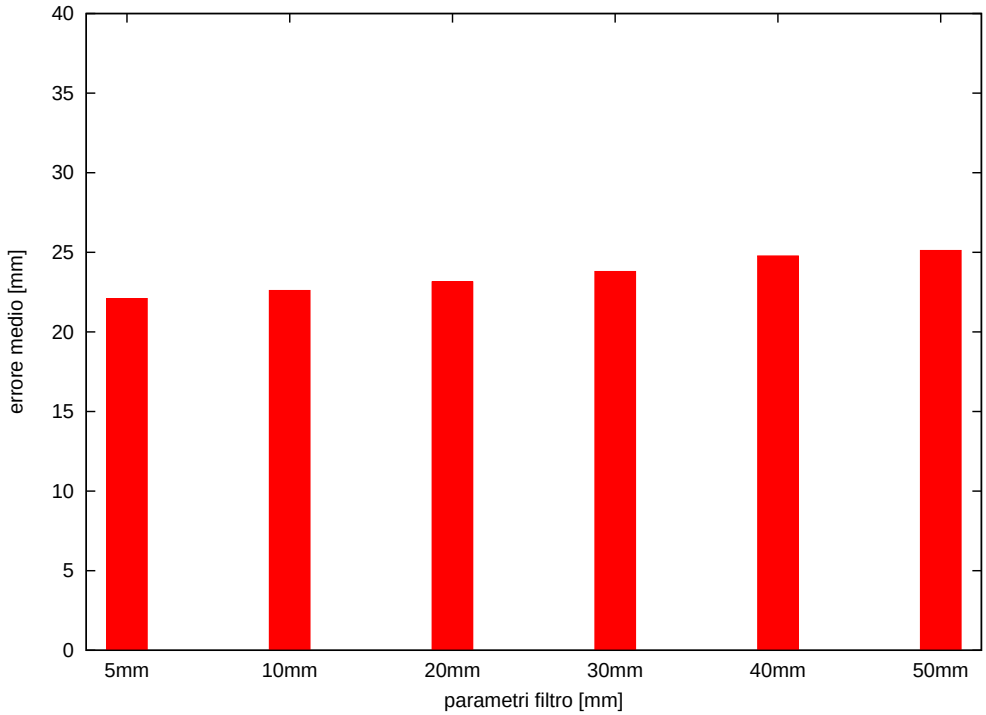


Figura 5.5: Distanza media di un marker considerato nel marker field reale rispetto a quello automatico variando i parametri del filtro.

[1] vedere funzione `apply_filter` pagina 77

[2] vedere pagina 87

Osservando il grafico possiamo notare che, anche utilizzando un valore di 50 mm, la distanza media di un marker nel marker field reale rispetto al medesimo marker nel marker field prodotto automaticamente, non varia molto rispetto all'utilizzare un valore di 5 mm.

Un vantaggio nell'utilizzare un valore di 50 mm è che il filtro non elimina un significativo numero di matrici di trasformazione, permettendo così di ottenere un marker field anche nel caso in cui le coppie di markers siano individuate un basso numero di volte. E'stato osservato infatti che, inquadrando poche volte una coppia di markers $\simeq 10$ ed utilizzando, nel filtro, valori inferiori a 50 mm, spesso il marker field non veniva prodotto.

Un utilizzatore non esperto, che impiega l'algoritmo per scopi personali e non attento al numero di volte che inquadra una coppia, può ottenere un marker field abbastanza accurato anche con poche matrici di trasformazioni acquisite.

I prossimi test sono stati eseguiti utilizzando implicitamente questo valore per il filtro.

Con i video registrati nello scenario descritto precedentemente, considerando le coppie di figura 5.2 nella pagina 88, è stato fatto un confronto tra i markers fields prodotti inquadrando un numero variabile di volte una stessa coppia: dapprima sono stati considerati tutti i frame del video registrato, avendo in media 250 frame in cui è presente la stessa coppia di markers per poi considerare, successivamente, un frame ogni 2, 4, 8 ed infine 16 (ovvero ogni volta viene dimezzato il numero totale di frame considerati), dove la media in cui compare la stessa coppia scende a 15 frame. Su queste matrici viene poi applicato il filtro come descritto dalla funzione `apply_filter`[3]

Per produrre il marker field in prima istanza sono stati considerati tutti i frame del video registrato, avendo in media 250 frame in cui è presente la medesima coppia di markers (e quindi vengono memorizzate 250 matrici di trasformazione che rappresentano la relazione di un marker rispetto all'altro della stessa coppia) per poi considerare, successivamente, un frame ogni 2, 4, 8 ed infine 16 in cui la media in cui compare la stessa coppia scende a

[3]vedere pagina 75

15 frame. Su queste matrici viene poi applicato il filtro come descritto dalla funzione `apply_filter`.

La figura 5.6 mostra l'errore ottenuto facendo una media dell'errore ottenuto dai 20 video registrati.

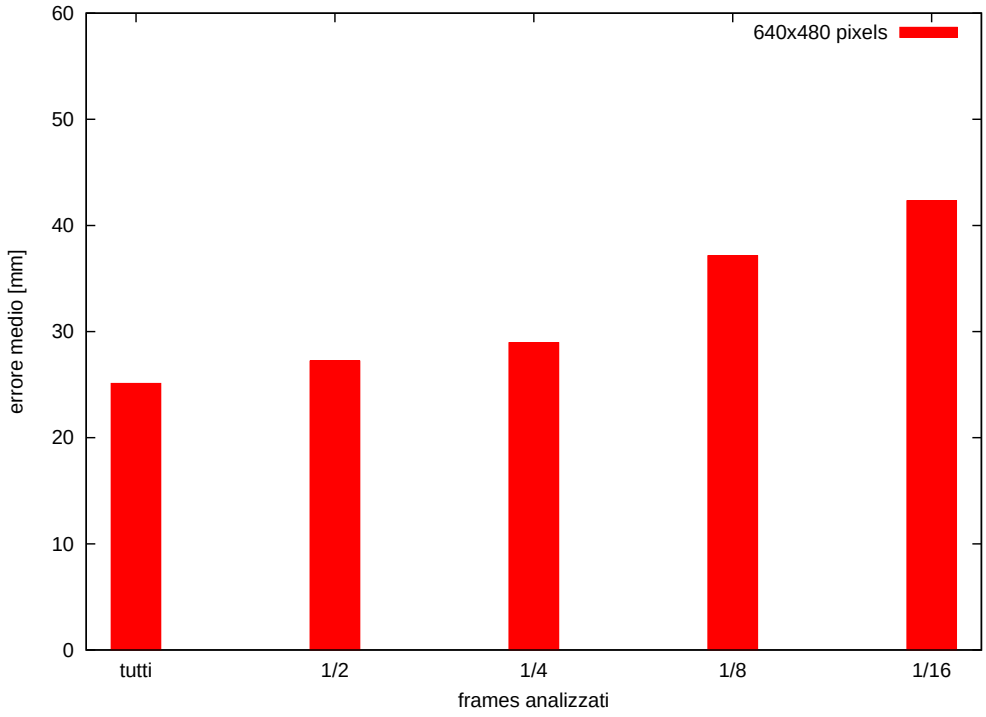


Figura 5.6: Errore medio ottenuto considerando tutti i frame, 1 frame ogni 2, 1 frame ogni 4, 1 frame ogni 8 ed 1 frame ogni 16.

Possiamo notare che l'errore aumenta al diminuire dei frames analizzati. Considerando un alto numero di frames, vengono memorizzate molte matrici di trasformazioni relative tra una coppia di markers: il filtro, calcolando la media sulle componenti x, y e z delle matrici memorizzate, restituisce la posizione relativa di un marker rispetto all'altro marker della coppia che si avvicina alla posizione reale, poiché, con un elevato numero di matrici di trasformazione, gli errori dovuti al calcolo di pose estimation, all'estrazione dei contorni, agli errori di arrotondamento nei calcoli ed all'applicazione della threshold, vengono attutiti.

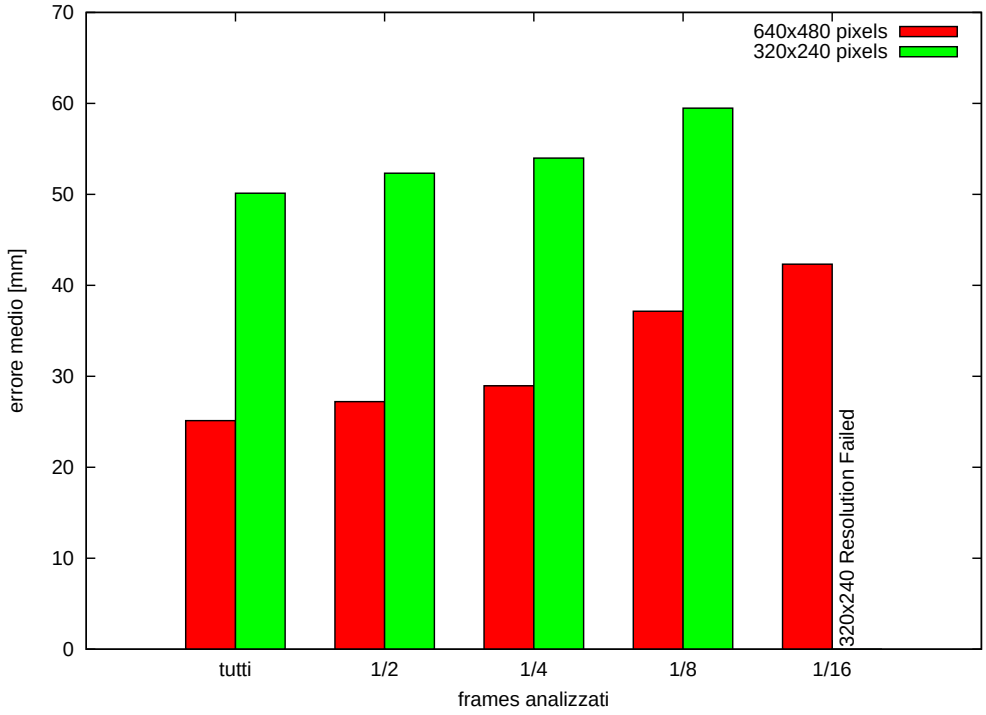


Figura 5.7: Confronto tra l'errore medio ottenuto considerando il video ad una risoluzione di 640x480 pixels e 320x240 pixels

Un altro parametro che è stato preso in considerazione è la risoluzione video: nel prossimo test la risoluzione dei video registrati è stata posta a 320x240 pixels, la metà dell'originale, e nella figura 5.7 viene mostrato un confronto tra l'errore ottenuto nella creazione del marker field prodotto a piena risoluzione con quello prodotto a metà risoluzione considerando tutti i frames del video.

Come precedentemente effettuato, viene fatta una media dell'errore sul numero di video registrati, ovvero cinquanta.

L'errore ottenuto considerando una risoluzione di 320x240 pixels è considerevolmente più elevato rispetto al caso di 640x480 pixels: l'estrazione dei contorni dei markers avviene con meno precisione con la conseguenza di ottenere una pose estimation non accurata.

Inoltre, per un basso numero di frames considerati, in molti casi il marker field non viene prodotto poiché, per qualche coppia di markers, non viene

ottenuta nessuna matrice di trasformazione valida.

Un vantaggio nell'uso di basse risoluzioni è un incremento della velocità con cui viene analizzata l'immagine, fattore che può essere utile se l'algoritmo viene utilizzato su dispositivi con basse capacità di calcolo, come i dispositivi portatili. E' evidente però che, se utilizzato in queste condizioni, per ottenere un marker field è necessario che una stessa coppia di markers sia inquadrata molte volte.

5.2 Test 2

Con la stessa configurazione di markers presentata precedentemente ed inquadrando il marker field ad una distanza di circa due metri dai markers, otteniamo il seguente grafo che mostra le coppie di markers individuate (Figura 5.8).

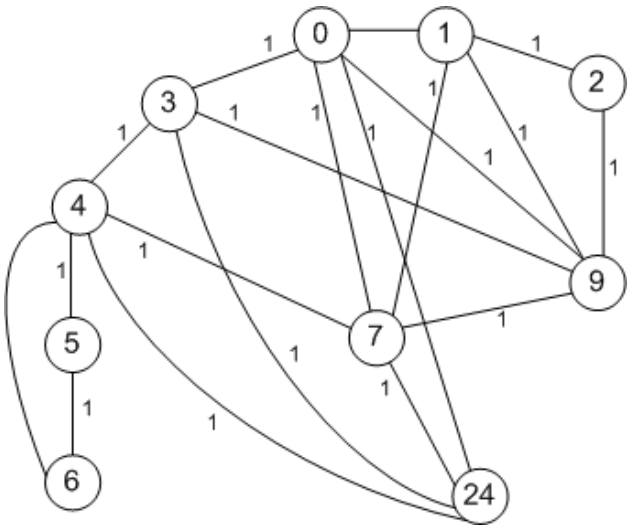


Figura 5.8: Grafo che rappresenta le relazioni tra i markers (2 metri)

Anche in questo caso il base marker è il marker con ID uguale a 0 ed il costo associato ad ogni arco è stato posto ad un valore unitario.

Dopo l'applicazione dell'algoritmo di Dijkstra, la rappresentazione ad albero del marker field è quella di figura 5.9 nella pagina seguente.

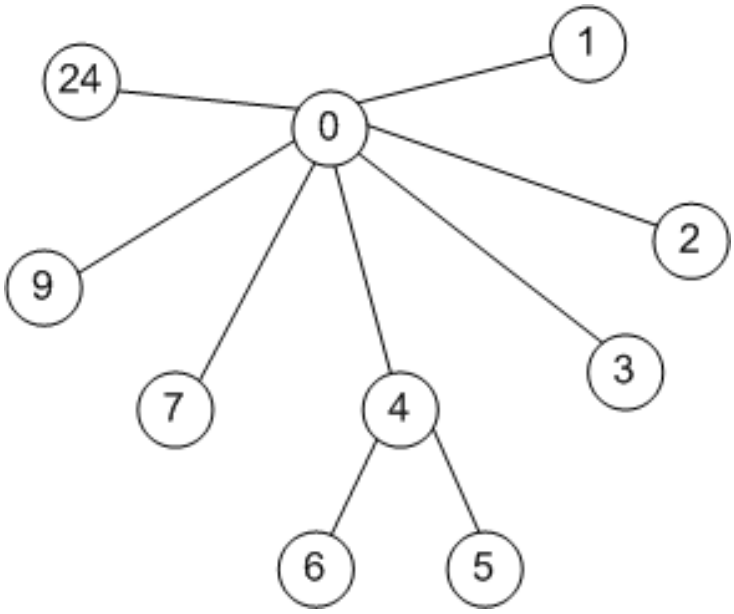


Figura 5.9: Albero ottenuto dopo l'applicazione dell'algoritmo di Dijkstra al grafo di figura 5.8 nella pagina precedente

L'errore medio sul numero di video analizzati è rappresentato nella figura 5.10 nella pagina successiva dove si può notare che l'errore, rispetto al caso in cui i marker siano inquadrati ad una distanza di circa 1 metro come nel test precedente, sia più elevato ed aumenta al diminuire dei frames considerati.

Se l'allineamento di contenuti virtuali tridimensionali (testi o immagini) a determinati oggetti non è un vincolo stringente, ma è richiesto solamente di visualizzare delle immagini virtuali all'interno del marker field, allora il marker field prodotto automaticamente può essere accettabile.

Dimezzando la risoluzione del video, il marker field non viene prodotto anche se vengono analizzati tutti i frames: i markers sono troppo piccoli rispetto alla distanza dalla quale vengono osservati impedendo così all'algoritmo di marker detection di individuare correttamente dei markers.

Se il costo del collegamento viene definito come la media aritmetica dell'inverso dei due valori di confidenza con i quali vengono individuati i markers, l'albero ottenuto dopo l'applicazione di Dijkstra è il medesimo di figura 5.8 nella pagina precedente.

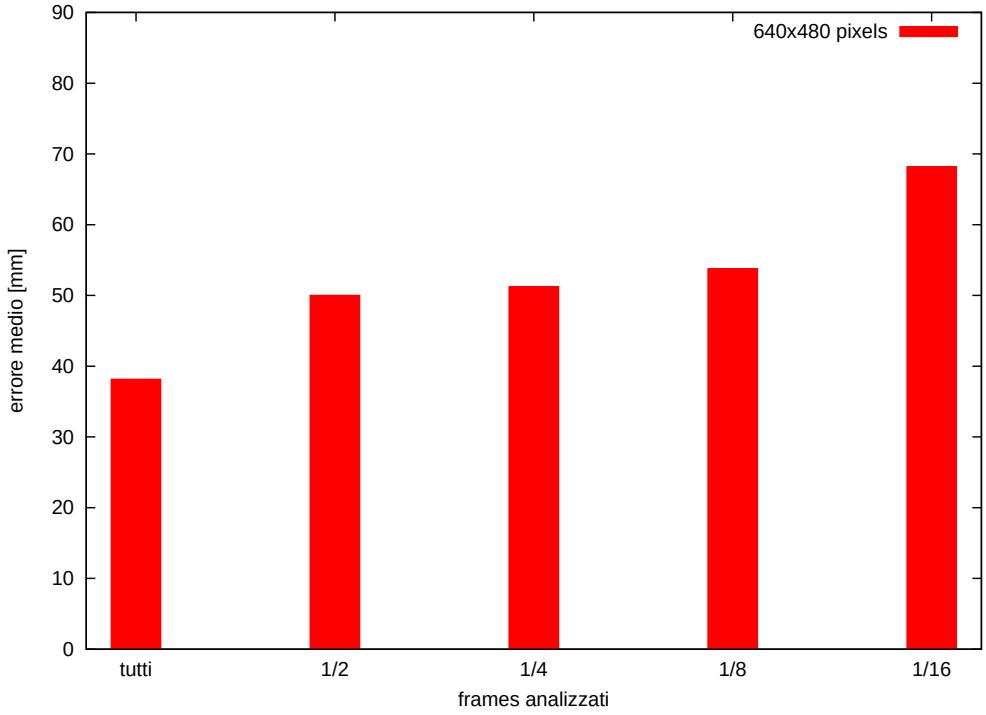


Figura 5.10: Errore ottenuto analizzando un diverso numero di frames ad una distanza di 2 metri dai markers

5.3 Test 3

Aumentando ulteriormente la distanza dalla quale vengono individuati i markers, cioè a circa 3 metri, nell'ambiente precedentemente descritto, la rappresentazione ad albero del marker field è quella di figura 5.11 nella pagina successiva ottenuta dopo l'applicazione dell'algoritmo di Dijkstra.

L'errore ottenuto nella creazione del marker field è mostrato nella figura 5.12 nella pagina seguente.

5.4 Test 4

I prossimi test sono stati eseguiti nello stesso ambiente descritto precedentemente ma illuminato con luce naturale e quindi sono presenti, all'interno del marker field, delle zone d'ombra.

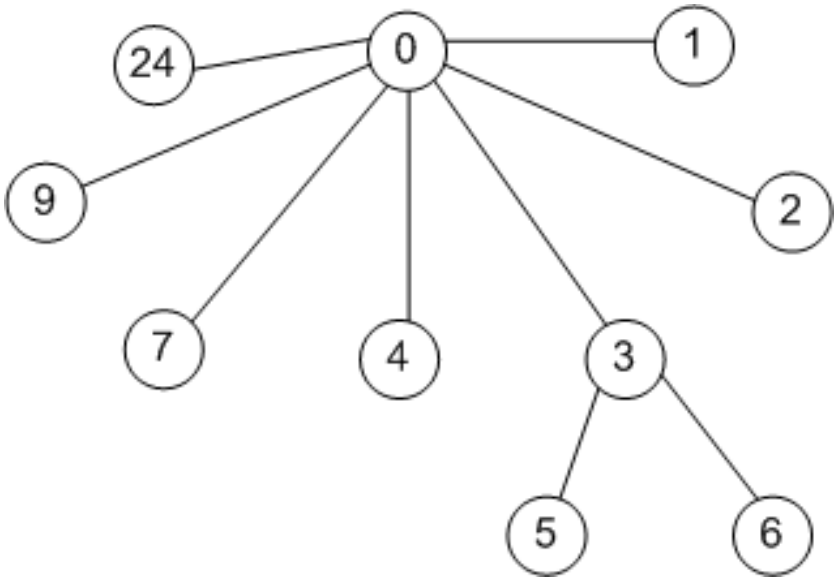


Figura 5.11: Albero ottenuto dopo l'applicazione dell'algoritmo di Dijkstra

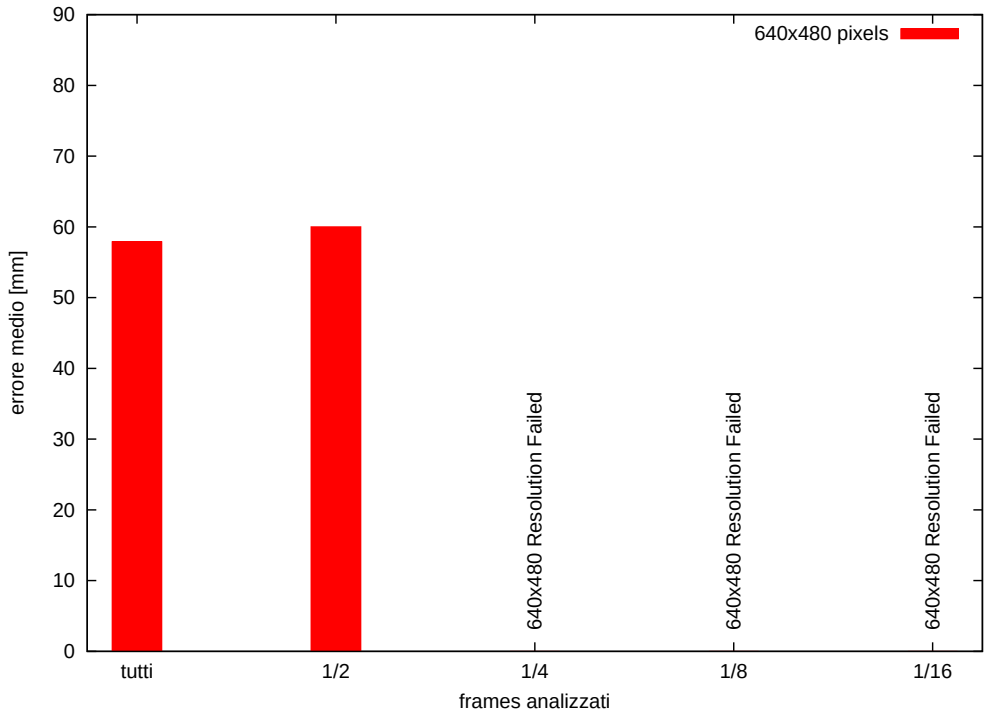


Figura 5.12: Errore medio ottenuto in un ambiente illuminato artificialmente ad una distanza di circa 3 metri dai markers

Inizialmente sono stati individuate coppie di markers come descritto nel Test 1 ed anche in questo caso, dopo l'esecuzione dell'algoritmo di Dijkstra, otteniamo la rappresentazione ad albero del marker field di figura 5.4 nella pagina 89: in questo caso l'errore medio, sia ad una risoluzione di 640x480 pixels che di 320x240 pixels, è maggiore rispetto al Test 1. In un ambiente scarsamente illuminato non si ha un netto contrasto tra il marker e lo sfondo su cui è posizionato: quando l'algoritmo di marker detection applica la threshold per individuare i markers all'interno dell'immagine acquisita, il contorno individuato può essere qualche pixels in più poiché, se il marker si trova in una posizione poco illuminata, zone intorno al marker possono essere convertite in pixels neri dalla threshold. Questo porta ad una stima non corretta della pose estimation.

Nella figura 5.13 è mostrato l'errore medio ottenuto ad una risoluzione di 640x480 pixels e 320x240 pixels

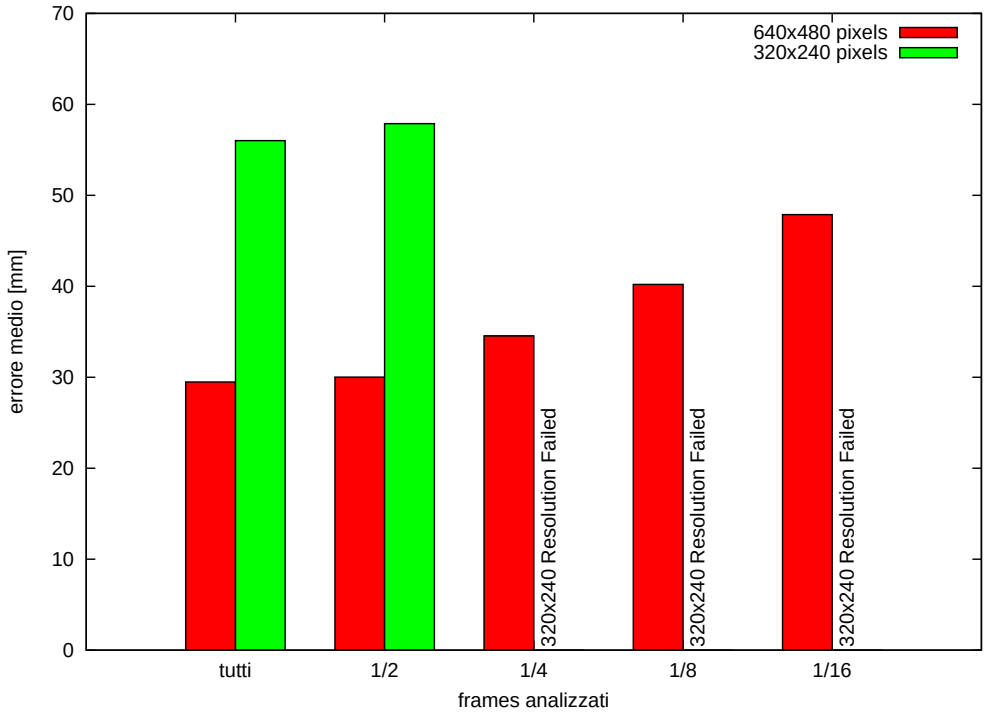


Figura 5.13: Confronto tra l'errore medio ottenuto considerando il video ad una risoluzione di 640x480 pixels e 320x240 pixels

5.5 Test 5

In questo test la distanza dalla quale vengono inquadrati i markers è di circa 2 metri e l'ambiente è illuminato con luce naturale. La rappresentazione ad albero del marker field, dopo l'esecuzione dell'algoritmo di Dijkstra, è quella ottenuta nel Test 2 di figura 5.9 nella pagina 95.

Anche in questo caso l'errore medio ottenuto è maggiore rispetto al caso in cui il marker field era illuminato uniformemente con luce artificiale a causa del motivo descritto precedentemente.

Nella figura 5.14 è mostrato l'errore medio ottenuto. Ad una risoluzione di 320x240 pixels non è stato possibile ottenere nessun marker field in quanto il filtro eliminava tutte le matrici di trasformazioni.

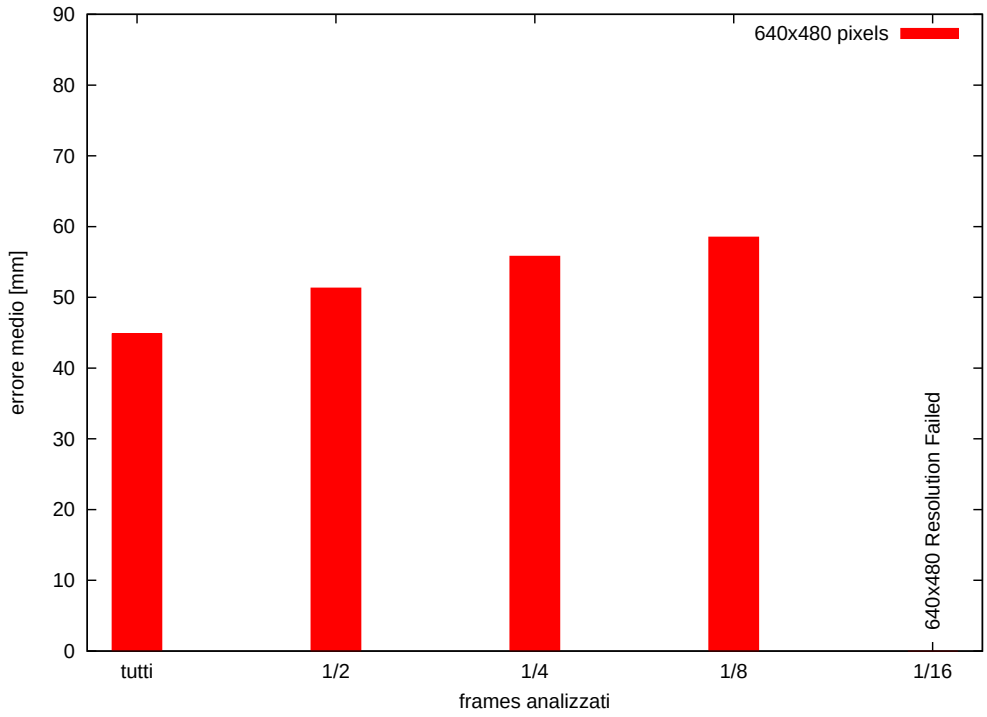


Figura 5.14: Errore ottenuto analizzando un diverso numero di frames ad una distanza di 2 metri dai markers senza luce

5.6 Test 6

Aumentando ulteriormente la distanza dalla quale vengono individuati i markers, cioè a circa 3 metri, nell'ambiente precedentemente descritto, il problema della dimensione dei markers e delle zone di ombra porta alla non creazione del markerfield poichè alcuni markers non venivano riconosciuti e quindi l'algoritmo di Dijkstra non veniva eseguito.

5.7 Test sul campo

Con riferimento ad un caso industriale specifico, sono stati applicati dei markers ad un macchinario industriale (Figura 5.15 nella pagina successiva) con l'intento di poter sovra-imporre al modello reale il suo modello CAD tridimensionale: questo può risultare utile nel caso in cui si debba effettuare manutenzione e riparazione del macchinario.

In un tale ambiente operativo un singolo marker non è sufficiente: affinchè possa essere visualizzato costantemente il modello virtuale, il marker deve sempre rientrare nel campo di vista dell'utente (più specificatamente nel campo di vista della videocamera). Si rende quindi necessario l'utilizzo multiplo di markers in modo che l'utente, spostandosi intorno al macchinario, sia sempre in grado di vedere almeno un marker in modo che il modello virtuale possa essere visualizzato.

L'algoritmo di creazione automatica del marker field si rende quindi uno strumento utile per la calibrazione dei markers, risparmiando così all'utilizzatore la misurazione manuale della distanza relativa tra i vari markers, operazione dispendiosa dal punto di vista temporale e molto imprecisa nel caso in cui i markers siano posti su piani diversi e con angolazioni arbitrarie.

La prima fase è consistita nell'applicare al macchinario quattordici markers di varie dimensioni posti su piani differenti (figura 5.16 nella pagina 102).

Successivamente è stato eseguito l'algoritmo di calibrazione automatica tenendo in considerazione di inquadrare tutti i markers in modo che possa essere trovato un percorso che va dal base marker ad ogni markers del marker field.

Figura 5.15: Macchinario industrale su cui è stato applicato l'algoritmo di calibrazione automatica

L'utente ora, spostandosi intorno al macchinario, è in grado di visualizzare il modello CAD (figura 5.17 nella pagina 103).

Il problema critico individuato in questi ambienti è l'illuminazione: a causa delle parti metalliche si creano spesso riflessioni che comportano la non *detection* dei markers con la conseguenza di non poter applicare la calibrazione automatica e quindi la non visualizzazione dei modelli virtuali.

Una possibile soluzione sarebbe quella di utilizzare vernici non riflettenti applicate alle parti metalliche oppure l'utilizzo di marker colorati invece dei marker bianchi/neri.

(a)

(b)

Figura 5.16: Applicazione dei markers sul macchinario

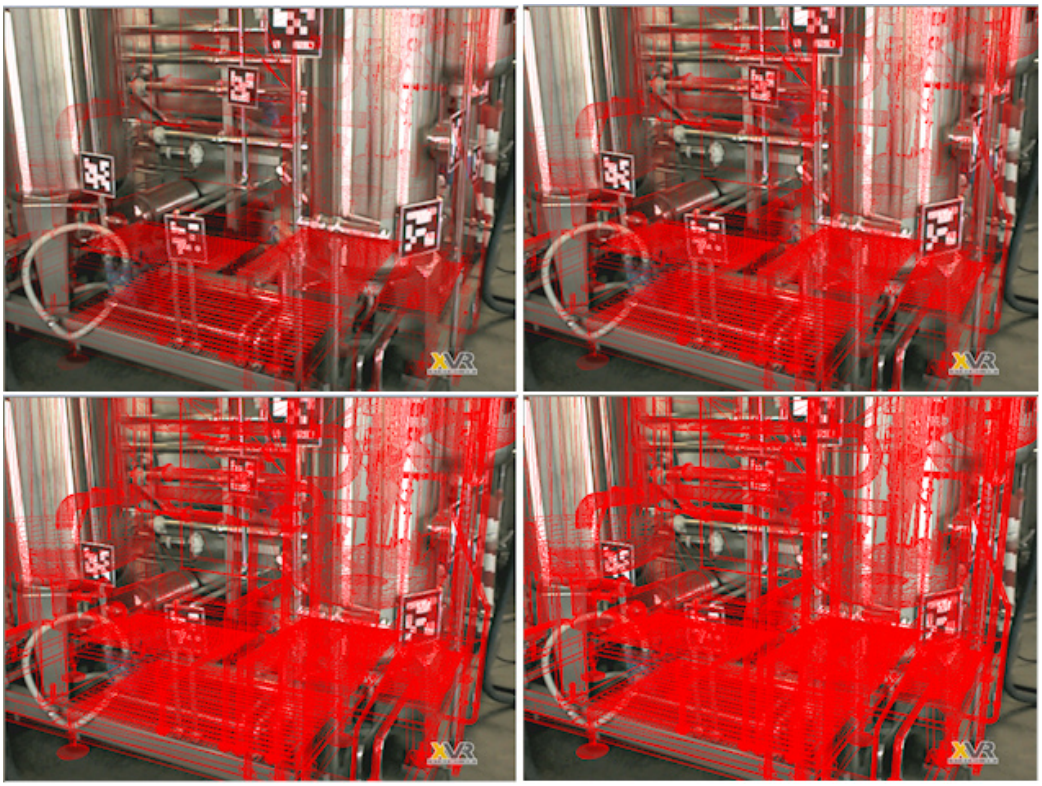


Figura 5.17: Sovraimpressione del modello cad wireframe al macchinario con diversi livelli di trasparenza

Capitolo 6

Conclusioni e sviluppi futuri

L'obiettivo della seguente tesi è stato quello di proporre e verificare un metodo per la creazione automatica del marker field per il tracking di oggetti complessi. L'uso di un marker field si rende necessario in scenari operativi dove un solo marker non costituisce un riferimento sufficiente poiché non sempre risulta essere nel campo visivo della videocamera.

La difficoltà nell'uso di un marker field deriva dal fatto che questi marker multipli devono essere messi in relazione tra loro durante una preventiva fase di calibrazione: deve essere infatti nota al calcolatore la posizione relativa tra i vari markers affinché il modello virtuale possa essere visualizzato correttamente.

Il lavoro è stato svolto organizzandolo logicamente in 5 capitoli più il presente:

- Il **Capitolo 1** ha dato una definizione di realtà aumentata (AR) e descritto le possibili applicazioni di questa tecnologia in medicina, assemblaggio, manutenzione e riparazione, architettura, archeologia, tu-

rismo, intrattenimento. Ha inoltre presentato la tipica architettura di un sistema AR.

- Il **Capitolo 2** ha esposto le nozioni alla base del tracking ottico basato su markers passivi, immagini bidimensionali bitonali che forniscono quattro punti complanari necessari al calcolo della pose estimation. Sono state inoltre analizzate ed le principali librerie basate su questo approccio (ARToolkit, ARToolkitPlus e ARTag).

- Il **Capitolo 3** ha fornito un'introduzione alle varie tecniche di creazione automatica del marker field e descritto dettagliatamente l'algoritmo implementato.

- Il **Capitolo 4** ha presentato i dettagli implementativi dei moduli sviluppati per la calibrazione automatica del marker field e per la visualizzazione dei modelli virtuali.

- Il **Capitolo 5** ha esposto i risultati dei test effettuati ed analizzato l'accuratezza nella creazione del marker field in varie condizioni operative.

6.1 Conclusioni

Il presente lavoro di tesi propone una procedura automatica per la calibrazione dei marker fields in grado di semplificare notevolmente il processo di instrumentazione di macchinari di grandi dimensioni. Il metodo di calibrazione si basa su di una applicazione iterativa dell'algoritmo di Dijkstra e funzioni di costo opportune per il calcolo del percorso a costo minimo. L'algoritmo implementato è semplice da utilizzare e permette di ottenere in breve tempo la calibrazione automatica di più markers anche in configurazioni complesse. Test sul campo su macchinari industriali veri hanno permesso di verificare l'effettiva efficacia dell'approccio.

Durante lo svolgimento della tesi sono emerse anche alcune difficoltà o elementi critici che influenzano grandemente i risultati: ad esempio, prima

che l'utente utilizzi questo algoritmo, è fondamentale che la videocamera utilizzata venga calibrata accuratamente, in quanto i parametri intrinseci della videocamera si sono rivelati di fondamentale importanza per l'accuratezza del metodo.

Test preliminari sulla libreria ARToolkitPlus hanno evidenziato che è opportuno utilizzare markers di dimensioni superiori ai 10 cm se questi devono essere inquadrati da distanze superiori ad 1 metro, altrimenti c'è la possibilità che molte volte il marker non venga riconosciuto oppure l'errore nel calcolo della pose estimation sia troppo elevato. In particolare, dai test effettuati si evidenzia che nel caso di oggetti di grandi dimensioni, quali la grande maggioranza dei macchinari industriali, per ottenere un marker field utilizzabile per la sovrapposizione di modelli CAD tridimensionali sui corrispettivi oggetti fisici, è necessario utilizzare markers con una dimensione di almeno 12 cm che devono essere inquadrati da una distanza di circa 1 metro.

E' importante che la risoluzione delle immagini acquisite sia la più elevata possibile in rapporto alle capacità hardware disponibili: questo perchè con una risoluzione più alta vengono individuati in modo più dettagliato i contorni del marker, con un conseguente calcolo più accurato della posizione della telecamera, ma a fronte di un tempo di calcolo più elevato per l'analisi dell'immagine.

Da tenere in considerazione sono anche le condizioni di illuminazione: dai test effettuati risulta che in caso di ambienti scarsamente illuminati l'accuratezza con cui viene prodotto il marker field è peggiore rispetto al caso di ambienti ben illuminati. Complementarmente pero', luci troppo intense in prossimità dei marker possono creare delle riflessioni che possono rendere impossibile un corretto processo di marker detection. Il problema può essere attenuato facendo uso per la produzione dei markers di materiali anti-riflettenti.

6.2 Sviluppi futuri

Sulla base dell'esperienza accumulata durante la presente tesi e considerando i risultati dei test effettuati e le considerazioni sopra esposte, è possi-

bile individuare tra gli sviluppi futuri più interessanti del presente lavoro i seguenti punti:

- Aggiornamento dinamico del marker field: invece di produrre un marker field statico può essere prodotto un marker field dinamico che si aggiorna in tempo reale così da tenere conto delle possibili variazioni nella configurazione dei markers e delle variazioni nelle condizioni ambientali (in primis le condizioni di illuminazione e gli effetti di riflessione della luce causati dalla specularità dei materiali). Di conseguenza anche la metrica di costo potrebbe essere variata dinamicamente al fine di ottenere un marker field più accurato. Ovviamente, a seguito di questa modifica, anche il modulo implementato per la visualizzazione di contenuti virtuali deve essere modificato in modo da poter supportare marker field dinamici.

- Scelta e test di altre metriche di costo di un collegamento tra una coppia di markers.

- Utilizzo e test di altre tipologie di markers come, ad esempio, markers circolari, markers retroriflettenti visibili solamente attraverso una videocamera ad infrarossi oppure markers che possono essere utilizzati come decorazione da interni tali da integrarsi perfettamente nell'ambiente lavorativo (Figura 6.1).

Figura 6.1: Esempio di markers utilizzati come decorazione da interni [31]

- Introduzione nel sistema di algoritmi di rendering per la visualizzazione ottimizzata dei contorni di un modello CAD (l'uso attuale della modalità wirefame non consente la migliore leggibilità del risultato).

  Con riferimento alla figura 5.17 a pagina 102 risulta infatti chiaro che il macchinario reale, con la sovrapposizione del modello CAD in modalità wireframe, non risulta chiaramente distinguibile [32].

Appendice A

La grafica tridimensionale e OpenGL

La grafica tridimensionale consiste nel rappresentare un ambiente a tre dimensioni (3D) su un dispositivo che, per sua natura, è 2D. Per effettuare tale rappresentazione si simula il comportamento di una macchina fotografica che osserva lo scenario da una certa posizione (*view point*) e con un certo orientamento.

La trasformazione della scena in una immagine bidimensionale (*frame*) è un processo che viene definito *pipeline grafica* che consiste in una serie di fasi consecutive. La prima fase è la fase di *trasformazione*, divisa a sua volta in tre sottofasi. La sottofase di *Modelview transformation* gestisce la trasformazione delle coordinate dei vertici degli oggetti: riceve in input le coordinate solidali all'oggetto e le trasforma in coordinate solidali con la posizione dell'osservatore (*world coordinate system*).

La sottofase successiva è quella di *Projection transformation*, che si occupa di definire il volume di vista (*view volume*). Possono essere scelte due tipi di proiezioni: la proiezione prospettica o la proiezione ortogonale. Nel primo caso, la distanza tra il centro della proiezione ed il piano di vista è finita, mentre nel secondo caso il centro della proiezione va all'infinito, tanto che è improprio parlare di centro della proiezione ma piuttosto di *direzione* di proiezione. Definito il volume di vista si procede ad eliminare dalla scena le

superfici che non rientrano all'interno di esso (*clipping*) e quelle che appaiono nascoste all'osservatore (*culling*), a causa del proprio orientamento.

L'ultima sottofase è la *Viewport Transformation* che consiste nel convertite le coordinate 3D dei vertici degli oggetti in coordinate 2D che indicano la posizione del vertice all'interno della finestra di visualizzazione nello schermo (*window coordinates*). Oltre a tali coordinate, tuttavia, ne viene memorizzata una terza che identifica la distanza dell'oggetto dall'osservatore: viene utilizzata per capire, nel caso in cui alcuni oggetti abbiano le stesse coordinate, quali di essi debba risultare coperto dall'altro. La terza coordinata di ogni vertice viene memorizzata in un apposito buffer, denominato *z-buffer*.

La fase successiva a quella di trasformazione è quella di *illuminazione* che ha lo scopo di determinare il colore di ogni pixel dell'immagine, facendo interagire le proprietà delle luci presenti nella scena con le proprietà delle superfici degli oggetti e della radiazione luminosa incidente.

L'ultima fase della pipeline è quella di *rasterizzazione*, che consiste nel produrre ogni pixel del frame buffer (buffer della scheda video dove viene memorizzata l'immagine che deve essere rappresentata sullo schermo), avendo a disposizione le posizioni, all'interno di tale frame, dei vertici di ogni primitiva visibile.

Affinchè un'applicazione possa accedere alla pipeline grafica è necessaria una libreria che si occupi di gestire l'hardware sottostante, sollevando così l'applicazione dal colloquio diretto con i driver del dispositivo grafico.

Nell'ambito di visualizzazione grafica esistono diverse librerie: alcune sono legate ad uno specifico hardware e sono configurate per funzionare solamente con esso. E' chiaro quindi che il codice scritto, utilizzando queste librerie, non è adattabile ad altre piataforme.

Altre librerie invece sono indipendenti dall'hardware ma legate ad un particolare sistema operativo: un esempio è dato dalle *DirectX* sviluppate da Microsoft, le quali non sono portabili su piattaforme differenti da Windows, come ad esempio Unix.

OpenGL [13] [7] è invece una libreria grafica che è indipendente sia dall'hardware che dalla piattaforma: è stata introdotta dalla Silicon Graphics e ne esiste una implementazione per tutti i maggiori sistemi operativi. Le spe-

cifiche OpenGL sono state decise e periodicamente riesaminate dall'*OpenGL Architecture Review Board* (ARB) , un comitato indipendente di produttori hardware e software tra i cui membri figurano IBM, Microsoft, Intel e Silicon Graphics.

OpenGL offre una serie di funzioni dedicate alla grafica 3D, ma è in grado anche di trattare il 2D, che viene considerato come un suo sottoinsieme. Non fornisce nessuna funzione di gestione delle finestra di visualizzazione per mantenere l'indipendenza dalla piattaforma usata: per questo motivo deve fare ricorso a funzioni dello specifico sistema operativo oppure utilizzare librerie *platfmor independent* come ad esempio le *GLUT*.

Le funzioni messe a disposizione da OpenGL permettono di disegnare semplici figure geometriche come punti, linee e triangoli, oppure poligoni convessi composti da un numero arbitrario di lati o curve espresse analiticamente. Le trasformazioni geometriche necessarie per la visualizzazione sono specificate mediante due matrici: la *modelview matrix* e la *projection matrix*. La prima permette di posizionare la videocamera ed i modelli 3D nell'ambiente: nelle applicazioni di realtà aumentata questa matrice viene ricavata attraverso l'algoritmo di *pose estimation* che ricava la posizione della videocamera reale rispetto al marker e quindi permette di settare la posizione della videocamera virtuale nell'ambiente 3D del contesto OpenGL. In OpenGL la modelview matrix viene caricata mediante i seguenti comandi: *glMatrixMode(GL_MODELVIEW)* e *glLoadMatrix(matrice[4X4])*.

La projection matrix definisce il volume di vista (*view volume*). Si possono utilizzare due tipi di proiezione: prospettica o ortografica. I punti o le parti di poligono che risultano fuori dal volume di vista vengono rimossi e quindi non disegnati nella finestra grafica. La proiezione prospettica viene definita attraverso la chiamata alla funzione *glFrustum()* (Figura A.1 nella pagina seguente) mentre quella ortografica mediante *glOrtho()* (Figura A.2 nella pagina successiva), dopo essere passati in modalità projection matrix attraverso la funzione *glMatrixMode(GL_PROJECTION)*.

Come detto precedentemente, ARToolkit setta la matrice di proiezione direttamente caricando la matrice ricavata dalla calibrazione della videocamera: in questo modo il volume di vista della videocamera virtuale coincide

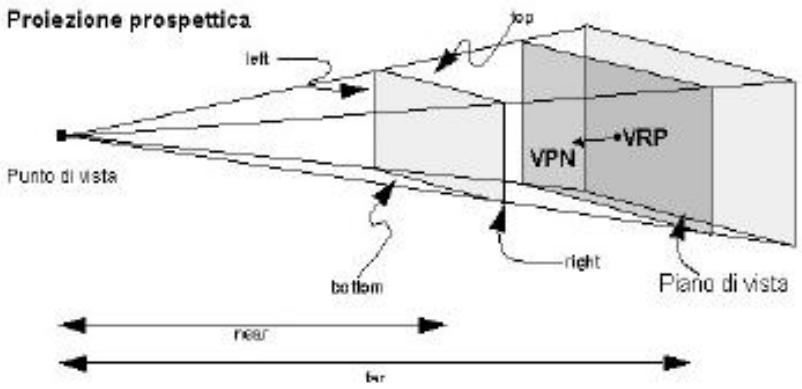


Figura A.1: Volume di vista definito con glFrutum

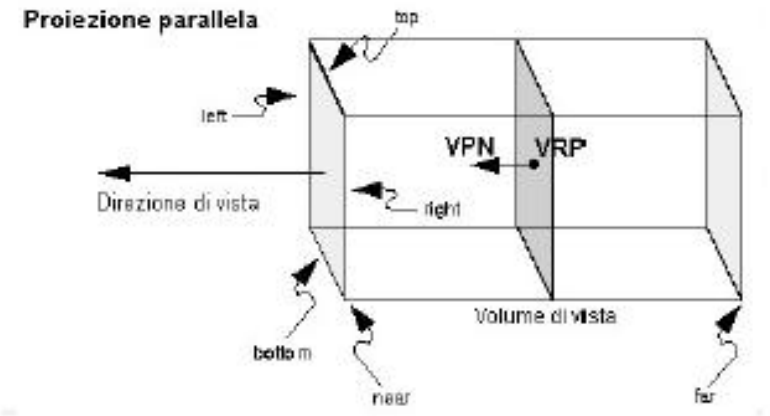


Figura A.2: Volume di vista definito con glOrtho

con quello della videocamera reale.Una volta eliminate le primitive che non rientrano nel volume di vista, alle rimanenti viene applicata la trasformazione di *viewport*, con la quale si ottengono le coordinate in pixel per la visualizzazione sullo schermo.

Per specificare l'illuminazione della scena, OpenGL prevede tre tipi di sorgenti luminose: *puntiformi, direzionali* e *spot*. La luce puntiforme proviene da una sorgente posta in un punto dello spazio ed emette raggi in ogni direzione. La luce direzionale proviene da una sorgente che si suppone posta all'infinito e con una direzione assegnata i cui raggi emessi sono, così, tutti

paralleli tra di loro. La luce spot è analoga a quella puntiforme, solo che il campo di emissione è ristretto ad un cono del quale è specificata l'apertura angolare.

Appendice B

Breve descrizione delle techiche di Pose Estimation

Matematicamente, la creazione di una immagine può essere definita come una prozieziope dallo spazio 3D al piano dell'immagine come nella figura B.1.

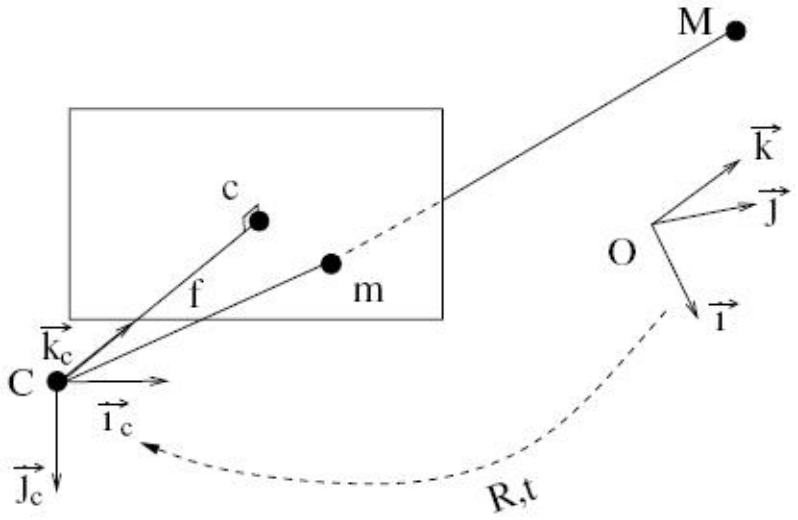


Figura B.1: Modello di proizione prospettica

Le coordinate di un punto $\mathbf{M}$ nello spazio 3D, $\mathbf{M} = \Big[X, Y, Z\Big]^T$, espresse in un sistema di coordinate Euclidee e il corrispondente punto 2D $\mathbf{m} = \Big[u, v\Big]^T$ nell'immagine sono legate dalla relazione

$$s\tilde{\mathbf{m}} = \mathbf{P}\tilde{\mathbf{M}}$$

dove s è un fattore di scala, $\tilde{\mathbf{m}} = \Big[u, v, 1\Big]^T$ e $\tilde{\mathbf{M}} = \Big[X, Y, Z, 1\Big]^T$ sono le coordinate omogenee del punto $\mathbf{m}$ e $\mathbf{M}$, e $\mathbf{P}$ è una matrice di proiezione 3x4. $\mathbf{P}$ può essere scomposta come:

$$\mathbf{P} = \mathbf{K}\Big[\mathbf{R}|\mathbf{t}\Big]$$

dove:

- $\mathbf{K}$ è la matrice 3x3 di calibrazione della videocamera che dipende da parametri interni come la lunghezza focale, come già visto nella sezione 2.2
- $\Big[\mathbf{R}|\mathbf{t}\Big]$ (definita anche come *external parameter matrix*)è una matrice 3x4 che fa corrispondere un punto nel sistema di coordinate Euclidee 3D al sistema di coordinate della videocamera: $\mathbf{R}$ rappresenta la matrice 3x3 di rotazione, $\mathbf{t}$ rappresenta la traslazione.

Ad oggi, esistono diverse tecniche per stimare la posizione della videocamera nell'ambiente, osservando le corrispondenze di punti 3D nel sistema di coordinate euclidee e la loro proiezione nel piano immagine della videocamera. La stima degli external parameters è spesso riferita come un problema di *pose estimation.*

Assumiamo di avere n corrispondenze tra i punti $\mathbf{M}_i$ nello spazio Euclideo e le loro proiezioni $\mathbf{m}_i$ sul piano immagine. Quando i parametri interni sono noti (ottenuti attraverso il processo di calibrazione della videocamera), note $n = 3$ corrispondenze $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ si hanno quattro soluzioni posibili. Quando invece sono note $n = 4$ (come nel caso dei marker quadrati) o $n = 5$ corrispondenze, ci sono almeno due soluzioni possibili, ma quando i punti sono coplanari e non ci sono triplette di punti collineari, la soluzione è unica per $n \geq 4$.

Un metodo per ottenere la pose estimation è il *Perspective-n-Point* (PnP). Questo metodo, quando abbiamo $n = 3$ corrispondenze (in questo caso, questo metodo viene definito come *perspective-3-point*(P3P)), nella maggior par-

te dei casi fornisce quattro soluzioni reali. Per quattro o più punti, in generale la soluzione è unica, indipendentemente se i punti sono coplanari o no, l'unico vincolo è che non siano collineari. In casi sfortunati ci possono essere però infinite soluzioni, indipendentemente da quanti punti di corrispondenza sono forniti.

Sono stati proposti altri approcci oltre al P3P, che cercano distimare la distanza $x_i = \|\mathbf{CM}_i\|$ tra il centro $\mathbf{C}$ della videocamera e i punti $\mathbf{M}_i$, dai vincoli dati dal triangolo $\mathbf{CM}_i\mathbf{M}_j$. Una volta noti gli x_i, gli $\mathbf{M}_i$ vengono espressi nel sistema di coordinate della videocamera, cioè come $\mathbf{M}_i^c$. Quindi $\Big[\mathbf{R}|\mathbf{t}\Big]$ rappresenta lo spostamento che allinea i punti $\mathbf{M}_i$ sui punti $\mathbf{M}_i^c$ e può essere trovata utilizzando i quaternioni o il *singular value decomposition* (SVD).

Il POSIT (*Pose from Orthography and Scaling with Iteration*) è un altro metodo, molto comune, per la pose estimation per $n \geq 4$. Inizialmente calcola una soluzione approssimata assumendo una *scaled orthographic projection* (approssimazione al primo ordine di una proiezione prospettica) per il modello della videocamera, che significa che la stima iniziale della posizione e dell'orientamento della videocamera, può essere trovata risolvendo un sistema lineare. Vengono poi effettuate successive iterazioni per ottenere una convergenza. Questo metodo è molto facile da implementare, ma è sensibile al rumore e non può essere applicato quando i punti sono complanari.

La posizione della videocamera può essere stimata anche da una struttura planare, quando i parametri interni sono noti. Questa proprietà è spesso utilizzata nel 3D tracking perché le proiezioni di strutture planari sono facili da ricavare nelle immagini.

La relazione tra un piano nello spazio 3D e la sua proiezione nel piano immagine della videocamera, può essere rappresentata mediante una matrice omogenea 3x3, chiamata matrice omografica. Consideriamo il piano $Z = 0$. L'espressione dell'omografia $\mathbf{H}$ che mappa un punto $\mathbf{M} = (X, Y, 0)^T$ su questo piano e il suo corrispondente punto 2D $\mathbf{m}$ mediante una proiezione prospettica $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ può essere ricavata in questo modo:

$$\tilde{\mathbf{m}} = \mathbf{P}\tilde{\mathbf{M}} = \mathbf{K}(\mathbf{R}^1\mathbf{R}^2\mathbf{R}^3\mathbf{t}) \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \mathbf{K}(\mathbf{R}^1\mathbf{R}^2\mathbf{t}) \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \mathbf{H} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

dove $\mathbf{R}^1$, $\mathbf{R}^2$ e $\mathbf{R}^3$ rappresentano, rispettivamente, la prima,seconda e terza colonna della matrice di rotazione $\mathbf{R}$. Viceversa, una volta note $\mathbf{H}$ e $\mathbf{H}$, la posizione dela videocamera può essere ricavata. La matrice $\mathbf{H}$ può essere stimata da 4 punti di corrispondenza $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ utilizzando ad esempio l'algoritmo *Direct Linear Transformation* (DLT). Poichè $(H)^t_w = \mathbf{K}(\mathbf{R}^1\mathbf{R}^2\mathbf{t})$, il vettore di traslazione $\mathbf{t}$ e le prime due colonne della matrice di rotazione $\mathbf{R}$ della posizione della videocamera, possono essere ricavate dal prodotto $\mathbf{K}^{-1}\mathbf{H}^t_w$. L'ultima colonna $\mathbf{R}^3$ è data dal prodotto $\mathbf{R}^1$x$\mathbf{R}^2$.

Bibliografia

[1] ARTag. "http://www.artag.net/.

[2] ARToolkit. http://www.hitl.washington.edu/artoolkit/.

[3] ARToolkitPlus. http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php.

[4] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[5] Gregory Baratoff and Scott Blanksteen. Tracking devices. http://www.hitl.washington.edu/scivw/EVE/I.D.1.b.TrackingDevices.html.

[6] Gregory Baratoff, Alexander Neubeck, and Holger Regenbrecht. Interactive multi-marker calibration for augmented reality applications. In *ISMAR '02: Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 107, Washington, DC, USA, 2002. IEEE Computer Society.

[7] OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, and Jackie Neider. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2.1 (6th Edition)*. Addison-Wesley Professional, 2007.

[8] Stephent Cawood and Mark Fiala. *Augmented Reality: A Practical Guide*. The Pragmatic Programmers, 2008.

[9] Kotake Daisuke, Uchiyama Shinji, and Yamamoto Hiroyuki. A marker calibration method utilizing a priori knowledge on marker arrangement.

In *ISMAR '04: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 89–98, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Fakespace. http://www.inition.co.uk/inition/product.php?URL_=product_hmd_fakespace_boom3&SubCatID_=34.

[11] Fastrack. http://www.polhemus.com/?page=Motion_Fastrak.

[12] Mark Fiala. *ARTag, An Improved Marker System Based on ARToolkit.* PhD thesis, National Research Council Canada, 2004.

[13] OpenGL The Industry's Foundation for High Performance Graphics. http://www.opengl.org.

[14] Nicoletta De Francesco. *Modulo di Algoritmi e Strutture Dati.* Corso di Ingegneria Informatica, Pisa.

[15] Kato Hirokazu and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmentedreality conferencing system. *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, 1999.

[16] Tobias Höllerer and Steven Feine. *Mobile Augmented Reality.* Francis Books Ltd, 2004.

[17] Intersense. http://www.intersense.com/.

[18] Grudun Klinker. http://campar.in.tum.de/Chair/TeachingWs08AugmentedReality.

[19] Vincent Lepetit and Pascal Fua. *Monocular model-based 3D tracking of rigid objects*, volume 1. Foundations and Trends in Computer Graphis and Vision, 2005.

[20] Kalkusch M., Lidy T., Knapp N., and Schmalstieg D. Structured visual markers for indoor pathfinding. In *Augmented Reality Toolkit, The First IEEE International Workshop*, page 8. IEEE Computer Society, 2002.

[21] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. Augmented reality: A class of displays on reality-virtuality continuum. In *Telemanipulation an Telepresence technologies*, pages 282–292, 1994.

[22] Alex Mulder. Human movement tracking technology. Technical report, School of Kinesiology, Simon Fraser University, 1994.

[23] Joseph Newman, Friedrich Fraundorfer, Gerhard Schall, and Dieter Schmalstieg. Construction and maintenance of augmented reality environments using a mixture of autonomous and manual surveying techniques. *Proc. 7th Conference on Optical 3-D Measurement Techniques*, 2005.

[24] OSGART. "http://www.artoolworks.com/community/osgart/.

[25] Azuma R., Baillot Y., Behringer R., Feiner S., Julier S., and MacIntyre B. Recent advances in augmented reality. In *Computer Graphics and Applications*, pages 34–47, 2001.

[26] Holloway R. and Lastra A. Virtual environments: A survey of the technology. Technical report, Chapel Hill, 1993.

[27] Dirk Reiners, Didier Stricker, Gudrun Klinker, and Stefan Můller. Augmented reality for construction tasks: Doorlock assembly. *In Proc.IEEE International Workshop on Augmented Reality*, 2008.

[28] Jun Rekimoto. Matrix: a realtime object identification and registration methodfor augmented reality. *Computer Human Interaction*, 1998.

[29] J.P. Rolland, L.D. Davis, and Y. Baillot. A survey of tracking technology for virtual environments. *Augmented Reality and Wearable Computers*, 2000.

[30] Siltanen S., Hakkarainen M., and Honkamaa P. Automatic marker field calibration. In *in Proc. Virtual Reality International Conference (VRIC)*, pages 261–267, April 2007.

[31] Shigeru Saito, Atsushi Hiyama, Tomohiro Tanikawa, and Michitaka Hirose. Indoor marker-based localization using coded seamless pattern for interior decoration. *Virtual Reality Conference, IEEE*, 0:67–74, 2007.

[32] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, 1990.

[33] Daniel Wagner and Dieter Schmalsteig. Artoolkitplus for pose tracking on mobile devices. *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, 2007.

[34] Greg Welch and Eric Foxlin. Motion tracking: no silver bullet, but a respectable arsenal. *Computer Graphics and Applications*, 2002.

[35] Charler Woodward. http://virtual.vtt.fi/virtual/proj2/multimedia/AR_MR_brief_2009-04.pdf, April 2009.

[36] Uematsu Yuko and Saito Hideo. Ar registration by merging multiple planar markers at arbitrary positions and poses via projective space. In *ICAT '05: Proceedings of the 2005 international conference on Augmented tele-existence*, pages 48–55, New York, NY, USA, 2005. ACM.

[37] Xiang Zhang, Stephan Fronz, and Nassir Navab. Visual marker detection and decoding in ar systems: A comparative study. In *ISMAR '02: Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 97, Washington, DC, USA, 2002. IEEE Computer Society.

[38] S. Zlatanova. *Augmented Reality Technology*, 2002.