UNIVERSITÀ DEGLI STUDI DI PARMA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

INTEGRAZIONE DI UN SISTEMA DI MOTION CAPTURE IN UN AMBIENTE VIRTUALE INTERATTIVO

INTEGRATION OF A MOTION CAPTURE SYSTEM IN AN INTERACTIVE VIRTUAL ENVIRONMENT

Relatore
Chiar.mo Prof. STEFANO CASELLI

Correlatori
Dott. Ing. VINCENZO MICELLI
Dott. Ing. JACOPO ALEOTTI

Tesi di Laurea di:
VOTTARI FRANCESCO

Anno Accademico 2009/10

UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

INTEGRAZIONE DI UN SISTEMA DI MOTION CAPTURE IN UN AMBIENTE VIRTUALE INTERATTIVO

INTEGRATION OF A MOTION CAPTURE SYSTEM IN AN INTERACTIVE VIRTUAL ENVIRONMENT

Relatore
Chiar.mo Prof. STEFANO CASELLI

Correlatori
Dott. Ing. VINCENZO MICELLI
Dott. Ing. JACOPO ALEOTTI

Tesi di Laurea di:
VOTTARI FRANCESCO

23 Marzo 2011

Ai miei nonni
sempre presenti

Il mio primo ringraziamento va al Prof. Stefano Caselli che con attenzione ha seguito me e la mia ragazza dal nostro arrivo a Parma fino alla stesura della tesi, fornendoci ottimi consigli ed incoraggiamento, in un momento in cui la nostra vita stava cambiando bruscamente.
Ringrazio molto anche il Dott. Jacopo Aleotti ed il Dott. Vincenzo Micelli, che si sono sempre dimostrati cortesi e disponibili per darmi delucidazioni ogni volta ne avevo bisogno.
Non può mancare un enorme ringraziamento alla mia ragazza Vanessa Manni che ha condiviso con me ogni ora di studio, fin dai precorsi e che ha affrontato con me ogni difficoltà incontrata negli ultimi anni, sia dentro che fuori l'università.
I miei ringraziamenti vanno anche a tutta la mia famiglia, in particolare i miei genitori Giuseppe e Maria, mio fratello Mauro e mia nonna Mafalda, che mi hanno permesso di continuare gli studi e mi hanno sostenuto anche quando le cose sembravano andar male.
Ringrazio molto anche tutta la famiglia, la nonna e gli zii della mia ragazza, che ci hanno sostenuto dopo alcune brutte esperienze vissute in questi anni di studio; in particolare ringrazio Gerardo e Luisa che sono sempre stati presenti per aiutarci e guidarci con i loro ottimi consigli.
Voglio ringraziare tutti i miei amici, di cui molti purtroppo sono oramai lontani, con i quali ho passato splendide giornate ed esperienze che non potrò dimenticare.
Un ringraziamento particolare va ad Andrea, Aurora ed Enrico che nonostante i loro impegni in diverse occasioni sono corsi da me per testare il lavoro svolto per questa tesi.

Un giorno le macchine riusciranno a risolvere tutti i problemi,
ma mai nessuna di esse potrà porne uno.
Albert Einstein

Indice

Introduzione

Negli ultimi anni molti studi sono stati indirizzati verso l'interazione uomo macchina (1), per mezzo di svariati dispositivi.

Questi dispositivi hanno sempre avuto un fattore in comune: devono supportare il compito dell'utente, il che porta a considerare una caratteristica molto importante, ovvero l'usabilità. Se il sistema costringe l'utente ad accettare un modello di lavoro inaccettabile allora il sistema non è utilizzabile. Un prodotto per ottenere successo deve godere di tre proprietà:

- **Utile,** deve fare quello che viene richiesto come riprodurre musica o stampare un documento.
- **Usabile,** deve consentire di farlo in modo naturale, senza pericolo di errore e così via.
- **Usato,** deve rendere le persone desiderose di usarlo, quindi essere interessante, divertente e piacevole.

L' evoluzione dei sistemi per l'interazione uomo-macchina ha influenzato il design dei dispositivi ed ha permesso la diffusione e l'utilizzo dei sistemi più noti come mouse e tastiera, passando poi a periferiche più sofisticate ed eleganti come il touch screen ed il Wii mote. Esistono poi sistemi che non sono distribuiti su larga scala, in quanto molto costosi ed ingombranti, come ad esempio i dispositivi (tute, telecamere e software di analisi video) di motion capture (2).

In particolare quest'ultimo è una tecnica di animazione digitale che permette di applicare a personaggi virtuali i movimenti di persone riprese in tempo reale e riportarli sullo schermo per mezzo di sensori posti sui punti di giuntura delle ossa e di contrazione dei muscoli. Le tecniche d'analisi di movimento (3) per mezzo del motion capture (anche detto mocap) sono svariate e sfruttano l'uso di sistemi differenti come: ottici, magnetici, meccanici ed acustici. Questi sistemi molto

spesso sono sinonimo di costi elevati. Ad esempio un ottimo sistema di motion capture è il sistema Vicon[1], adottato tra l'altro all'ospedale pediatrico Bambino Gesù di Roma al costo di circa trecentomila euro. I sistemi di motion capture più economici possono comunque richiedere un budget che si aggira tra i tremila e diecimila euro.

Tra i diversi motivi che hanno portato all'introduzione di questi sistemi, c'è il desiderio di raggiungere una maggiore naturalezza tra le interazioni che possono coinvolgere l'uomo e le macchine. Da questa ambizione sono nate le librerie per la Natural Interaction (4), che permettono l'interazione uomo-macchina basata sui sensi umani come udito, vista e movimento, rendendo del tutto obsoleti dispositivi come tastiere, mouse e telecomandi. Queste librerie di interazione naturale si basano sull'uso di interfacce naturali, cioè dispositivi capaci di catturare i movimenti del corpo ed i suoni; un esempio di dispositivo può essere la periferica di gioco Kinect di casa Microsoft. I campi di utilizzo di sistemi di interazione naturale possono essere svariati, come: videogiochi, medicina, cinema, robotica, industrie, ricerca.

Con la tecnologia a disponibile è possibile realizzare un sistema estremamente economico e poco ingombrante di motion capture, che potrà essere destinato allo studio delle traiettorie umane o alla computer graphics (5). Inoltre l'uso e l'evoluzione di sistemi basati sull'interazione naturale consentirebbe di svolgere facilmente operazioni scomode o del tutto impossibili alle persone disabili, offrirebbero la possibilità di rivoluzionare le tecnologie utilizzate sul lavoro e negli ambienti domestici. Ad esempio sarebbe comodo fare zapping tra i programmi tv con un semplice gesto della mano, o navigare su internet stando seduti sul divano senza dover usare mouse o tastiere, oppure per un medico proiettare i diversi documenti della cartella clinica di un paziente su uno schermo della sala operatoria con un semplice comando vocale. Questi esempi mostrano solo tre dei molti campi di ricerca che si sono ampliati negli ultimi mesi dopo l'uscita del sensore Kinect.

Per ora i tre principali modi di interazione uomo-macchina con l'uso delle librerie per la "Natural Interaction" sono: l'invio d'istruzioni tramite comandi vocali; il riconoscimento di gesti delle mani, che permettono di controllare i dispositivi elettronici ed interagire con varie applicazioni; il Body Motion Tracking, ovvero il monitoraggio e l'analisi del movimento del corpo. Quest'ultimo è utile per fini di

[1] Il sistema Vicon è uno strumento che permette l'analisi tridimensionale del movimento di un paziente o di un atleta, con applicazioni sia in ambito medico che sportivo. Sito web per maggiori approfondimenti http://www.vicon.com

gioco oppure per l'analisi di un ambiente per ricerca e sicurezza, ad esempio per acquisire dati per eventuali simulazioni di folle.
Questa tesi ha avuto come obbiettivo lo studio di un dispositivo per l'interfaccia naturale e di alcune librerie che ne permettono l'uso, al fine di progettare e sviluppare un'interfaccia utente a basso costo basata sul movimento delle mani. Il dispositivo utilizzato è il sensore Kinect accompagnato dalle librerie OpenNI e NITE.
Dopo una prima parte di analisi ed introduzione all'uso delle interfacce naturali si è realizzato un primo programma di prova, un "Hello World", al fine di prendere pratica con gli strumenti studiati. In un secondo momento per mostrare una maggiore interazione con il dispositivo è stata creata una classe per riconoscere un alfabeto di gesti e tenere costantemente traccia del movimento di una mano. I dati elaborati da questa classe sono stati successivamente inviati per mezzo di un'architettura client-server ad un motore grafico che ha permesso di simulare l'interazione all'interno di un ambiente virtuale.

La tesi è strutturata nel seguente modo:

- Nel primo capitolo viene presentata la diffusione di interfacce naturali a basso costo grazie al lancio sul mercato del sensore Kinect. Viene analizzato il suo hardware e vengono descritte le principali funzionalità di cui è dotata questa periferica.

- Nel secondo capitolo sono elencati e discussi gli strumenti utilizzati per realizzare il progetto di tesi, in particolare sono descritti i criteri di scelta dei driver attualmente disponibili per PC fornendo una guida all'installazione. Viene inoltre discusso come utilizzare le librerie per interfacciarsi con il sensore Kinect e sono fornite indicazioni utili alla realizzazione di progetti futuri. Per ultima cosa viene presentato anche il motore grafico utilizzato descrivendone le caratteristiche.

- Nel terzo capitolo viene esposto il progetto realizzato, introducendo il sistema per navigare in ambienti 3D. È descritta l'architettura client server e come avviene l'interpretazione dei dati ottenuti dal sensore per la navigazione e l'interazione con oggetti in un ambiente virtuale.

- Nel quarto capitolo viene descritta la fase di collaudo e vengono presentati i test svolti per analizzare l'effettiva naturalezza dell'uso di questo sistema. Il test è stato eseguito facendo utilizzare l'applicativo ed il sensore ad un insieme di persone e valutando i loro tempi di reazione.

- Nel quinto capitolo vengono esposte le conclusioni di questa tesi e verranno fornite indicazioni ed idee per possibili sviluppi futuri.

Capitolo 1
Il dispositivo Kinect

1.1 Introduzione del dispositivo

Il dispositivo Kinect, è una periferica di gioco per la console XBOX360 di Microsoft, realizzata inizialmente come semplice dispositivo di controllo dei movimenti per applicazioni video ludiche, in risposta a sistemi concorrenti come PSeye e Wiimonte; grazie alle sue elevate caratteristiche prestazionali, la periferica sta attualmente riscontrando successo anche in altri ambiti e sebbene non ufficialmente confermato da Microsoft, il sistema operativo Windows 8 potrebbe supportarla, essendo le SDK per il suo utilizzo già in lavorazione.
In campo video ludico tuttavia, dispositivi simili erano stati già realizzati. Tralasciando le prime tecnologie che prevedevano l'interazione con i software per mezzo di guanti e sensori, è solo alla fine degli anni 90 che vengono investite maggiori risorse per la realizzazione di dispositivi basati sulla visione artificiale. Negli anni 2000 viene distribuito il Dreameye di SEGA, che a causa della novità della periferica e dello scarso supporto, venne semplicemente utilizzata come webcam e come sistema di acquisizione di immagini fotografiche, riducendo di molto le potenzialità del prodotto. E' tuttavia con il successore, l'EyeToy Sony (2003), che sono stati sviluppati software che permettevano, sebbene in maniera poco precisa, di interagire con i software mediante il movimento del corpo piuttosto che con l'utilizzo di un joystick. Sia Dreameye che EyeToy sono estremamente simili alle webcam e usano la tecnologia della computer vision e del riconoscimento dei gesti per elaborare le immagini catturate dalla fotocamera. Come detto in precedenza, questi due dispositivi hanno avuto un discreto

successo, ma il loro utilizzo era ancora rudimentale e soprattutto relegato al semplice utilizzo nel campo dei videogame; sono stati comunque una buona base di partenza per progetti più avanzati che offrono una migliore interazione dell'utente con il software.

Eredi di questi sistemi, sono i dispositivi Playstation Move[2] di Sony, Wii Remote[3] di Nintendo (mostrati in figura 1) e Kinect di Microsoft. Sebbene tutti i dispositivi elencati offrano ottime possibilità d'utilizzo che si discostano dal campo dei videogiochi, è solo con Kinect che è possibile interagire con i sistemi informatici senza l'uso di ulteriori periferiche e contemporaneamente ottenere soddisfacenti risultati. Il sensore Kinect fin dal momento del lancio (4 novembre 2010), ha riscontato un notevole successo tra le comunità hacker e nelle università di tutto il mondo, che ne hanno intravisto immediatamente le potenzialità; per tale motivo Microsoft, inizialmente contraria all'utilizzo al di fuori del sistema di gioco Xbox360, avrebbe permesso lo studio della periferica e lo sviluppo di applicazioni con finalità differenti.

Figura 1: *Sulla sinistra il controller Wii remote e sulla destra il Playstation Move*

In data 10 novembre i driver sono stati resi disponibili e consentono l'utilizzo della telecamera RGB, del'IR e delle funzionalità per percepire la profondità.

[2] Il PlayStation Move è un controller di gioco motion-sensing, utilizza una luce che viene individuata dal PlayStation Eye ed ha dei sensori inerziali per rivelare il movimento. Il PlayStation Eye è una fotocamera digitale che basandosi sulla computer vision ed il riconoscimento dei gesti rivela il movimento e con un arrey di microfoni permette di rivelare il suono.

[3] Il Wii Remote è il primo telecomando basato sulla capacità di rivelazione del movimento (motion-sensing), che permette all'utente d'interagire con oggetti sullo schermo attraverso il riconoscimento di gesti usando un accelerometro ed un sensore ottico.

Nonostante sia stato fatto un notevole lavoro di hacking sul sensore, collegare al PC il dispositivo in esame non porta a nessuna violazione dell'hardware Kinect o della console, né tantomeno vengono modificati gli algoritmi della XBOX360. I driver open-source permettono solo l'utilizzo di una porta USB per l'invio e la ricezione di dati, pertanto i driver e l'uso di Kinect su calcolatore sono del tutto legali. Nel dicembre del 2010 la PrimeSense (6) ha rilasciato i primi driver "ufficiali" con un monitoraggio del movimento chiamato NITE. Il lancio di Kinect ed il rilascio dei driver hanno creato grande interesse tra ricercatori e programmatori. In poche settimane sono nate piccole comunità on-line, spesso guidate da industrie del settore come la citata PrimeSense, che forniscono supporto alla realizzazione di progetti basati sull'interazione naturale.

E' interessante notare che anche ASUS, abbia presentato la sua periferica di Motion Control in collaborazione con Prime Sense. Questo prodotto, dal nome "Wavi Xtion" è specificatamente realizzato per applicazioni su PC e verrà rilasciato nel corso del 2012.

In tre settimane dall'uscita della periferica è stato dimostrato come con Kinect e periferiche simili sia possibile sostituire mouse, tastiere e telecomandi: persone disabili potrebbero utilizzare questi dispositivi per abbattere numerose barriere che impediscono loro l'utilizzo della tecnologia. È stato dimostrato inoltre, come Kinect porti notevoli vantaggi alla robotica con l'uso della visione artificiale per far muovere degli automi. Ad esempio, è possibile, utilizzare Kinect per far volare un elicottero o per far muovere un piccolo veicolo, evitando ostacoli e permettendo di ricreare un ambiente in 3D[4]. Il dispositivo permette anche di risparmiare enormi budget per la realizzazione un sistema di motion capture; altri progetti riguardano l'intrattenimento ed il supporto alla medicina.

Considerato il grande numero di applicazioni possibili è difficile decidere cosa realizzare e limitarsi ad una delle numerose possibilità offerte da dispositivi come Kinect; la prima cosa che si può fare per lavorare con Kinect è analizzare l'hardware della periferica e scoprire cosa può realmente utilizzare uno sviluppatore per rapportarsi con questo sensore. In secondo luogo bisogna scoprire quali driver sono disponibili e per quali sistemi operativi vengono forniti; infine scegliere il linguaggio di programmazione.

[4] La stanza in cui si muoveva il robot è stata ricreata come una nuvola di punti, che rappresentava sedie, tavoli, persone. L'immagine rappresentava il mondo visto dagli occhi del robot.

1.2 Hardware Kinect

Il Kinect è progettato per esser montato sotto un display video ed ha una serie di sensori collegati su una barra orizzontale collegata ad una base per mezzo di un perno motore. Per osservare il meccanismo che permette al Kinect di muoversi è necessario rimuovere la plastica e le 4 viti sotto la base d'appoggio. Qui sotto è situato il motore che permette alla periferica di ruotare. I componenti non sono molto robusti, escluso il piccolo motore tutti gli ingranaggi sono in plastica e quindi facilmente danneggiabili con l'uso. In particolare Kinect raccoglie due tipi di informazioni importati: mappe di profondità e immagini a colori.
Dato che le immagini vengono elaborate sul Kinect e non sul PC, Microsoft ha inserito nella periferica due schede ed una barra di supporto metallico in parallelo, separati da quattro distanziatori metallici. Sul lato è montata una piccola ventola per il raffreddamento, che evita il surriscaldamento e il danneggiamento del dispositivo. Sulla barra metallica sono montati gli occhi della periferica, due telecamere ed un proiettore IR; le telecamere sono simili a webcam dotata di autofocus. La risoluzione è 320x240 per l'IR mentre per le RGB è 640x480.

Figura 2: *Sulla sinistra il dispositivo Kinect e sulla destra i sensori che permettono l'acquisizione delle immagini e della profondità*

La figura 2 mostra il sensore Kinect. Un particolare difficile da notare è il dispositivo Peltier (7), anche detta cella di Peltier, posto tra l'IR e la barra metallica che svolge il ruolo di sistema di raffreddamento. Tra le telecamere e l'IR è situato anche un piccolo LED di stato.
L'IR crea la mappa di profondità, mentre le telecamere forniscono lo spettro visivo. Il sistema è teoricamente in grado di misurare le distanze all'interno di un area di 2 metri con un margine di errore di 1 cm, anche se questi parametri di precisione forniti da Microsoft non sono stati ancora testati realmente.

Il dispositivo è dotato di un array di quattro microfoni collegato alla scheda madre con un connettore a cavo unico. L'array di microfoni permette al Kinect di ricevere i comandi vocali. I microfoni sono tutti e quattro orientati verso il basso, tre sono sul lato destro del dispositivo ed uno sul lato sinistro. L'orientamento dei microfoni non è casuale: la scelta è stata dettata da Microsoft in quanto ritiene che l'orientamento verso il basso sia quello ottimale per la raccolta del suono. Per far si che i comandi vocali funzionino al meglio è necessario calibrare l'array di microfoni ogni qual volta si cambia la disposizione dei mobili nella stanza in cui è montato il Kinect. In figura 3 si può vedere il principio di funzionamento del sensore.

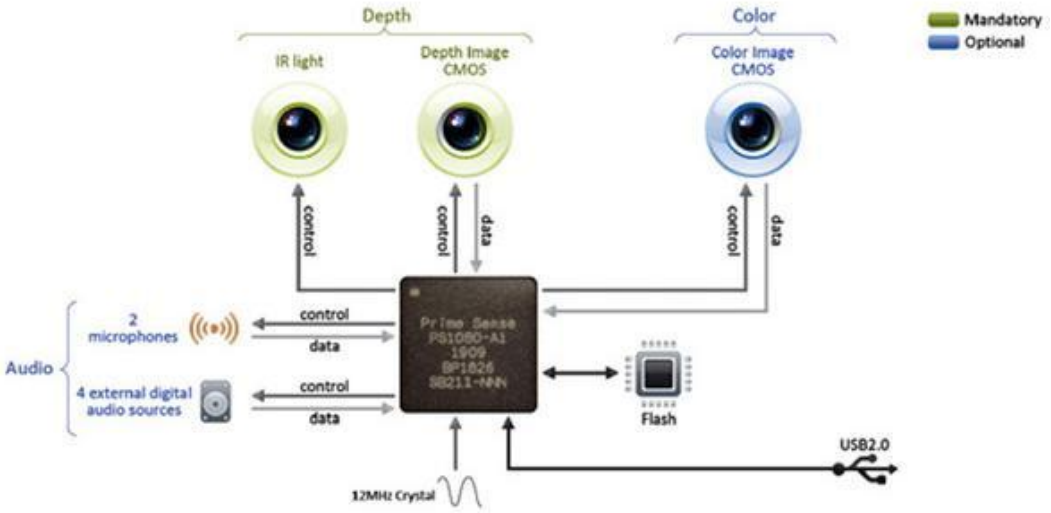

Figura 3: *Schema generale di funzionamento del sensore Kinect*

Per alimentare la periferica, Microsoft usa ben 12 Watt mentre le porte USB sono in grado di fornire in media 2,5 Watt di potenza. Pertanto Kinect necessita anche di un cavo di alimentazione. Questo particolare è molto importante qualora si volesse montare il dispositivo su un robot mobile. Ora che è stato descritto il sistema visivo ed uditivo del dispositivo bisogna capire come avviene l'elaborazione dei dati.

La scheda madre ha 6 chip. Osservando la scheda A della figura 4, da sinistra a destra sono rispettivamente montati:

- Stereo ADC con microfono preamplificato (Wolfson Microelectronics WM8737G)
- N-Channel PowerTrench MOSFET (Fairchild Semiconductor FDS8984)

- Controller USB 2.0 hub (NEC uPD720114)
- Pacchetto SAP 6 mm x 4,9 mm non identificato forse SPI flash (H1026567 XBOX1001 X851716-005 Gepp)
- SoC per il controller dell'interfaccia della macchina fotografica (Marvell AP102)
- SDRAM DDR2 512 megabit (Hynix H5PS5162FF)

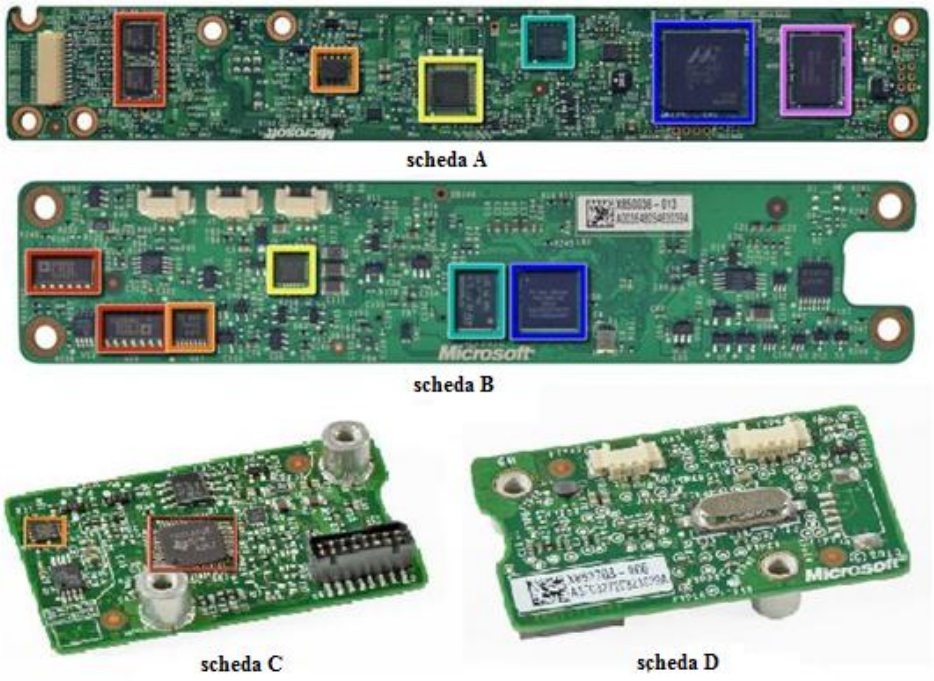

Figura 4: *Componenti interni del sensore Kinect*

Nella scheda B sono montati:

- Due CMOS Rail-to-Rail amplificatore d'uscita a basso costo (Analog Devices AD8694)
- Un campionatore e convertitore A/D 8 bit ad 8 canali, con interfaccia 12C (TI ADS7830I)
- Allegro Microsystems A3906
- Una memoria Flash 1Mb x 8 oppure 512Kb x 16 (ST Microelectronics M29W800DB)
- Un processore d'immagini Soc Sensor (PrimeSense PS1080-A2)

Infine le schede C e D mostrano una scheda che dispone di un controller audio USB frontale e centrale (TI TAS1020B) ed infine sul lato sinistro della scheda si può vedere un accelerometro (Kionix MEMS KXSD9) che probabilmente è utilizzato come stabilizzatore d'immagine. Un'ultima cosa importante è il cavo USB proprietario che la periferica Microsoft usa per collegarsi alle normali console. Questo cavo potrebbe non essere riconosciuto dai sistemi operativi su PC, ma ci sono due alternative. La soluzione più semplice è utilizzare un adattatore che di norma viene venduto con il Kinect che Microsoft inserisce nella confezione per questioni di compatibilità con le prime versioni delle console XBOX 360. La seconda soluzione è costruire un adattatore ad hoc.

1.3 Il funzionamento della periferica

Per capire come funziona la periferica è possibile dividere il sistema in tre sottoblocchi: il monitoraggio dei movimenti, il riconoscimento vocale ed il motore. Li descriveremo brevemente per poi osservare nei successivi capitoli come queste funzionalità permettono di ottenere notevoli risultati con l'uso del Kinect. La prima cosa che interessa ad un utente è farsi riconoscere. Questo compito è svolto dal sistema ottico, che permette di monitorare i movimenti in tempo reale. La struttura è molto complicata ma fornisce funzionalità che fino ad ora erano disponibili solo a fronte di spese notevoli.

Il sistema, come abbiamo visto, è composto principalmente da due parti: un proiettore IR e una fotocamera VGA. La prima cosa che la periferica fa è creare un campo di profondità nella stanza separando l'utente dagli oggetti inanimati.

A secondo della distanza del Kinect, le figure compariranno in diversi colori sullo schermo: gli oggetti in grigio scuro sono quelli più lontani, in grigio chiaro quelli più vicini. Le figure umane che vengono riconosciute possono essere blu, verde, rosso, e così via. Per avere un idea di una mappa di profondità si può pensare di osservare degli oggetti su un tavolino e scattare una foto. Dobbiamo poi colorare i pixel dell'immagine con varie tonalità di grigio in base alla distanza di ogni pixel dalla telecamera.

Quello che otteniamo potrebbe essere un esempio di mappa di profondità, come mostrato il figura 5.

Figura 5: Esempio di mappa di profondità

Tali figure verranno ricostruite dai pixel che il Kinect acquisisce con la telecamera IR. Per essere più precisi il proiettore IR del Kinect getta un fascio di raggi infrarossi (Microsoft ha assicurato che non sono pericolosi per il corpo e per la vista). I raggi riflessi (figura 6) vengono catturati dalla telecamera ad infrarossi e con un algoritmo viene determinato quanto può essere lontano o vicino un punto. Sulla base di queste informazioni è possibile assegnare una tonalità di grigio ad oggetti più o meno distanti.

Figura 6: Il fascio di infrarossi proiettato dal Kinect

L'immagine acquisita dalla periferica Microsoft viene poi fatta passare in diversi filtri, in modo tale che Kinect possa capire cosa è una persona e cosa non lo è. L'intero sistema segue delle linee guida, ad esempio deve sapere che una persona può avere in media un'altezza x, che avrà due gambe, due braccia e "molto

probabilmente" una testa. Questo permetterà in fase di calibrazione, a fronte di una partita, di non avere un frigorifero come avversario.
Non tutte le persone hanno però la stessa conformazione fisica, inoltre spesso vengono utilizzati indumenti larghi o cappelli. Per questo Microsoft ha inserito tra le linee guida degli algoritmi che riconoscono possibili cappelli o maglioni larghi.
Quando questa fase di calibrazione è terminata il dispositivo converte la parte dell'immagine relativa all'identificazione del corpo in uno scheletro che nella fase di tracking permette il movimento delle articolazioni, escluse per ora quelle delle dita. L'intero sistema lavora a 30 fps ed ha 200 pose comuni per lo scheletro precaricate. Nel caso l'utente faccia un movimento che impedisca alla telecamera di riconoscere il gesto fatto, il Kinect userà una delle pose tra quelle presenti che più si adatta al caso per non perdere il tracciamento dell'utente.
La seconda funzionalità importante è il riconoscimento vocale. Abbiamo visto infatti che il Kinect ha un array di quattro microfoni pronto per essere usato a tale scopo. Il sottosistema microfoni della Microsoft ha come obbiettivo di essere sensibile al riconoscimento delle voci fino a 10 metri di distanza cercando di ignorare rumori ambientali. La larghezza del dispositivo Kinect è dovuta proprio al sistema di microfoni: durante i suoi lavori Microsoft ha effettuato test in 250 abitazioni utilizzando 16 microfoni disposti in modo differente.
La soluzione ottima è stata l'array di quattro microfoni rivolti verso il basso, in modo da mantenere pulita la parte anteriore della periferica. L'array funziona meglio nel raccogliere le voci a distanza, ma necessita di aiuto. C'è un unità di elaborazione a bordo del Kinect che toglie il rumore che si crea in prossimità dei sistemi surround 5.1, mentre un secondo sistema software Beam Forming (8) agisce con la telecamera per capire dove si sta creando una possibile fonte di suoni intorno a l'utente. Questo permette di aiutare il Kinect a capire quando non è l'utente a parlare ma altre persone intorno a lui. Il sistema di riconoscimento vocale, attivo solo su console, ha un modello acustico per ogni singolo paese che comprende anche diversi dialetti regionali. I microfoni sono in ascolto in ogni momento rendendo il sistema Kinect open-mic.
A questo punto rimane da capire come funziona il sottoblocco motore. L'idea di inserire un piccolo motore all'interno della base inferiore del Kinect è dovuta alle necessità di calibrazione nelle diverse abitazioni europee, asiatiche ed americane. Per Microsoft la telecamera doveva essere in grado di muoversi in su ed in giù per calibrare ogni singolo spazio, effettuando movimenti di circa 30 gradi.
Un'altra funzionalità importante del motore è quella dello zoom per la fotocamera, che permette di espandere lo spazio visivo.

Questa funzionalità è stata progettata per la video chat di Kinect, in modo che se più utenti sono nella scena ed uno viene tagliato il motore gestisce in automatico lo zoom della fotocamera per far entrare tutti i partecipanti della conversazione sullo schermo.

Capitolo 2
Strumenti software utilizzati

2.1 Installazione dei Driver

Ora che conosciamo i componenti della periferica e quali sono le funzionalità di cui Microsoft ha voluto dotarla possiamo passare alla scelta del sistema operativo e dei driver. Inizialmente si poteva lavorare solo con sistemi operativi Linux ma successivamente i driver per Kinect sono stati portati anche sui sistema operativi Windows di Microsoft (Xp, Vista e 7) ed infine gli ultimi driver sono usciti anche per Mac.

Per provare il dispositivo si è scelto di lavorare sui sistemi operativi Windows, la procedura d'installazione che verrà descritta ed il software che verrà realizzato alla fine del lavoro di tesi è stato provata su Xp, Vista e Windows 7. Il dispositivo è stato fatto funzionare con due tipi di driver differenti che chiaramente non possono lavorare assieme. I primi sono libusb-win32 che si possono procurare dal sito di source forge, mentre i secondi sono le Avin2-SensorKinect forniti dalla PrimeSense che ha collaborato alla realizzazione del Kinect. La scelta del driver va fatta considerando con che cosa si vorrà lavorare. Istallando i primi driver la maggior parte del lavoro verrà orientata sulle librerie OpenCV (9), e sarà anche possibile programmare con il linguaggio Processing. Istallando i driver della PrimeSense invece si lavorerà principalmente con le librerie OpenNI per la Natural Interaction e con il linguaggio C# o C++.

Per il lettore che successivamente vorrà testare il progetto allegato alla tesi si consiglia di non installare le librerie Libusb ma di prendere solo visione della procedura qualora si decida di cambiare ambiente di lavoro.
Le pagine che seguiranno potranno risultare noiose per chi leggerà questa tesi per puro piacere, ma per chi vorrà mettersi all'opera con l'uso del Kinect le prossime pagine permetteranno di risparmiare molto tempo e saranno una guida fondamentale per i primi passi da compiere. Si è voluto porre attenzione alle fasi principali dell'istallazione dei driver in quanto un errata procedura impedirebbe di lavorare con il sensore. La parte più importante resta comunque la seconda, (dedicata ad OpenNI e NITE) che riguarda l'installazione di tre pacchetti e la modifica di alcuni file.

2.1.1 Libusb-win32

La prima cosa da fare è procurarsi le librerie open source libusb-win32 (10) da sourceforge[5] e scompattarlo sul computer, si consiglia di farlo in "c:\kinect". È possibile compilare le librerie sul proprio computer oppure usare la versione già compilata nella cartella Bin.
Successivamente scaricare i driver Kinect[6], in particolare i file .cat e .inf, ci aiuteranno a capire quali driver dovranno essere utilizzati sul sistema.
Bisogna decomprimere i file in "c:\kinect\libusb-win32-bin-1.2.2.0\bin", questo servirà a Windows per trovare tutti i file appropriati in una cartella.
A questo punto andremo a collegare il Kinect al computer utilizzando l'apposito adattatore ed attenderemo che si apra la finestra di pop-up driver per selezionare "Individuare ed Installare i driver" facendo clic su Avanti.
Ci vorrà poco ed apparirà una richiesta di sicurezza, bisognerà accettarla e continuare fino a quando una successiva richiesta chiederà dove prendere i driver per la periferica. Non avendo con noi un disco selezioneremo "Non ho un disco, mostra altre opzioni". Dobbiamo cliccare su "Sfoglia il computer per il software del driver (Avanzato)" in modo che potremmo specificare manualmente la cartella Bin di libusb (dove abbiamo precedentemente decompresso i file .inf) e fare clic su avanti. Comparirà un messaggio di errore in quanto i driver non sono siglati, quindi bisognerà selezionare "Installa il software del driver". Questa operazione dovrà essere ripetuta per altre due volte fino a quando tutti i dispositivi Camera,

[5] Link download delle libus-win32 (http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.2.0/libusb-win32-bin-1.2.2.0.zip/download)
[6] Link download driver Kinect (http://www.roborealm.com/downloads/Kinect_Drivers.zip)

Audio e Motor non saranno installati. Ora il Kinect è pronto per essere utilizzato, si può fare una prima prova per verificare la corretta installazione dei driver scaricando l'esempio di MicrosoftKinect sul sito della RoboRealm[7].

2.1.2 Avin2-SensorKinect

Questo secondo procedimento va effettuato solo se non sono stati installati i driver libus-win32 o se sono stati opportunamente disinstallati.
I driver si possono trovare sul sito della Github[8], successivamente bisogna procurarsi OpenNI [9] in particolare i file binari win32 v1.0.0.25 e NITE[10] versione 1.3.0.18 non stabile.
Bisognerebbe soffermarsi per spiegare cosa sono OpenNI e NITE, ma lo faremo accuratamente in un secondo momento, per ora è importante sapere che OpenNI (11) è una piattaforma aperta per la Natural Interaction, non è legata a Kinect e può funzionare con altri sensori, permettendo di lavorare sulle mappe di profondità. NITE è una struttura, un insieme di algoritmi, che funzionano in cima ad OpenNI. Grazie a NITE è possibile usare le informazioni acquisite per creare dei modelli per il monitoraggio del corpo, delle mani e l'individuazione di utenti multipli. Se si dispone dei file elencati far funzionare il Kinect su pc è abbastanza semplice, la prima cosa da fare è decomprimere i file del SensorKinect sul proprio PC (c:\kinect) e raggiungere la directory **Platform\Win32\Driver.**
Bisognerà scegliere quali driver installare a secondo del proprio sistema operativo: x86 per sistemi a 32bit, oppure amd64 per Windows a 64 bit.
La periferica Kinect andrà collegata al computer con apposito adattatore e cavo di alimentazione. Windows non tarderà a chiedere l'installazione dei driver in tre parti: camera, audio, motor. Quando il sistema operativo vi permetterà di scegliere la cartella da cui prendere i driver si dovrà andare nella directory dove è stato estratto SensorKinect.
In pochi minuti i driver saranno installati, per controllare che tutto sia andato a buon fine andare su *Start→Pannello di controllo→Sistema→Gestione dispositivi* sarà possibile trovare la periferica PrimeSensor come in figura 7. Si potrà notare che mancano i driver per l'audio, questi non sono ancora disponibili, quindi per ora non sarà possibile lavorare sui comandi vocali e l'array di microfoni non verrà sfruttato.

[7] Sito RoboRealm (http://www.roborealm.com/help/Microsoft_Kinect.php)
[8] Link download driver da github (https://github.com/avin2/SensorKinect)
[9] Link download OpenNI (www.openni.org/?q=node/2)
[10] Link download NITE (www.primesense.com/?p=515)

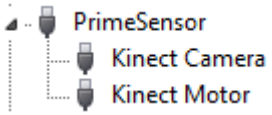

Figura 7: *Dispositivo Kinect attivo sul proprio sistema*

Ora non resta che creare il proprio ambiente di lavoro installando l'SDK OpenNI (preferibilmente in C:\Programmi\) avviando l'eseguibile.
Installare allo stesso modo NITE lasciando la directory di default ed usando la chiave (**0KOIk2JeIBYClPWVnMoRKn5cdY4=**) qualora venisse richiesta.
Una volta ultimate le due installazioni accedere alla sottodirectory Bin del SensorKinect-0124bd2 e selezionare l'eseguibile **SensorKinect-Win32-5.0.0** (la 5.0 è l'attuale versione corrente).
Adesso bisognerà copiare e sovrascrivere i file XML che si trovano nella directory PrimeSense\Sensor\SampleXMLs\NITE\Data in PrimeSense\NITE\Data.
Dopo copiamo i file in PrimeSense\Sensor\SampleXMLs\OPENNI\Data dentro la directory c:\Programmi\OpenNI\Data.
Ora dobbiamo apportare qualche modifica sui file installati. Questa parte è molto importante in quanto qualsiasi sensore può essere mappato con un file XML esattamente come SamplesConfig e se un giorno vorremmo sostituire il Kinect con un altro dispositivo basterà modificare questo file.

SamplesConfig.xml - Blocco note

File Modifica Formato Visualizza ?

```
<OpenNI>
        <Licenses>
                <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4=" />
        </Licenses>
        <Log writeToConsole="true" writeToFile="true">
                <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
                <LogLevel value="3"/>
                <Masks>
                        <Mask name="ALL" on="true"/>
                </Masks>
                <Dumps>
                </Dumps>
        </Log>
        <ProductionNodes>
                <Node type="Image" name="Image1">
                        <Configuration>
                                <MapOutputMode xRes="640" yRes="480" FPS="30"/>
                                <Mirror on="true"/>
                        </Configuration>
                </Node>
                <Node type="Depth" name="Depth1">
                        <Configuration>
                                <MapOutputMode xRes="640" yRes="480" FPS="30"/>
                                <Mirror on="true"/>
                        </Configuration>
                </Node>
                <!--
                <Node type="Audio" name="Audio1">
                </Node>
                -->
        </ProductionNodes>
</OpenNI>
```

Figura 8: *esempio di file XML*

Il file XML, come quello in figura 8, ha tre importanti sezioni: Licenza, Nodi di registrazione e produzione. Per i nostri scopi basterà cambiare la licenza dei file XML appena copiati all'interno delle cartelle Data di OpenNI e NITE. La sezione licenza è molto semplice è composta da un solo tag che cita il fornitore e la chiave. Questa è l'unica parte del documento che si deve modificare necessariamente, eliminando eventuali commenti ed inserendo la seguente riga al posto di quella presente.

```
<License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4=" />
```

Se si prende in considerazione come esempio il file SamplesConfig di OpenNI si può notare che nel documento segue la sezione Log che permetterà di generare file utili per il debug.
Per attivare questa funzione bisogna impostare writeToConsole e writeToFile a "true". L'ultima cosa importante è la sezione che si prende cura dei nodi di produzione, come è stato precedentemente detto le OpenNI possono lavorare con una varietà di sensori ed input sensoriale (profondità, immagine, gesti, ecc…) che devono essere inseriti in un nodo in questo file XML. Si può infatti notare che sul file originale ci sono due nodi, il primo Image permette di lavorare sulle immagini RGB, il secondo Depth permette di lavorare sulla profondità.
La risoluzione del Kinect è 640x480 ed i frame per secondo sono 30 (per Depth può esser portato il valore fps fino ad un massimo di 60), per questo verranno settati opportunamente i parametri MapOutputMode e fps.
Se la procedura è stata seguita correttamente da questo momento è possibile provare tutti gli esempi presenti nelle cartelle Sample di OpenNI e NITE, inoltre nella cartella Documentation la PrimeSense ha rilasciato delle brevi guide.
NITE ha reso disponibili anche delle demo in flash con apposita documentazione all'interno della cartella Wrappers.
In caso di errore nell'eseguire i programmi della OpenNI o NITE, comparirà il messaggio "Bad Parameter Sent!" bisogna controllare i file XML, assicurandosi che la chiave di licenza oppure i valori di risoluzione e fps siano corretti.

2.2 OpenNI

OpenNI ovvero Open Natural Interaction è un framework sotto licenza LGPL (12) indipendente dalle piattaforme di lavoro e multi-linguaggio che definisce le API per scrivere applicazioni che usano le Natural Interaction.

L'intento di OpenNI è creare uno standard API per svolgere due compiti:

- comunicare con sensori visivi ed audio, per percepire figure e acquisire suoni
- sviluppare funzionalità software (middleware) per analizzare e elaborare dati video ed audio registrati in una scena.

I sensori ed i componenti middlware non sono dipendenti, ciò permette di scrivere applicazioni che elaborano dati senza doversi preoccupare del sensore che li ha prodotti. Il framework OpenNI è un livello astratto che fornisce l'interfaccia per i dispositivi fisici e componenti middlware.

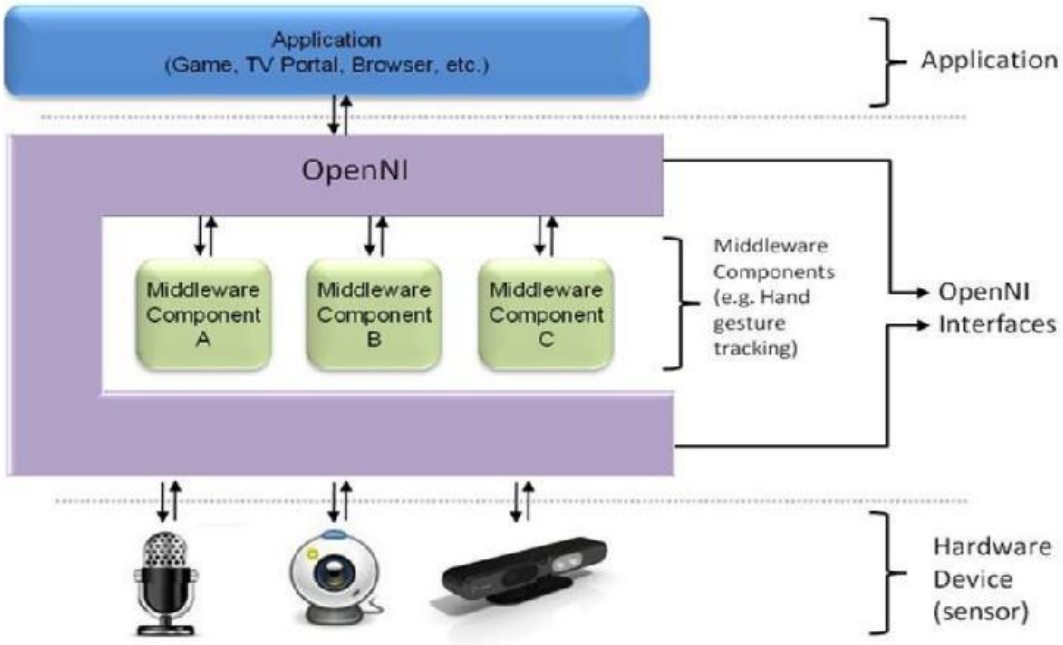

Figura 9: *Astrazione dei livelli di funzionamento di OpenNI*

Il livello più alto rappresenta il software che fa uso di OpenNI implementando le Natural Interaction. Il livello centrale (middle) rappresenta l'interfaccia OpenNI con le sue capacità di comunicare con i vari sensori e con le funzionalità disponibili (tracciamento dei gesti, riconoscimento delle mani, ecc…).
Il livello più basso è il livello hardware composto da tutti i sensori che possono inviare dati audio o visivi. Questi componenti sono indicati come moduli ed attualmente quelli supportati dalle API sono:

Moduli per i sensori

- Sensore 3D
- RGB fotocamera
- macchina fotografica di IR
- Audio dispositivo (un microfono o un array di microfoni)

Middleware componenti

- Analisi totale del corpo: è un componente software che elabora i dati sensoriali e genera informazioni relative al corpo come una struttura dati che descrive le articolazioni, il loro orientamento, il centro di massa del corpo e molto altro.
- Analisi della mano e punto: è un componente software che elabora i dati sensoriali, individua la figura di una mano assegnando alla sua posizione un punto.
- Rilevamento gesto: è un componente software che identifica gesti predefiniti come una mano che fluttua associando al gesto una richiesta.
- Analisi della Scene: è un componente software che analizza l'immagine della scena, al fine di produrre informazioni come: individuare più persone nella scena, trovare le coordinate del piano, separare oggetti in primo piano da quelli sullo sfondo.

Proprio come ROS[11] anche OpenNI ha rilasciato i driver ed ha aperto i propri codici sorgente per far si che le sue librerie vengano implementate su più architetture possibili e su qualsiasi sistema operativo, in modo da accelerare l'introduzione di applicazioni di Natural Interaction sul mercato. Con l'uscita del dispositivo Kinect la popolarità di OpenNI è nettamente aumentata, grazie anche alla creatività dei numerosi sviluppatori che lavorano con queste librerie.
Va sottolineato che OpenNI non è Kinect, ma la facilità del framework di comunicare con qualsiasi sensore ha solo facilitato l'uso del dispositivo Microsoft.

[11] ROS è un sistema meta-operativo per i Robot. Fornisce i servizi di un sistema operativo, include l'astrazione hardware e consente le funzionalità di passaggio di messaggi tra processi. È possibile usare ROS con OpenNI che permette l'integrazione dei sensori PrimeSense con ROS. (http://www.ros.org/wiki/ni)

2.2.1 Uso delle librerie OpenNI

Fino a questo momento abbiamo osservato il sensore Kinect e fatto si che questo possa interagire con noi ed il nostro computer in un ambiente Windows.

Se i passi descritti fin ora sono stati seguiti correttamente, è possibile eseguire qualsiasi programma già compilato che interagisca con il Kinect basandosi sui driver SensorKinect. È facile trovare molte demo che sono basate su altri driver, questi programmi non funzioneranno correttamente. Come abbiamo già detto precedentemente ancora non c'è nulla di ufficiale, se si sceglie di usare OpenNI su una macchina non possiamo tenere installati anche i driver libusb-win32. Una volta scelto in che ramo indirizzare il lavoro bisognerà assicurarsi di avere installato la versione più recente di Microsoft Platform SDK.

Per prima cosa aprire un nuovo progetto di Visual Studio 2008 (o superiore), nel menù laterale di Visual studio cliccare con il tasto destro del mouse sul nome del progetto e successivamente su proprietà.

Nella sezione *C/C++* → *General* selezionare **Additional Include Directories** ed aggiungi "$(OPEN_NI_INCLUDE)", questa variabile terrà conto della directory Include di OpenNI che di default sarà *C:\Program files\OpenNI\Include*.

Ora bisogna andare nella sezione *Linker*→ *General* e selezionare **Additional Library Directories** per aggiungere la variabile "$(OPEN_NI_LIB)" che punta alla directory Lib di OpenNI; il valore di default è *C:\Program files\OpenNI\Lib*.

Nella sezione *Linker* → *Input* →*Additional Dependencies* bisogna aggiungere anche OpenNI.lib come nodo di input.

Abbiamo già chiarito come OpenNI usa i file XML, quindi bisogna assicurarsi che all'interno della directory *C:\Program files\OpenNI\Data* i file XML siano configurati correttamente. In particolare la licenza dei file licenses.xml e SamplesConfig.xml deve essere la seguente:

```
<Licenses>
 <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4=" />
</Licenses>
```

Bisogna assicurarsi di aver aggiunto le directory Include e Library nelle proprie directory di Release e Debug.

Infine bisogna andare a creare due variabili d'ambiente. La prima variabile d'ambiente[12] sarà OPEN_NI_INCLUDE ed il suo valore sarà la directory di default *C:\Program files\OpenNI\Include*. La seconda variabile d'ambiente sarà OPEN_NI_LIB ed il suo valore sarà *C:\Program files\OpenNI\Lib*. Chiaramente i due valori cambieranno se OpenNI è stato installato in una posizione differente.

Ora che l'ambiente di lavoro è pronto basta includere come unica libreria obbligatoria XnOpenNI.h se decidiamo di lavorare in C, altrimenti XnCppWrapper.h se usiamo il C++.

Possiamo in alternativa compilare uno dei codici sorgenti d'esempio forniti dalla PrimeSense per assicurarsi che il framework OpenNI sia stato installato correttamente. I sorgenti si trovano nella directory OpenNI\Samples.

Bisogna semplicemente aprire ad esempio NiSimpleViewer e compilare il progetto **NiSimpleViewer_2008.vcproj** dopo essersi assicurati come sempre che il file SamplesConfig.xml sia stato modificato adeguatamente.

2.2.2 Generare mappe di profondità

La funzionalità di base di OpenNI è creare una mappa di profondità come quella illustrata in figura 10, per farlo basta inizializzare un oggetto Context che useremo per creare e leggere i dati di profondità.

Figura 10: *Esempio di mappa di profondità.*

[12] Sotto Windows Vista e Windows 7 andare su Start→Pannello di Controllo→Sistema→Cambia impostazioni

La prima cosa da sapere è che OpenNI ha un meccanismo di controllo delle funzioni che ritorna un codice di errore di tipo XnStatus se si è verificato un problema. Possiamo creare una variabile di questo tipo ed assegnargli il valore **XN_STATUS_OK**, questo è l'unico valore per indicare che non ci sono stati problemi. Possiamo usare questa variabile per controllare i valori restituiti dalle varie funzioni per verificare che l'operazione sia andata a buon fine. Volendo una descrizione dell'errore si può usare la funzione **xnGetStatusString().**

```
#define CHECK_RC(rc, what)
  if (rc != XN_STATUS_OK){
    printf("%s failed: %s\n", what, xnGetStatusString(rc));
    return rc;
    }
```

Si può così creare un controllo degli errori che useremo ogni volta che ci sarà la creazione o l'inizializzazione di un oggetto. Se l'errore si verifica verrà stampato un messaggio a video e verrà chiusa l'applicazione. A questo punto è possibile procedere alla creazione della mappa di profondità, il procedimento non è molto complesso e l'uso delle librerie OpenNI permette di ottenere il risultato in poche righe di codice all'interno del main. La prima cosa da fare è creare un oggetto Context ed un oggetto generatore di profondità, inizializzarli e verificare che non ci siano stati errori.

```
using namespace xn;
int main(){
      // Associo ad una variabile dei valori di status del sensore
      XnStatus nRetVal = XN_STATUS_OK;
      // Creazione ed inizializzazione dell'oggetto Context
      Context context;
      nRetVal = context.Init();
      //controllo eventuali errori
      CHECK_RC(nRetVal, "Initialize context");
      // Creo il generatore di profondità
      DepthGenerator depth;
      nRetVal = depth.Create(context);
      // Controllo eventuali errori
      CHECK_RC(nRetVal, "Create depth generator");
```

Sarà così possibile iniziare a generare i dati che vengono acquisiti dal sensore Kinect. La funzione che si utilizzerà sarà **StartGeneratingAll**(), anche questa come tutte le funzioni genera un messaggio di errore che dovrà esser controllato.
OpenNI ha messo a disposizione un oggetto **DepthMetaData** di supporto ai generatori che creano mappe di profondità.
Quando generiamo la mappa di profondità useremo il metodo **Data()** di questo oggetto, che restituisce un puntatore ai dati di profondità.

```
        nRetVal = context.StartGeneratingAll();
        CHECK_RC(nRetVal, "StartGeneratingAll");
        DepthMetaData depthMD;

        while (!xnOSWasKeyboardHit())  // mando in loop
        {
                // Attendo che i nuovi dati siano disponibili
                nRetVal = context.WaitOneUpdateAll(depth);
                controllo eventuali errori
                if (nRetVal != XN_STATUS_OK)
                {
                        printf("UpdateData failed: %s\n",
                        xnGetStatusString(nRetVal));
                        continue;
                }
                // Ottengo gli attuali meta-dati di profondità
                depth.GetMetaData(depthMD);
                // Prendo l'attuale mappa di profondità
                const XnDepthPixel* pDepthMap = depthMD.Data();
                printf("Frame %d Middle point is: %u.\n",
                depthMD.FrameID(), depthMD(depthMD.XRes()/2,
                depthMD.YRes()/2));
        }
        context.Shutdown();   // Chiudo il Context
        return 0;
}
```

La mappa verrà generata sui dati forniti dal sensore ad ogni instante, per questo useremo un ciclo while che userà al suo interno l'oggetto context per richiedere

nuovi dati facendo uso del metodo **WaitOneUpdateAll().** Nel momento in cui questo metodo verrà invocato, sarà inviata la richiesta di aggiornamento a tutti i nodi generatori del contesto profondità.
Con l'uso di queste classi messe a disposizione da OpenNI quello che otterremo dal codice descritto sarà una finestra che mostrerà al suo interno un immagine con diverse tonalità di grigio (o giallo) che abbiamo descritto precedentemente ed abbiamo chiamato mappa di profondità.
Le classi messe a disposizione da OpenNI permettono ad un programmatore di ricevere ed utilizzati dati ottenuti da un generico sensore, sensa preoccuparsi del tipo di hardware che li fornisce. La creazione di mappe di profondità non è l'unica funzionalità che OpenNI ci fornisce. OpenNI mette a disposizione delle classi che permettono di individuare un nuovo utente quando questo entra nel campo visivo del sensore ed associa ad esso un colore per distinguerlo da altri. Fornisce metodi ed algoritmi di ricerca di una determinata posa, che permette la calibrazione dell'utente per consentire la creazione di uno scheletro basato sul suo corpo.
Tale scheletro tiene traccia dei movimenti dei vari arti del corpo in uno spazio 3D. Altre classi permettono di individuare il movimento delle mani, di riconoscerli e di tenere traccia della mano per mezzo dell'assegnazione di un "punto mano". Da questo momento con "punto mano" ci riferiremo ad un punto in uno spazio a tre dimensioni (X,Y,Z). Tale punto identifica il palmo della mano dell'utente che verrà usato per interfacciarsi con il sensore ed i programmi.
OpenNI permette anche di lavorare con i suoi generatori Audio che consentono di raccogliere audio all'interno di un buffer con una qualità definita dall'utente in base alla qualità dei microfoni a disposizione. È inoltre possibile registrare i dati generati come lo scheletro dell'utente e salvarli su file con il formato proprietario (.Oni).

2.3 NITE

OpenNI restituisce dati a basso livello come mappe di profondità, mappe di colori, audio ed altro. Nite[13] (13) lavora ad un livello superiore: è un toolbox implementato sulle interfacce OpenNI per facilitare la creazione di applicazioni di controllo basato sul movimento delle mani dell'utente e sullo scheletro. Il funzionamento di NITE è basato sui livelli illustrati in figura 11.

[13] Attualmente NITE è disponibile per i sistemi Windows, Linux e Mac.

OpenNI Modules: I moduli OpenNI supportati da NITE sono il generatore di gesti e il generatore di mani. Inoltre NITE fornisce esempi di moduli per analizzare scene, generare utenti e fare il tracking dello scheletro.

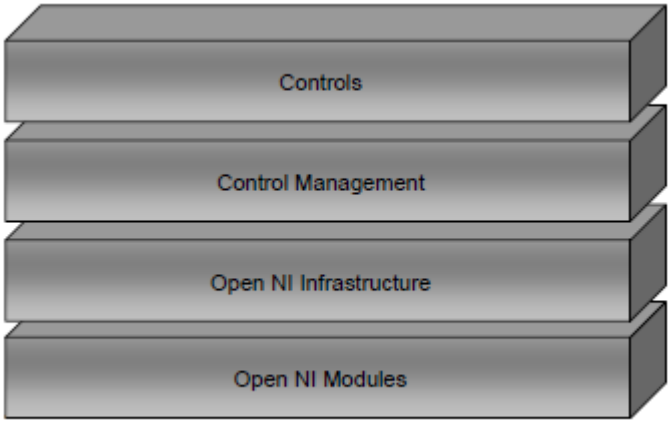

Figura 11: *Schema dei livelli NITE*

OpenNI infrastructure: descritta nei documenti OpenNI e nel paragrafo 2.2

Control Management: Riceve un insieme di punti e linee appropriate al NITE controls.

Controls: Ogni controllo riceve in ingresso un flusso di punti e li traduce in azioni significative specifiche a tale controllo. Il controllo successivamente effettua una callback dall'applicazione che potrà cambiare l'azione corrente nel livello di controllo di gestione. Questo definisce il flusso dell'applicazione.
Un esempio di Controls è il Controllo del punto, che permette di registrare quando un punto viene creato, quando scompare e quando si muove.
In realtà i controlli sono degli oggetti sempre in ascolto che attendono l'arrivo di nuovi dati ad ogni fotogramma. Le funzioni di callback saranno invocate solo quando un determinato evento si verifica.
Per capire con degli esempi di cosa si occupano i controllori è possibile fornire una lista di alcuni esempi con i relativi eventi associati:

- **Push Detector:** Il controllore cerca di riconoscere il moto della mano in avanti e indietro nella direzione del sensore come il movimento di una spinta. Riconoscere questo potrebbe permette ad un utente di aprire una cartella su desktop.

- **Swipe Detector:** Il controllore cerca di riconoscere il movimento di una mano che si muove in una direzione specifica (ad esempio su e giù o destra e sinistra), con una traiettoria rettilinea che termina in un punto. Questo evento potrebbe essere usato per far scorrere le pagine di un documento pdf.

- **Steady Detector:** Il controllore cerca di riconoscere quando una mano è ferma per qualche istante. Questo è utile per passare da un controllo ad un altro ed avere un punto di riferimento iniziale fisso.

- **Wave Detector:** Il controllore prende come evento l'onda, ovvero un movimento della mano che all'interno di un intervallo di tempo effettua un movimento contiguo con quattro cambi di direzione. Più avanti verrà spiegato nel dettaglio.

- **SelectableSlider1D:** Cerca di riconoscere il movimento della mano come punto di un cursore. Viene preso in considerazione uno spazio tridimensionale e gli assi sono così definiti: x (sinistra-destra), y (su–giù), z (avanti-indietro). Questo controllore può esser usato per implementare dei menù.

- **SelectableSlider2D:** Questo punto di controllo cerca di riconoscere il movimento della mano come punto di un dispositivo di scorrimento 2D sul piano XY.

- **Circle Detector:** L'utente disegna due circonferenze complete, in senso orario (considerato positivo) e antiorario (negativo), a questo punto il generatore di output seguirà i movimenti circolari successivi. Questo controllo molto probabilmente è stato utilizzato per far funzionare una versione simulata dell'interfaccia tastiera di 8Pen Android per scrivere testo su windows 7 (figura 12).

NITE permette inoltre di creare delle proprie classi che ereditano tutte le funzionalità delle classi base e mette a disposizione dei filtri che puliscono i dati ricevuti rimuovendo rumore o dati non utili. Da qui nasce il problema di capire cosa è rumore da filtrare e cosa no, per fare questo è possibile creare dei vincoli addizionali nei controllori. Ad esempio si può definire un intervallo di tempo

all'interno del quale un gesto può essere riconosciuto, mentre tutto quello che viene immediatamente prima o dopo è rumore.

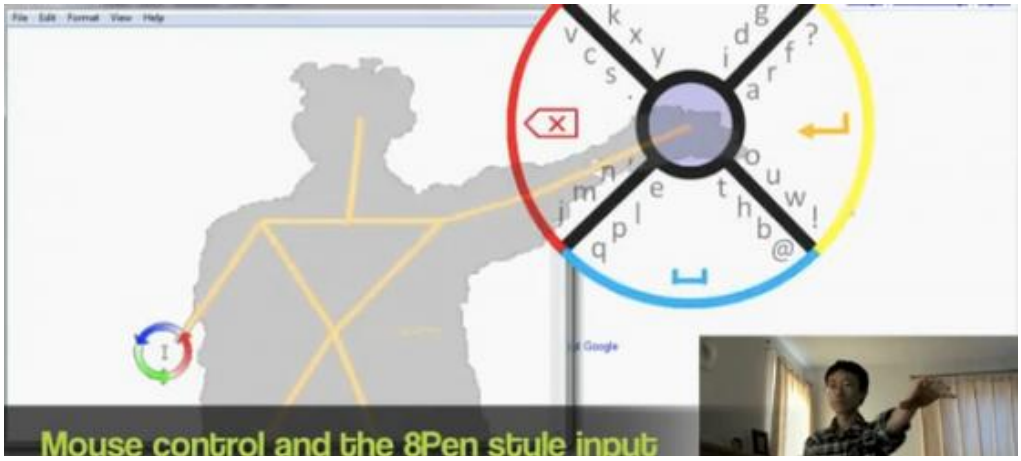

Figura 12: *Esempio di controllo mouse e tastiera per window*

Attualmente il software fornito da NITE consente di lavorare con i moduli per l'interfaccia utente e con Skeleton. Inoltre, per permettere uno sviluppo rapido di applicazioni, NITE offre anche un sistema di gestione delle sessioni che affronteremo nelle prossime pagine.

2.3.1 Configurazione delle librerie software

Se si lavora con NITE vuol dire che si sta già lavorando con OpenNI, quindi tutti i passaggi descritti nei paragrafi precedenti devo esser stati eseguiti correttamente, in particolare quelli relativi alla creazione di un progetto OpenNI. A questo punto è possibile apportare delle modifiche al progetto.

Cliccando sul nome del progetto ed aprendo la pagina delle proprietà si andrà a modificare il campo **Additional Include Directories** sotto *C/C++* → *General* e si inserirà la variabile "$(XN_NITE_INSTALL_PATH)" che punta alla directory Include di NITE, il cui valore di default è il seguente *C:\Programmi\Prime Sense\NITE\Include*.

Nella sezione Linker → *General* bisognerà inoltre modificare il campo **Additional Library Directories** inserendo la variabile "$(XN_NITE_INSTALL_PATH)" che punta alla directory Lib di NITE (default *C:\Programmi\Prime Sense\NITE\Lib)*. Sempre nella sezione Linker bisognerà effettuare una seconda modifica aggiungendo le librerie OpenNI.lib e XnVNite.lib in **Additional Dependencies** sotto nodi di *Input*.

A questo punto dobbiamo solo definire la cartella di debug (…bin\Debug) e seguire lo stesso procedimento descritto nel paragrafo 2.2.1 per creare le variabili d'ambiente necessarie. Sarà possibile da questo momento far funzionare NITE con i moduli di OpenNI per il riconoscimento dei gesti. Prima della descrizione di un progetto realizzato con le librerie d'interazione naturale, verranno descritte alcune delle funzionalità NITE che sono state messe a disposizione e sulle quali possiamo basare l'applicazione. In particolare verranno osservate le sessioni, il riconoscimento della mano, dei gesti, la calibrazione ed il tracking dello scheletro umano. Questo insieme di strumenti ci permetterà di creare un'interfaccia per interagire direttamente con un applicativo o con una macchina senza utilizzare joystick, mouse, tastiere o telecomandi di vario genere.

2.3.2 Gestione delle sessioni

Per sessione si intende lo stato dell'utente di poter controllare un sistema per mezzo dei punti mano o scheletro. La sessione può avere tre stati:

- **Not in session**. Esiste un gesto che gli algoritmi di NITE riconoscono come input per avviare una sessione: questo gesto è detto "gesto di focus". Si resta nello stato di "non in sessione" fino a quando questo gesto non viene riconosciuto. Se l'utente compie il gesto focus si passa al secondo stato "In session".

- **In session.** In questo stato gli algoritmi della NITE cercano di monitorare le mani che dovranno restare davanti al sensore. Non è richiesto che le mani restino ferme. Quando la mano è monitorata viene assegnato un punto mano che è visualizzato sullo schermo come un punto bianco nella mappa di profondità. Questa operazione di visualizzazione si appoggia alle librerie OpenGL.

- **Quick refocus.** Quando siamo in una sessione, ma i punti mano vengono resi non identificabili volontariamente o involontariamente, NITE fornisce un lasso di tempo entro il quale non chiude la sessione permettendo all'utente di recuperarla attraverso il gesto focus oppure uno differente e più semplice.

L'oggetto che si occuperà di gestire la sessione sarà **XnVSessionManager**. È possibile creare dei gesti che non siano moduli OpenNI. Per farlo il gesto deve ereditare **XnVGesture** ed essere passato a XnVSessionManager per mezzo del suo metodo **SetGesture(XnVGesture* pGesture)**. Il parametro che gli viene passato è il nuovo gesto creato dall'utente. Per far funzionare una sessione sono necessarie quattro fasi: creazione delle chiamate d'avvio e chiusura della sessione, inizializzazione della sessione, registrazione delle chiamate di sessione.

Creare le chiamate di sessione

```
void XN_CALLBACK_TYPE SessionStart (const XnPoint3D& pFocus,
void* UserCxt)
{
    // Sessione iniziata
    printf("Session start: (%f,%f,%f)\n", ptPosition.X,
    ptPosition.Y, ptPosition.Z);
    g_SessionState = IN_SESSION;
}

void XN_CALLBACK_TYPE SessionEnd(void* UserCxt)
{
    // Sessione terminata
    printf("Session end\n");
    g_SessionState = NOT_IN_SESSION;
}
```

Inizializzazione

In questa fase viene creato il contesto OpenNI ed un oggetto per gestire la sessione che successivamente verrà inizializzata per mezzo del metodo Initialize. I parametri che vengono passati ad Initialize sono: il gesto da usare come focus, il gesto da usare come refocus, ed il tracker da usare per il monitoraggio.

```
xn::Context context;
XnVSessionManager sessionManager;
XnStatus  rc=sessionManager.Initialize(&context,"Wave,Click",
"RaiseHand");
```

Registrare le chiamate di sessione

Il primo parametro di RegisterSession indica il contesto da utilizzare quando vengono invocate le funzioni, il secondo ed il terzo parametro sono le funzioni da invocare quando si avvia o si chiude una sessione:

```
sessionManager.RegisterSession(NULL,&SessionStart,
&SessionEnd);
```

Gestione della session nel main

```
while (1)
{
context.WaitAndUpdateAll();
sessionManager.Update(&context);
}
```

È importante infine evidenziare le capacità di NITE di gestire più attività concorrenti. NITE offre un supporto multi-threaded per l'uso dell'oggetto **XnVMessageListener** che sta alla base di tutti i controlli NITE.

Un oggetto XnVMessageListener quando viene creato ha come impostazione predefinita un singolo thread, ma può gestire operazioni su più thread. In particolare se viene ricevuto un evento nel thread in esecuzione questo viene eseguito immediatamente, altrimenti se l'evento ricevuto non fa parte di un thread in esecuzione allora viene inserito all'interno di una coda di funzionamento interno. Se XnVMessageListener è eseguito da un thread diverso da quello che genera i messaggi allora deve essere invocato con il metodo Run(). Tale metodo legge gli eventi dalla coda di funzionamento e li gestisce. NITE offre anche un supporto multi-process basato sull'uso degli oggetti XnVMultiProcessFlowServer e XnVMultiProcessFlowClient.

2.3.3 Tracking della mano

Sebbene OpenNI sia in grado di monitorare i movimenti della mano, NITE mette a disposizione degli algoritmi per facilitare i programmatori ad individuare un punto che identifichi il palmo della mano. Difatti è OpenNI che si occupa di individuare il palmo della mano all'interno della scena e di assegnargli un. Tale punto viene memorizzato all'interno di una struttura dati in modo da fornire

costantemente le coordinate nello spazio (X,Y,Z) che possono essere monitorate per fare il tracking dei movimenti dell'utente. Il nodo di OpenNI che si occupa di questo è l'Hand Point Generator.

NITE usa il Point Generatot ricevendo un flusso di punti che instrada e invia al controllo NITE appropriato. Il controllo tradurrà il flusso di punti in azioni specifiche e significative per far funzionare il riconoscimento d'eventi. NITE fornisce molteplici strumenti per facilitare il riconoscimento della mano. La prima cosa da fare è ottenere il controllo del dispositivo per mezzo di un gesto speciale chiamato "gesto di focus". Ci sono due gesti di focus: "click" e "wave". Per il gesto "click" si dovrebbe tenere la mano alzata e rivolgere il palmo verso il sensore con le dita verso l'alto, per poi spingere la mano in avanti e subito tirarla indietro. Il movimento non deve essere troppo lento e neanche troppo veloce; inoltre il movimento click deve essere abbastanza lungo, in particolare la mano dovrebbe spostarsi in avanti di almeno 20 cm. Nel secondo gesto "Wave" si deve sempre tenere la mano alzata e con il palmo rivolto al sensore spostarla più volte da destra a sinistra e viceversa. Sebbene alcune volte agitare la mano davanti la periferica basti per avere il controllo del dispositivo, la PrimeSense consiglia di eseguire almeno cinque movimenti orizzontali sinistra-destra o destra-sinistra per il Wave e di essere ad una distanza non maggiore di 2 metri dal sensore, tenendo la mano all'interno del campo visivo.

Una volta che l'utente è riconosciuto ed ottiene il controllo del dispositivo (nessun altro potrà ottenerlo), egli lo manterrà fino a quando non nasconde la mano o la porta fuori del raggio d'azione del sensore. Questi gesti devono esser eseguiti con particolare attenzione in quanto se l'utente non avrà il controllo del dispositivo non potrà mai interagire con i programmi che si basano sull'interazione naturale.

La difficoltà che una persona può avere nel rapportarsi con il Kinect per la prima volta è molto simile a quella che le persone hanno avuto nell'utilizzare per la prima volta una tastiera, un mouse oppure un touch screen. Come non si riuscirebbe ad inviare una mail battendo tasti a caso su una tastiera, così non si può sperare di interagire con il Kinect muovendo le mani davanti al sensore senza tracciare traiettorie precise.

Nella figura 13 è presente una mappa di profondità e si può notare come vengono monitorati ben tre punti mano[14]. In generare è possibile monitorare anche più punti.

[14] File Nite.ini (C:\Programmi\PrimeSense\NITE\Hands\Data), modificarlo togliendo i commenti:
[HandTrackerManager]
AllowMultipleHands=1
TrackAdditionalHands=1

Figura 13: Riconoscimento e tracciamento delle mani

Gli oggetti OpenNI che vengono utilizzati sono un generatore di mappa di profondità DepthGenerator, un generatore di mani HandsGenerator, ed un Context. Gli oggetti NITE utilizzati sono un Manager per la sessione XnVSessionManager ed un ascoltatore di messaggi XnVFlowRouter. Un XnVFlowRouter è un ascoltatore di messaggi che internamente ha un altro ascoltatore. Ogni messaggio che arriva viene inviato all'ascoltatore interno, chiamato anche Active Control. L'Active Control può essere modificato per creare un automa interno dell'applicazione. Oltre agli oggetti NITE ed OpenNI si utilizzerà anche un'altra classe di oggetti (XnVPointDrawer) che fornirà gli strumenti per memorizzatore i punti della mano e disegnarli su video.

Oltre all'inclusione delle librerie necessarie all'utilizzo di questi oggetti, alla creazione di funzioni per gestire finestre OpenGL, alla cancellazione di tutti gli oggetti creati nel programma, uno dei passaggi fondamentali è la creazione delle chiamate per gestire le azioni che si dovranno compiere nel momento in cui la sessione ha inizio o fine.

Inoltre è necessario specificare cosa deve succedere una volta che l'Hand Point Generator di OpenNI ha riconosciuto una mano all'interno della mappa di profondità. Le chiamate in questione hanno la stessa struttura di quelle osservate precedentemente quando sono state descritte le sessioni. Il programma inoltre dovrà conoscere la directory del file XML Sample Tracking e dovrà creare una variabile destinata a gestire gli errori che potranno verificarsi durante l'interfacciamento con il sensore e l'uso delle librerie OpenNI.

Ogni metodo messo a disposizione da OpenNI, come la lettura del file Sample Tracking, restituirà una stringa che identifica un possibile successo oppure un possibile errore, (XnStatus rc = XN_STATUS_OK;). Oltre al tipo di errore, sarà possibile trovare anche una sua breve descrizione grazie alla chiamata xnGetStatusString(rc).
Questo passaggio, anche se può sembrare superfluo, sarà una delle operazioni più frequenti che dovrà essere ripetuta ogni volta che il Context verrà utilizzato per leggere un file XML, controllare se esiste una mappa di profondità o un generatore di mani, controllare lo stato della sessione o se i dati sulla profondità e sui punti mano vengono generati correttamente. Trattandosi semplicemente di controllare se la variabile rc è diversa da XN_STATUS_OK, ometteremo il passaggio nelle prossime righe.

```
// Inizializzo OpenNI
rc = g_Context.InitFromXmlFile(SAMPLE_XML_PATH);
/*Cerco se esiste un nodo e restituisce il primo trovato: in qesto caso    mappe di
profondità*/
rc = g_Context.FindExistingNode(XN_NODE_TYPE_DEPTH, g_DepthGenerator);
//Cerco se esiste un nodo e restituisce il primo trovato, generatore di mani
rc = g_Context.FindExistingNode(XN_NODE_TYPE_HANDS, g_HandsGenerator);
// Inizio a generare I dati
rc = g_Context.StartGeneratingAll();
```

Si farà riferimento al Context OpenNI e verranno utilizzati i sui metodi per: leggere il file Sample Tracking, controllare l'esistenza del nodo che genera la mappa di profondità e individuare la mano. InitFromXnlFile apre il file nella directory definita in SAMPLE_XML_PATH[15], legge i dati di licenza della Prime Sense per le librerie NITE e controlla quali sono i nodi a disposizione.
Saranno presenti Depth, Gesture ed Hands. Con il metodo FindExistingNode ci accerteremo successivamente che i nodi che ci interessano esistano. Il metodo FindExistingNode prende in ingresso due parametri: il primo specifica il tipo di nodo d'interesse, il secondo un generatore. La funzione controlla se il generatore che passiamo come secondo parametro corrisponde al tipo dichiarato. Se è vero restituirà lo status ok. L'ultimo metodo StartGeneratingAll() è la chiamata che viene utilizzata per generare dati su cui si andrà a lavorare. La reale posizione di

[15] #define SAMPLE_XML_PATH "../../../Data/Sample-Tracking.xml"

questo ultimo metodo è alla fine del main prima che si invochi una funzione che manderà in loop la richiesta di dati dal sensore.
Abbiamo già visto come si crea e si gestisce una sessione, di seguito verrà creato un Manager per gestirla e verrà inizializzato specificando il Context ed i gesti da fare per far partire la sessione. In questo caso il gesto click corrisponde ad una piccola spinta in avanti, mentre Wave corrisponde ad un leggero movimento della mano che va da destra a sinistra e viceversa. Nel codice viene omesso il controllo degli errori che andrebbe dopo l'inizializzazione. L'ultima cosa che faremo sarà registrare la sessone specificando quattro parametri. Il primo è il contesto da usare quando si invocheranno le funzioni (NULL se usato nel main, this se viene gestito all'interno di una classe). I restati tre parametri sono appunto le funzioni che verranno invocate nel momento in cui la sessione ha inizio, la sessione termina o la sessione si blocca ma può essere riavviata.

```
//Lavoro con NITE
//Il manager per le sessioni
g_pSessionManager = new XnVSessionManager;
/* Inizializzo la sessione, passando il tracker da usare per il monitoraggio mani il
gesto da usare come focus ed il resto da usare come refocus */
rc = g_pSessionManager->Initialize(&g_Context, "Click,Wave", "RaiseHand");
// Registo le chiamate di sessione
g_pSessionManager->RegisterSession(NULL, SessionStarting, SessionEnding,
FocusProgress);
```

Nella successiva riga verrà creato un oggetto Drawer, che con i suoi metodi ci aiuterà a tenere traccia di tutti i punti mano che verranno monitorati e li mostrerà a video come un puntino seguito da una scia. La scia di punti che segue il punto principale rappresenta la storia del punto. Qualora si volesse creare una nuova classe simile ma con altre funzionalità è qui che deve essere istanziato l'oggetto.
È possibile creare anche un oggetto di una classe già presente nelle librerie NITE come il controllo del gesto cerchio, oppure il controllo del gesto Push. Qualsiasi sia l'oggetto, se ha delle funzionalità per monitorare i punti o riconoscere i gesti dovrà essere creata in questo livello del main e dovrà seguire le stesse procedure che segue l'oggetto Drawer.

```
g_pDrawer = new XnVPointDrawer(20, g_DepthGenerator);
```

L'oggetto Drawer, una volta che il programma è in esecuzione, inizierà a fornire dati sulla posizione dei punti. Per raccogliere tutti questi dati viene utilizzato l'ascoltatore di messaggi XnVFlowRouter. Una volta creato viene settato attivo sull'oggetto che dovrà ascoltare ed a sua volta verrà inserito nella lista del SessionManager.

```
//Creo un instradatore di flusso
g_pFlowRouter = new XnVFlowRouter;
//Setto attivo l'ascoltatore di messaggi che si occopa di XnVPointDrawer
g_pFlowRouter->SetActive(g_pDrawer);
//Per il Message Generator, passo l'evento che devo ascoltare
g_pSessionManager->AddListener(g_pFlowRouter);
```

Da questo momento la sessione non dipenderà solo dai gesti definiti col metodo Initialize, ma dipenderanno anche dal ciclo di vita dell' oggetto Drawer e dal suo ascoltatore. Come è stato fatto per la sessione si andrà a registrare anche l'oggetto Drawer specificando il contesto e la funzione che dovrà essere invocata se si verifica un determinato evento.

```
/* I due metodi seguenti sono della classe creata dall'utente XnVPointDrawer, ereditati
da XnVPointControl. Registro una chiamata per quando non abbiamo punti */
g_pDrawer->RegisterNoPoints(NULL, NoHands);
//Registro una chiamata per settare la mappa di profondità
g_pDrawer->SetDepthMap(g_bDrawDepthMap);
```

Per concludere basterà settare la mappa di profondità, iniziare a generare i dati usando il Context ed eseguire una funzione che mandi il main in loop.

2.3.4 Riconoscimenti dei gesti

I gesti che possono esser riconosciuti con le librerie NITE sono stati già presentati e la loro implementazione in un applicativo non si discosta molto dall'esempio appena esposto. La lista dei movimenti è la seguente: Push, Swipe, Steady, Wave, Circle, SelectableSlider1D e SelectableSlider2D.
Ogni gesto fa riferimento ad una classe che dovrà esser inclusa se si vuole lavorare con esso. Ad esempio includeremo XnVCircleDetector.h per il gesto Circle, XnVPushDetector.h per il gesto Push e così via per tutti gli altri.

I gesti per esser utilizzati inoltre avranno bisogno di un oggetto che si occupi di cercare particolari movimenti tra i dati trasmessi dal sensore e necessiteranno anche di appropriate chiamate per sapere quale effetto nel programma dovrà esser associato al riconoscimento di un gesto, proprio come viene associato al click del mouse l'apertura di una cartella e nessun effetto se semplicemente ci si passa sopra con il puntatore.

```
//Creo un riferimento ad gestore di evento circlo
XnVCircleDetector*  g_pCircle = NULL;
//Questa funzione viene invocata se l'utente fa il gesto cerchio
void XN_CALLBACK_TYPE CircleCB(XnFloat fTimes, XnBool bConfident,
            const XnVCircle* pCircle, void* pUserCxt)
{
  //Scrivo qui cosa fare se l'utente ha fatto il gesto cerchio
}
// Questa funzione viene invocata se il gesto cerchio non viene riconosciuto
void XN_CALLBACK_TYPE NoCircleCB(XnFloat fLastValue,
                  XnVCircleDetector::XnVNoCircleReason reason, void * pUserCxt)
{
  // Scrivo qui cosa fare se l'utente non ha fatto il gesto cerchio
}
```

Trascurando i dettagli delle altre CALLBACK, il resto della gestione dei gesti va trattato nel main oppure in una classe creata ad hoc (come XnVPointDrawer). Anche questa volta bisognerà creare un oggetto che svolga il ruolo di controllore per sapere quando il gesto si verifica, inoltre registreremo le CALLBACK apposite per associare un evento nel programma ad un gesto effettuato.

```
//Creo un controllore per il gesto cerchio
g_pCircle = new XnVCircleDetector;
// CALLBACK – Il primo parametro indica il contesto, il secondo la funzione da invocare
g_pCircle->RegisterCircle(NULL, &CircleCB);
g_pCircle->RegisterNoCircle(NULL, &NoCircleCB);
g_pCircle->RegisterPrimaryPointCreate(NULL, &Circle_PrimaryCreate);
g_pCircle->RegisterPrimaryPointDestroy(NULL, &Circle_PrimaryDestroy);
//Per il Message Generator, passo l'evento che devo ascoltare
g_pSessionManager->AddListener(g_pCircle);
```

Infine dato che la sessione dipenderà anche dall'evento Circle includeremo il controllo del gesto Circle nella lista del SessionManager. A questo punto potrebbe essere superfluo sottolineare che qualsiasi altro gesto può esser integrato allo stesso modo.

2.3.5 Tracking del corpo umano

Per utilizzare il Kinect per tale funzione è necessario ricordare che il raggio d'azione del sensore è compreso tra 1 e 3,5 metri. Se siamo troppo vicini al sensore, oppure troppo lontani, sarà difficile riuscire ad avere un buon tracking dello scheletro.
Innanzi tutto si deve assegnare un ID all'utente, questo ID è persistente e permetterà di riconoscere diversi utenti in scena. Questo processo di assegnazione di ID viene chiamato segmentazione degli utenti e restituisce una mappa delle etichette assegnate. L'algoritmo di monitoraggio dello scheletro andrà ad utilizzare questa mappa per generare uno scheletro. Ogni possibile errore nella segmentazione dell'utente potrà influire sulla posa di calibrazione.
I fattori che possono dare problemi alla fase di segmentazione ed alla fase di calibrazione sono:

- Utenti troppo vicini che si toccano tra di loro.
- Oggetti (come sedie o tavoli) troppo vicini che non permettono il corretto riconoscimento del corpo.
- Utenti in movimento che non mantiene la posa di calibrazione.
- Spostamento del sensore mentre la fase di segmentazione è attiva.

Si ottiene così un ID per ogni pixel che rappresenta un determinato utente. La figura 14 rappresenta uno dei possibili problemi appena descritti, l'utente nella scena tenendo la mano sullo schienale della sedia ha portato il processo di segmentazione ad assegnare il proprio ID anche ad i pixel che raffigurano la sedia. Il problema in questo caso non è permanente: non bisognerà riavviare l'applicazione ma semplicemente togliere la sedia dal campo visivo e verrà ricreata una nuova mappa. Il colore blu è un feedback visivo per l'utente per capire che quei pixel della mappa di profondità appartengono a lui.

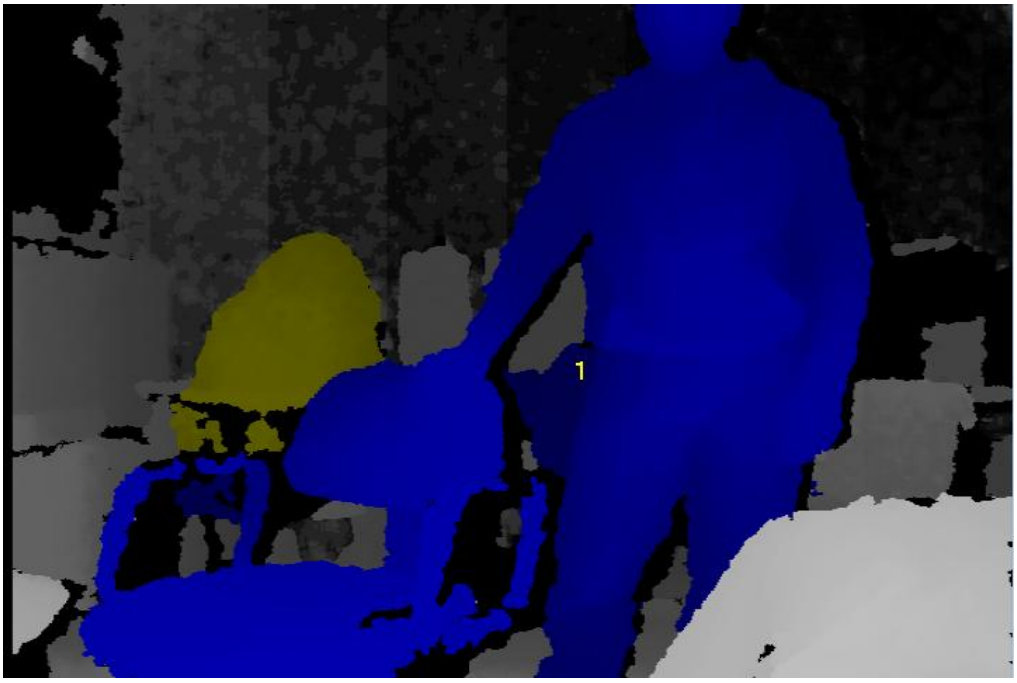

Figura 14: R*isultato della fase di segmentazione*

Si può notare sullo sfondo come un'altra figura è stata colorata di giallo. Sebbene la forma sullo schermo non sembri una figura umana l'algoritmo di segmentazione degli utenti le ha assegnato un ID in quanto la ragazza sullo sfondo era in movimento durante il test e l'algoritmo di segmentazione ha escluso la sua forma dall'insieme degli oggetto inanimati.

Nel momento in cui ci sono più utenti in scena può succedere che uno impedisca al sensore di monitorare gli altri. Ad esempio se due si mettono in fila indiana, l'ID del primo viene perso e successivamente dovrà essere riassegnato, oppure i due ID potrebbe anche essere scambiati. Inoltre se un utente esce dalla scena per più di 10 secondi il suo ID viene cancellato e potrà essere assegnato ad un altro utente in un secondo momento, in quanto gli ID sono riciclabili.

Dopo che un utente è stato individuato segue la fase di calibrazione, necessaria per poter monitorare in un secondo momento lo scheletro. In questa fase è richiesto che l'utente mantenga per qualche secondo una determinata posa, chiamata posa di calibrazione. L'utente deve esser rivolto verso il sensore e deve alzare le mani, in particolare i gomiti devono disegnare un angolo di 90 gradi e le mani devo essere allo stesso livello della testa. Per avere un idea più precisa si può osservare la figura 15. Come si può vedere le ginocchia non devono essere piegate inoltre la testa e le braccia devono essere visibili.

Una volta che la calibrazione è terminata si avrà un completo monitoraggio delle articolazioni dello scheletro; in particolare saranno monitorati: testa, collo, spalla

destra e sinistra, gomito destro e sinistro, torso, anca destra e sinistra, ginocchio destro e sinistro ed infine piede destro e sinistro.
Per capire come il corpo viene monitorato dobbiamo soffermarci un istante sul sistema di coordinate e capire cosa si intende per destra e sinistra.

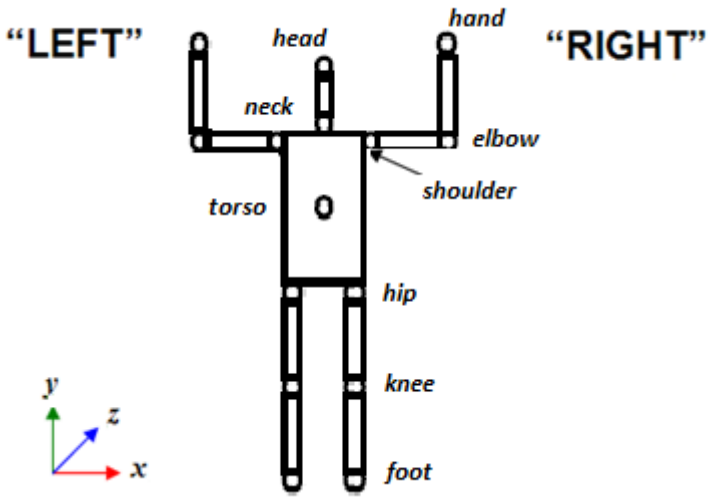

Figura 15: *Posa di calibrazione*

Le posizioni e l'orientamento sono dati dal sistema di coordinate del mondo reale e l'origine di tale sistema di coordinate è il sensore.
Gli assi X,Y,Z sono rappresentati nella figura che mostra la posa di calibrazione. Gli orientamenti sono memorizzati in una matrice di rotazione 3X3 (ortonormale). Le tre colonne rappresentano rispettivamente la direzione dell'asse X, Y e Z. Durante la posa T il nostro orientamento verrà rappresentato come una matrice identità, in quanto l'orientamento di ogni articolazione è allineato con il sistema di coordinate.
Supponiamo ora di restare fermi davanti al sensore con una posa che rappresenti una T, se osserviamo la mappa di profondità su uno schermo, quello che stiamo osservando è un riflesso speculare di noi stessi.
Se muoviamo la mano destra noi vedremo una mano che si muove sul lato destro dello schermo, ma dal punto di vista dello scheletro in realtà quella che si sta muovendo è la mano sinistra. Quindi nell'etichettatura tutto ciò che è stato etichettato destro è in realtà il lato sinistro dello scheletro.
La figura 16 mostra la fase di tracking dello scheletro; come si può vedere il disegno sull'utente che rappresenta lo scheletro umano segue il movimento degli

arti. La fase di tacking è ancora lenta, può quindi capitare che se l'utente si muove troppo velocemente il movimento dello scheletro sullo schermo avrà un ritardo.

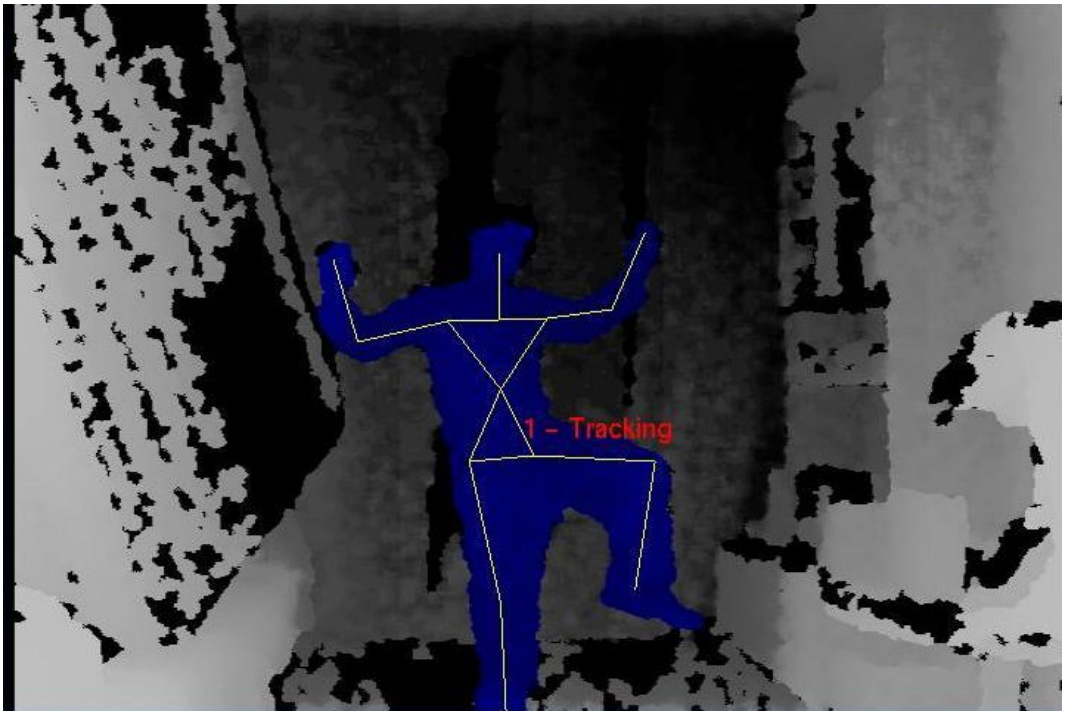

Figura 16: *Esempio di tracciamento dello scheletro*

Va sottolineato che i dati forniti dal tracciamento dello scheletro oltre ad essere utilizzati in tempo reale in diverse applicazioni e giochi, possono essere registrati in un formato proprietario .ONI o con appropriate modifiche sul codice sorgente della demo NITE esportate in formato .bvh[16].

Quest'ultimo formato in particolare viene utilizzato per il motion capture, ciò implica che con una modesta spesa per l'acquisto del sensore Kinect e l'uso di librerie open source si può avere un sistema di mocap casalingo perfettamente funzionante. Per ottenere i file BVH non si necessiterà di tute nere, led, green screen, accelerometri, telecamere e software costosi ma semplicemente dei semplici strumenti fin ora descritti. I file BVH così ottenuti potranno poi esser importati in 3ds Max (come in figura 17) per creare il set di animazioni destinate a videogiochi, simulazione di folle o effetti speciali. Un'altra soluzione più

[16] BVH (Biovision Hierarchy) è un formato sviluppato da Biovision per le animazioni. Fornisce informazioni sullo scheletro, dal 2008 il formato BVH è ampliamente utilizzato. Alcuni software che permettono l'importazione del formato sono: Lightwave 3D, 3ds Max, Blender, DAZ Studio, Maya e Poser.

efficiente sarebbe quella di utilizzare il kinect con 3ds Max e Autodesk MotionBuild, utilizzando anche il software open source Brekel[17] Kinect 3D Scanner Setup e Brekel Kinect MoBu Device.

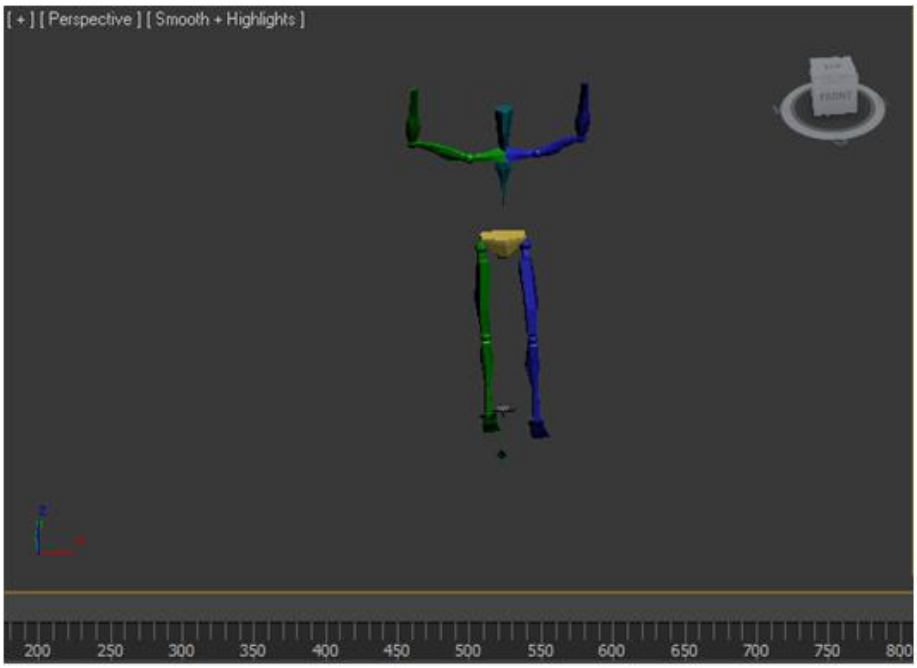

Figura 17: *Animazione su 3Ds Max con BHV acquisito dal tracciamento scheletro con Kinect*

2.4 Descrizione dell'ambiente grafico

Il framework utilizzato per la realizzazione dell'ambiente virtuale e per la gestione dell'interazione e della navigazione è Instant Reality realizzato da Mixed reality che offre un interfaccia per l'uso di uno strumento di rendering ed interazione 3D tra l'utente ed il calcolatore. Questo framework offre un player per la visualizzazione della scena in 3D scritta con linguaggio VRML (14). VRML (Virtual Reality Modeling Language) (15) a differenza di OpenGL è un linguaggio ad altro livello che permette la simulazione di mondi virtuali tridimensionali. È possibile descrivere ambienti virtuali contenenti oggetti, fonti di luci, suoni e filmati. Tali mondi possono essere animati, possono esser esplorati da utenti multi ed essere interattivi. I documenti VRML come tutti i documenti ipertestuali prevedono l'impiego di link a URL remoti e ad altri file .wrl (formato dei file VRML). Il VRML si può pensare come Virtual Reality Markup Language analogo di HTML e dei linguaggi che fanno uso di marcatori nel proprio

[17] Link del sito per download e tutorial http://www.brekel.com/?page_id=170.

documento. In seguito è stata cambiata la parola Markup in Modeling per evidenziare la modellazione grafica del linguaggio. Il file .wrl è un file di testo che utilizza caratteri ASCII ed al suo interno contiene i comandi per descrivere una scena tridimensionale. Sebbene esistono numerosi programmi di rendering che permettono di avere modellazione tridimensionale di qualità superiore la caratteristica che rende il VRML interessante è la possibilità di pubblicare i file su internet ed accedervi tramite URL. Essendo Internet accessibile da qualsiasi macchina e da qualsiasi piattaforma, come i documenti HTML anche i documenti VRML possono essere sostanzialmente visitati allo stesso modo da macchine Windows, Max, Unix, Linux, ecc.

L'utilizzo combinato del sensore con VRML permetterebbe quindi non solo all'utente di interagire in un mondo virtuale ma allo stesso tempo di navigare su Internet. La realtà virtuale realizzata potrebbe contenere aspetti puramente video ludici oppure anche elementi pubblicitari. Sarebbe interessante per un visitatore andare su un sito di automobili e navigare ed interagire con un concessionario virtuale in 3D per mezzo del Kinect o dispositivi simili.

Uno degli aspetti considerati negativi per interazione con ambienti virtuali su internet è il fatto che si cerca di interagire con un mondo 3D utilizzando un dispositivo di navigazione 2D come il mouse. Sebbene questa soluzione è adeguata per navigare in una pagina HTML richiede maggiore tempo per imparare a navigare in uno spazio 3D con una periferica nata per il 2D. Con l'utilizzo di un mouse a 3 dimensioni che svilupperemo nelle prossime pagine, questo aspetto negativo potrà essere trascurato, mentre un altro fattore critico per i file VRML dipenderà dall'evoluzione tecnologica.

I file VRML sono notevolmente più pesanti di un semplice documento HTML ciò porta alla necessità di disporre di una linea a banda larga o di ridurre la qualità del rendering al fine di permettere a più utenti di navigare nel mondo virtuale. Dopo la nascita di VRML sono stati sviluppati molti strumenti software per generare ambienti tridimensionali. In alcuni casi i prodotti erano creati ad hoc per questo scopo, mentre in altri casi sono stati modificati programmi per il 3D esistenti per aggiungere nuove funzionalità, tra cui la possibilità di esportare in formato wrl, come ad esempio 3D Studio Max. Un esempio di un possibile ambiente 3D è presentato in figura 18.

Essendo VRML in grado di gestire solo eventi interni al mondo virtuale, la parte di controllo dell'interazione del giocatore deve avvenire in Java, come dovrebbe anche avvenire in Java la gestione delle connessioni per un sistema multi-utenza.

Figura 18: *Esempio di mondo 3D realizzato con VMRL*

Linguaggi di scripting come VrmlScript o JavaScript non sarebbero d'aiuto, per questo è stato dotato VRML 2.0 di un'interfaccia per il linguaggio di programmazione Java.
Una delle caratteristiche principali di Java è l'indipendenza dalle piattaforme, quindi le applicazioni create con il linguaggio Java potranno essere eseguite sui sistemi DOS, Windows, Linux e Unix. I programmi creati potranno esser eseguiti dall'interprete JVM (Java Virtual Machine) di Java che cambia a secondo del sistema operativo su cui stiamo lavorando. Un'altra caratteristica fondamentale del Java è che permette di creare due tipi di programmi: le applicazioni e gli applets. La differenza è che gli applets vengono inseriti in pagine HTML e vengono visualizzati dagli utenti all'interno di browsers Web.
Esistono due modi per far interagire Java con VRML: attraverso JSAI (Java in Script Nodes Authoring Interface) o EAI (External Authoring Interface).
Il primo modo che prevede l'uso del nodo script è quello originariamente scelto per la versione VRML 2.0 e si basa sul seguente principio.
Una classe Java viene attivata da un opportuno nodo di Script presente nel mondo VRML, questo nodo indicherà la classe che deve essere attivata in fase di caricamento nel mondo ed i campi che possono essere modificati dalla classe.
Le classi Java collegate al mondo VRML non girano quindi come applets ma sono definite come estensioni della classe Script. Per poter accedere ai campi VRML indicati nello Script esistono dei packages[18], che vengono forniti con il browser

[18] Col termine packages si intendono collezioni di classi

VRML, con i quali è possibile accedere ad un campo del mondo VRML e cambiarne i valori.
Il secondo metodo tramite interfaccia esterna EAI ci permette di controllare un mondo virtuale da un applet Java. Questa volta l'interazione non è inizializzata nel mondo VRML ma è l'applet stessa che provvede ad ottenere un riferimento al browser VRML e che permetterà di utilizzare ulteriori references ai nodi presenti nel mondo 3D. Anche per il metodo EAI esistono appositi packages.
Java può essere usato quindi per aumentare la potenza di VRML oppure come strumento di output grafico. Una cosa fondamentale per l'interazione con Java è procurarsi i packages VRML, in particolare è necessario il kit JDK[19] (Java Development Kit), che permette di realizzare programmi ed applet e fornisce il compilatore che consente di tradurre il codice sorgente .java in codebyte .class. Per questo lavoro è stato utilizzato il jdk 1.6.0. L'installazione è molto semplice l'unica accortezza è creare le variabili d'ambiente necessarie aggiungendo in PATH la directory bin ad esempio C:\ProgramFiles\Java\jdk1.6.0_23\bin e controllare l'esistenza della variabile d'ambiente Java home.
Nella seconda parte del prossimo capitolo daremo ulteriori informazioni sull'interazione tra Java e VRML collegando un mouse 3D per la navigazione e l'interazione in un ambiente virtuale.

[19] Le JDK sono scaricabili gratuitamente dal sito http://www.javasoft.com/.

Capitolo 3
Realizzazione di un ambiente virtuale interattivo

3.1 Movimento nell'ambiente 3D

Il sistema che è stato realizzato ha l'obbiettivo di permettere l'interazione con un ambiente virtuale, quindi la costruzione di un'interfaccia d'interazione è alla base del progetto. Dovremmo definire inoltre un insieme di operazioni che saranno disponibili, capire l'evoluzione dello stato del sistema ed assicurarci che il suo uso non sia complicato. Per soddisfare tutti questi bisogni andremo a creare un insieme di requisiti che il sistema dovrà soddisfare alla fine della progettazione.
Sappiamo a questo punto quali sono gli strumenti a disposizione per il programmatore, ora dobbiamo chiederci quali saranno gli strumenti di cui potrà disporre l'utente che vorrà usare il sistema.

- Per prima cosa dovrà possedere un sensore Kinect e dovrà essere funzionante sul proprio PC ed in secondo luogo dovrà disporre di una copia dell'ambiente grafico che forniremo per l'interazione.
- Dovremmo fornire all'utente un alfabeto di gesti che vengano correttamente riconosciuti dal sensore e tradotti in dati di input per interagire correttamente con il programma.

- Dovremmo far si che i gesti tradotti in input non siano confondibili dal sistema e che ad ogni movimento corrisponde una ben precisa azione.
- Il sistema dovrà essere veloce in particolare nella fase di calibrazione e tracciamento del punto. L'utente deve sentirsi a suo agio interfacciandosi con l'applicazione e deve trovare un riscontro pratico ai suoi movimenti.
- L'alfabeto di gesti non dovrà essere poco chiaro o scomodo da usare, in quanto l'interazione per l'utente deve essere il più possibile comoda e non dovrà essere faticosa durante l'uso. (Non è il caso di far saltare l'utente sul poso per scorrere in alto le pagine di un documento)

Questi requisiti verranno soddisfatti durante l'esposizione del progetto, inoltre aggiungeremo a questa lista ulteriori requisiti, non appena si presenteranno nuovi problemi da risolvere. Nelle prossime pagine cominceremo a descrivere il sistema in tutte le sue parti e spiegheremo il principio di funzionamento.

3.2.1 Architettura del sistema

Prima di presentare come verrà organizzato il codice, bisognerà descrivere il sistema.

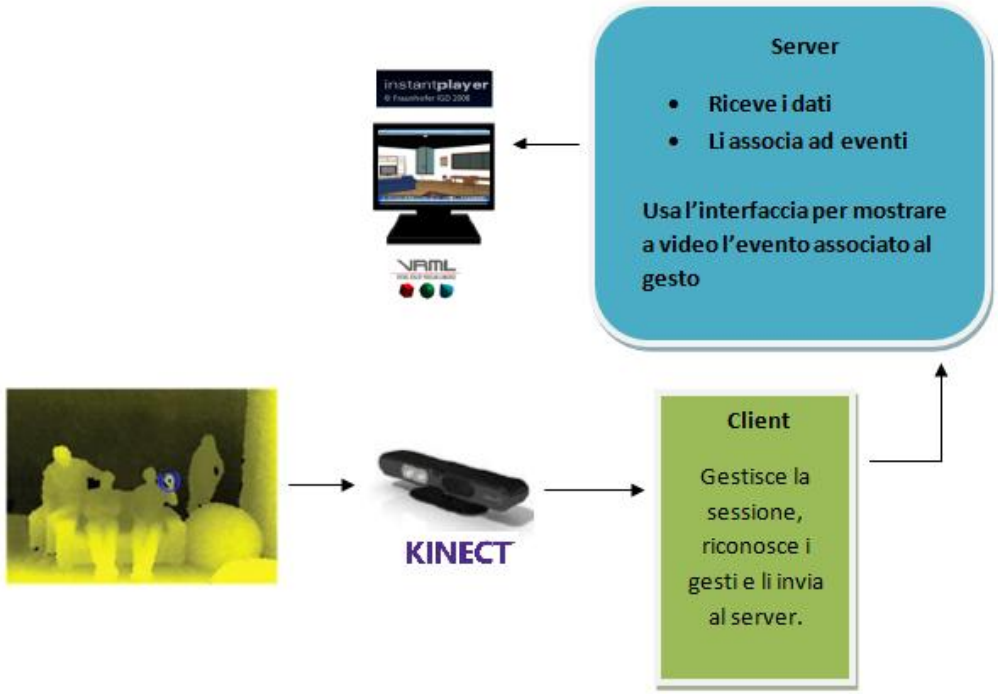

Figura 19: *Architettura Client-Server*

Per prima cosa si deve chiarire che lavoreremo su un architettura Client-Server (mostrata in figura 19), ed i dati verranno inviati dal client al server per mezzo di una Socket.
Questa scelta è stata inizialmente fatta per utilizzare un ambiente virtuale già sviluppato che avrebbe permesso in un secondo momento di confrontare l'interazione per mezzo del Kinect con un altro sistema sviluppato all'interno del RIMLab (Robotics and Intelligent Machines Laboratory) dell'università di Parma. Questo sistema si basava sull'interazione tra il Wii remote ed un guanto munito di diodi led per riconoscere gesti successivamente utilizzati per la navigazione in un ambiente virtuale. Per tale ragione non è stato costruito un nuovo ambiente virtuale utilizzando un motore grafico come Irrlicht, che avrebbe permesso l'uso di un solo linguaggio di programmazione, ma i dati elaborati dal Client sono stati inviati per mezzo di una Socket ad un applicativo Java per la gestione del motore grafico. Questa scelta progettuale ha però presentato un secondo vantaggio; le funzioni e le classi che realizziamo per interfacciarci con il sensore sono totalmente indipendenti dalla visualizzazione grafica, quindi i dati inviati via Socket un domani potranno esser destinati ad altre applicazioni. Ad esempio potranno esser utilizzate per muovere un puntatore su Desktop oppure essere una base di partenza per manovrare un braccio meccanico.

Dalla figura 20 si può osservare il diagramma di flusso relativo al Client, che mostra il principale ciclo di attività dell'utente e della sessione. Come si può vedere il Client si occupa principalmente di gestire la sessione e tenere monitorizzati i diversi movimenti della mano, alcuni di questi movimenti verranno riconosciuti dagli algoritmi NITE, altri invece verranno riconosciuti da una classe creata ad hoc. Questa seconda classe si occuperà anche di inviare i dati al Server utilizzando la Socket.

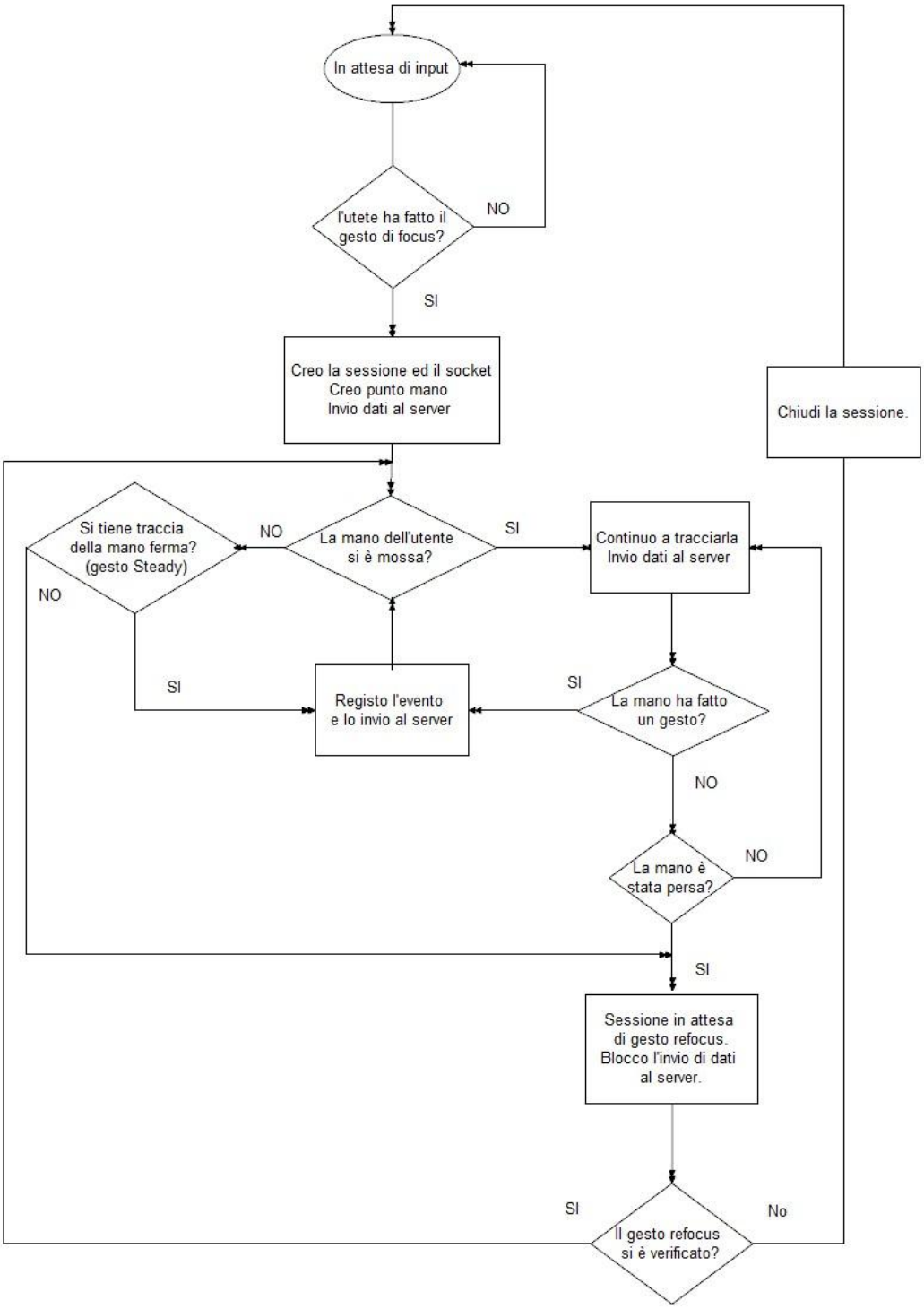

Figura 20: *Diagramma di flusso del Client*

3.2.1 Riconoscimento di gesti

Non avendo stabilito durante la fase di progettazione quali gesti sarebbero stati associati a determinate funzionalità la classe XnVPointDrawer è stata dotata di tutti i rilevatori di gesto a disposizione.

```
/* Create internal objects
      Un ascoltatore di messaggi
      Un controllore del movimento spinta
      Un controllore del movimento colpo
      Un controllore del movimento steady
      Un controllore del movimento wave
      Un controllore del movimento cerchio - disattivato
*/
      m_pInnerFlowRouter = new XnVFlowRouter;
      m_pPushDetector = new XnVPushDetector;
      m_pWaveDetector = new XnVWaveDetector;
      m_pSwipeDetector = new XnVSwipeDetector;
      m_pSteadyDetector = new XnVSteadyDetector;
      g_pMainSlider = new XnVSelectableSlider1D(3);
      //m_pCircleDetector = new XnVCircleDetector;
```

Solo in un secondo momento è stato deciso di disattivare il controllore del gesto Circle in quanto non è stato riscontrato nessun tipo di utilizzo concreto all'interno dell'applicazione. Dato che ogni movimento della mano viene monitorato per mezzo di un solo punto, osserveremo il moto del punto nello spazio per capire quando un gesto si verifica e faremo delle brevi considerazioni progettuali per sottolineare eventuali combinazioni critiche di gesti.
Prendiamo in considerazione ad esempio il gesto Wave già presentato in precedenza. Sappiamo che questo gesto serve come input al programma per iniziare a monitorare la mano. Nel momento in cui l'utente muove la mano da sinistra a destra o viceversa per un numero n di volte, il movimento viene riconosciuto e registrato all'interno di un oggetto chiamato Bridge.
Questo oggetto si occupa di fare da ponte tra le due applicazioni in C++ e Java. I dati vengono costantemente inviati al Bridge che cambierà di volta in volta il suo stato e con la stessa periodicità invierà i dati al Server. Possiamo osservare la porzione di codice che si occupa di svolgere questo compito.

```
// Per ogni mano nella scena
for (PointIterator = m_History.begin();
      PointIterator != m_History.end(); ++PointIterator){
      // Pulisi il buffer
      XnUInt32 nPoints = 0;
      XnUInt32 i = 0;
      XnUInt32 Id = PointIterator->first;
//Vai su tutte le posizioni precedenti della mano corrente
      std::list<XnPoint3D>::const_iterator PositionIterator;
      //Per ogni punto della mano
      for (PositionIterator = PointIterator->second.begin();
            PositionIterator != PointIterator->second.end();
            ++PositionIterator, ++i){
```

```
                // Aggiungi la posizione al buffer
                XnPoint3D pt(*PositionIterator);

                //Se sono nel primo punto della prima mano
                if(onehand == 0 & (i-1)== 0)
                {
                      //Passo un boolean
                      m_Bridge->SetPush(statepush);
                      m_Bridge->SetSteady(statesteady);
                      m_Bridge->SetWave(statewave);
                      m_Bridge->SetUp(stateup);
                      m_Bridge->SetDown(statedown);
                      m_Bridge->SetFarward(statefarward);
                      m_Bridge>SetBackward(statebackward);
                      //Passo un float
                      m_Bridge->SetSlider(stateslider);
                      //Passo un punto
                      m_Bridge->SetWalk(pt);
                      m_Bridge->SetRotation(pt);
                      //Ripristino gli stati di default
                      statepush=statewave=false;
                      statesteady=stateup=statedown=false;
                      statebackward=statefarward=false;
                      stateslider = -1;
                      //Invio i dati al server
                      m_Bridge->SendData(lhSocket);

                }
          . . .
```

Il codice mostra come in ogni istante settiamo lo stato dell'oggetto Bridge ed invochiamo il metodo SendData per trasmettere lo stato del Bridge al programma Java. È logico pensare che in alcuni istanti se si verifica il gesto Up, avremo per Up il valore booleano true e per Down false. Ma cosa accade se i due gesti sono molto simili? Si può infatti notare che il gesto Wave menzionato precedentemente ha alcune similitudini con lo Slider. Per mezzo dello Slider muovendo la mano davanti al sensore possiamo, a secondo della posizione sull'asse X, identificare la parte sinistra, centrale o destra dello schermo. Per avere un idea più chiara si può osservare la prossima figura.

Sarebbe i due movimenti sono diversi concettualmente sarebbe sbagliato usarli insieme. Immaginiamo di voler navigare all'interno di un menù. Potremmo pensare di spostarci tra i vari elementi usando lo Slider ed uscire dal menù con il gesto Wave. Sebbene l'idea potrebbe risultare corretta questa associazione renderebbe il sistema scomodo o addirittura inutilizzabile. La figura 21 mostra le analogie di movimento tra i gesti Wave e Slider. Nel movimento dello Slider il sensore potrebbe interpretare il movimento della mano come un gesto di Wave e

farci uscire dalla fase di scelta prima ancora di aver osservato tutti gli elementi del menù.

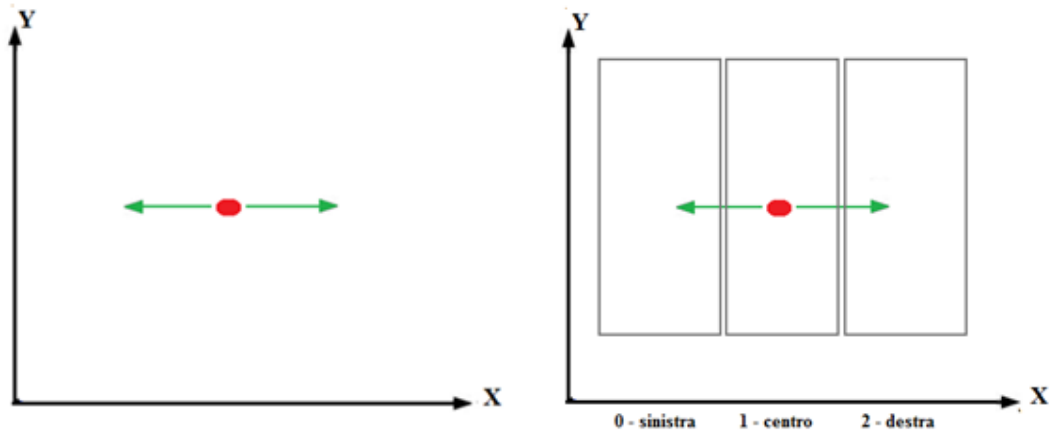

Figura 21: *A sinistra il movimento per il gesto Wave, sulla destra le zone interessate dallo Slider.*

Un idea per risolvere il problema e uscire dal menù senza aver scelto nulla potrebbe essere quella di muovere la mano in alto o indietro così l'asse su cui avviene il movimento di uscita è diverso rispetto a quello utilizzato per i movimenti di scelta. Con lo stesso criterio va scelto il gesto per selezionare gli elementi del menù.

Per svolgere questo compito si potrebbe utilizzare il gesto Push, che corrisponde ad un movimento della mano in avanti. Questo gesto non deve essere obbligatoriamente veloce ed utilizza l'asse Z, quindi è totalmente indipendente sia dall'asse X dello Slider che dall'asse Y di Up per uscire. Si noti che nel caso si scelga un movimento della mano all'indietro per uscire verrà utilizzato sempre l'asse Z ma in direzioni opposte, quindi Push e Backward non andranno mai in conflitto. Il gesto però potrebbe portare l'utente a stancarsi oppure ad avere la tendenza ad avvicinare troppo la mano al sensore, arrivando al punto in cui il gesto non viene più riconosciuto. Questo può verificarsi nel caso l'utente deve effettuare più scelte consecutive. Un'altra possibilità ci viene data da casa Microsoft che ha utilizzato Steady come gesto di selezione. L'utente una volta che ha deciso quale elemento del menù selezionare deve solo tenere la mano ferma sull'oggetto ed attendere che questo venga acquisito.

Sebbene l'idea possa sembrare interessante, per utilizzare questo sistema Microsoft fa forza su un feedback visivo, mostra su schermo una barra circolare di caricamento che aumenta ed indica il tempo rimanente per la selezione.

Per il sistema sono stati testati entrambi i criteri di scelta, ma non avendo a disposizione un feedback visivo e volendo evitare che l'utente si stanchi nel tenere la mano ferma in una determinata posizione per troppo tempo si è deciso di adottare la prima soluzione e quindi utilizzare il gesto Push.
Per i movimenti Up e Down sono stati utilizzati sia lo SwipeDetector che riconosce il movimento rettilineo della mano lungo una specifica direzione che termina in un punto, sia lo Slider. Questa scelta è stata fatta per dare maggiore sicurezza all'individuazione del movimento della mano in alto. Sebbene lo Slider si occupi principalmente di gestire i movimenti per la selezione lungo l'asse X esso è in grado di capire quando la mano si muove in alto, in basso, indietro oppure in avanti e quindi ha direzioni sbagliate rispetto a quelle necessarie per la corretta navigazione nel menù.
Durante lo sviluppo del software si è voluto utilizzare questo "controllo" dello stato dello Slider affinchè il gesto Up e Down venissero sempre riconosiuti. Ora che abbiamo definito quali gesti ci serviranno per interagire con gli oggetti è possibile sviluppare un sistema di navigazione che rappresenta la parte fondamentale del progetto. Questi gesti dovranno poi trovare una corrispondenza nel programma Java che girerà sul server.

3.2.2 Strumenti per la navigazione dell'utente

Uno dei vantaggi che si ha nell'utilizzare un sensore come il Kinect è di poter tracciare il movimento della mano all'interno di uno spazio a tre dimensioni, questo ci permette di realizzare un mouse che non si muove su un piano ma nello spazio. Lo spazio ha come riferimento il sensore, lui rappresenta la coordinata (0,0,0), ma per poterci muovere nelle varie direzione, si necessita di un altro riferimento, nettamente più comodo, come in figura 22.

Figura 22: *Esempio del punto di riferimento per lo spazio*

Per questo durante la fase di calibrazione della mano si è deciso di memorizzare il primo punto che veniva assegnato alla mano ed usarlo come origine degli assi.
Usando il Context per il punto mano (XnVHandPointContext) abbiamo ottenuto il riferimento nel punto reale e lo abbiamo salvato in un vettore.
A questo punto però si presenta un problema, il punto che viene memorizzato non deve essere quello che viene disegnato nello schermo, bensì un punto che fa riferimento al mondo reale. Non potendo usare direttamente i punti nello spazio virtuale bisogna trovare una corrispondenza che lega questi ai punti del mondo reale. Per chiarire questo problema bisogna introdurre e spiegare cos'è una coordinata omogenea (16) e cos'è uno spazio proiettivo (17).
In matematica lo coordinate omogenee[20] sono uno strumento analogo alle coordinate cartesiane della geometria analitica, ma usate per descrivere i punti nella geometria proiettiva. La geometria proiettiva è la parte della geometria che crea modelli per quei concetti che sono legati a prospettiva ed all'orizzonte, definendo e studiando gli enti geometrici usuali come punti e rette senza utilizzare le misure che permettono il confronto di lunghezze.

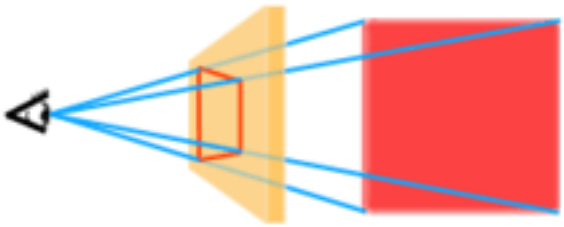

Figura 23: *Esempio per la geometria proiettiva*

Si può pensare alla geometria proiettiva come la geometria che nasce nel collocare il proprio occhio in un punto dello spazio, così che ogni linea che intersechi l'occhio appaia come un punto (si ha un esempio in figura 23). Dato che l'intero mondo viene visto da un solo occhio, non si possono avere informazioni sulla profondità. L'assenza d'informazioni sulla profondità porta all'assenza di strumenti per quantificare direttamente le grandezze degli oggetti in uno spazio proiettivo. Va sottolineato che l'orizzonte nello spazio proiettivo è parte integrante dello spazio; inoltre dalla geometria piana proiettiva abbiamo la conseguenza che due rette si intersecano sempre e non sono mai parallele.

[20] Introdotto nel 1837 dal matematico ed astronomo tedesco August Ferdinand Möbius, la cui notorietà è dovuta principalmente alla scoperta del nastro di Möbius, una superficie bidimensionale immersa in uno spazio tridimensionale euclideo che presenta una sola linea di bordo e una sola faccia.

Volendo dare una definizione più formale dello spazio proiettivo possiamo per prima cosa dare una definizione delle coordinare omogenee che sono ampiamente usate nell'arte digitale per la rappresentazione di oggetti e dei loro movimenti nello spazio. Una definizione informale di coordinate omogenee è la seguente:
Dato un insieme di oggetti, esso è rappresentato tramite coordinate omogenee se ciascuna sequenza di numeri $[x_o, x_1, \dots, x_n]$ diversa da $[0,0,\dots,0]$ identifica un oggetto. Due sequenze identificano lo stesso oggetto se e solo se una è multipla dell'altra, cioè se esiste un numero λ tale che le due sequenze sono del tipo $[x_o, x_1, \dots, x_n]$ e $[\lambda x_o, \lambda x_1, \dots, \lambda x_n]$.

Ad esempio $[1,2]$ e $[-2,-4]$ identificano lo stesso oggetto.

Abbiamo definito per mezzo della relazione di proporzionalità una relazione di equivalenza, dove un oggetto determina univocamente una classe di equivalenza di sequenze.
Ora è possibile definire uno spazio proiettivo associato ad uno spazio vettoriale V come l'insieme dei sottospazi vettoriali di dimensione uno (insieme delle rette) di V. Se lo spazio vettoriale V è munito di una base finita, ogni vettore di V è descrivibile tramite le sue coordinate $(x_1, x_2, \dots, x_n)$.
Ogni retta di V è descrivibile come lo span lineare[21] di un vettore non nullo v. Le coordinate omogenee di questa retta, rispetto alla base scelta, sono la n-upla di punti $[x_1, x_2, \dots, x_n]$ data dalle coordinate di v e definita a meno della moltiplicazione per uno scalare. Si intende che:

$$[x_1, x_2, \dots, x_n] = [\lambda x_o, \lambda x_1, \dots, \lambda x_n] \text{ per ogni } \lambda \neq 0$$

Le coordinate della retta sono ben definite e v e v' sono due vettori che generano la stessa retta se e solo se le loro coordinate differiscono per un multiplo scalare.
Ora che è stato definito cos'è una coordinata proiettiva possiamo utilizzare gli strumenti che OpenNI mette a disposizione per trasformare un punto virtuale, utilizzato per il disegno sulla mappa di profondità, in un punto reale che noi utilizzeremo per gestire il mouse.

[21] Termine inglese per indicare che se *V* è uno spazio vettoriale su un campo *K* e siano $(v_1, v_2, \dots, v_n)$ alcuni vettori di *V*, il sottospazio generato da questi vettori è il sottoinsieme di *V* formato da tutte le combinazioni lineari di questi vettori. Da cui si ha la notazione: $\text{Span}_{(v_1, v_2, \dots, v_n)} = \{a_1 v_1 + .. + a_n v_n | a_1, .., a_1 \in K\}$

Il metodo che useremo sarà ConvertRealWorldToProjective che prenderà come parametri d'ingresso: una maniglia dell'instanza, il numero di punti proiettivi da convertire, una matrice di punti di coordinate del mondo reale e un array da riempire di coordinate di punti proiettivi. Il metodo va usato con un nodo che sia in grado di generare mappe di profondità, altrimenti restituirà il messaggio di errore XN_STATUS_INVALID_OPERATION.

```
//Creo un punto 3D che tiene le coordinate della mia mano
XnPoint3D ptProjective(cxt->ptPosition);
//Converto i miei punti
m_DepthGenerator.ConvertRealWorldToProjective(1,
      &ptProjective, &ptProjective);
```

Fatta la conversione sarà possibile memorizzare ed usare come riferimento il primo punto della mano trovato nel mondo reale.

```
//creo l'origine del mio nuovo sistema di riferimento
m_Bridge->SetStartPoint(ptProjective.X, ptProjective.Y,
      ptProjective.Z);
```

Ora che si dispone di un riferimento nello spazio e di un punto che si muove con la mano dell'utente si può realizzare un muose a 3 dimensioni.
Il sistema di navigazione dovrà però soddisfare determinate richieste e garantire di non essere soggetto a disturbi. I requisiti da soddisfare sono:

- Portando la mano in avanti l'utente dovrà essere in grado di avanzare.
- Portando la mano indietro l'utente dovrà essere in grado di indietreggiare.
- Spostando la mano a destra o a sinistra l'utente dovrà essere in grado ruotare la telecamera.
- Spostando la mano in alto o in basso l'utente dovrà essere in grado di osservare rispettivamente in alto o in basso.
- L'utente dovrà essere in grado di combinare le quattro richieste precedenti al fine di poter muoversi e ruotare o muoversi ed osservare il mondo che lo circonda.
- L'utente dovrà decidere la velocità della sua camminata, avvicinando o allontanando la mano dal sensore.
- Nel momento in cui avrà un contatto visivo con un oggetto dovrà essere in grado di poter interagire con esso per mezzo del gesto Push.

- Quando vorrà uscire dall'interazione con un oggetto dovrà essere in grado di farlo con il movimento Backward.
- L'utente che si trova in contatto visivo con più oggetti dovrà essere in grado di scegliere muovendo semplicemente la mano a destra o a sinistra.
- Quando vorrà utilizzare un oggetto dovrà essere in grado di farlo con il gesto Push.

Durante la fase di test verrà valutato se questi requisiti garantiscono realmente che il sistema sia utilizzabile, se risulterà poco pratico allora verranno cambiati i metodi di interazione per l'utente. Affinchè la navigazione sia corretta bisognerà creare un margine d'immunità al rumore sui dati ricevuti; per evitare che a piccoli movimenti della mano involontari corrisponda un mavomento in avanti o indietro oppure una rotazione della telecamera. Creare dei margini per il rumore aiuterà anche la gestione dei movimenti combinati.
Prima di studiare i margini d'immunità bisogna osservare come viene identificato un movimento lungo un generico asse. Possiamo prendere in considerazione la camminata in avanti o indietro che è associata rispettivamente al movimento della mano sull'asse Z. Quello che viene fatto è calcolare la distanza Δz tra il punto attuale ed il punto iniziale, se il valore di Δz è un valore positivo allora vuol dire che è stato fatto un movimento in avanti con la mano, se Δz sarà negativo allora la mano è stata portata indietro. Possiamo considerare questo principio su un piano (come in figura 24) ed in seguito allo stesso modo nello spazio.

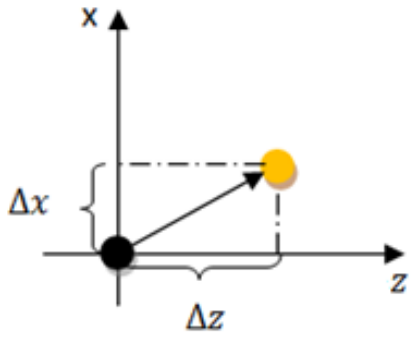

Figura 24: *Movimenti su un piano Cartesiano*

Considerando i millimetri come unità di misura possiamo quindi definire il seguente sistema. Siano P e P' due punti nello spazio, con P ≡ O (con O origine degli assi) e P' un punto in movimento, definisco:

$$\begin{cases} ratation = P.X - P'.X \\ watch = P.Y - P'.Y \\ walk = P.Z - P'.Z \end{cases}$$

Ora che abbiamo i movimenti dobbiamo creare allo stesso modo un sistema di equazioni per il magine di immunità al rumore.
Considereremo come sempre un caso più semplice lavorando con due soli assi, quindi ipotiziamo per il momento nulli gli spostamenti sull'asse Y. Muovendoci su un piano vogliamo creare due circonferenze C e C′ con centro nell'origine degli assi e che abbiano rispettivamente raggi r ed r′ con r < r′. Come rappresentato nella figura 25.

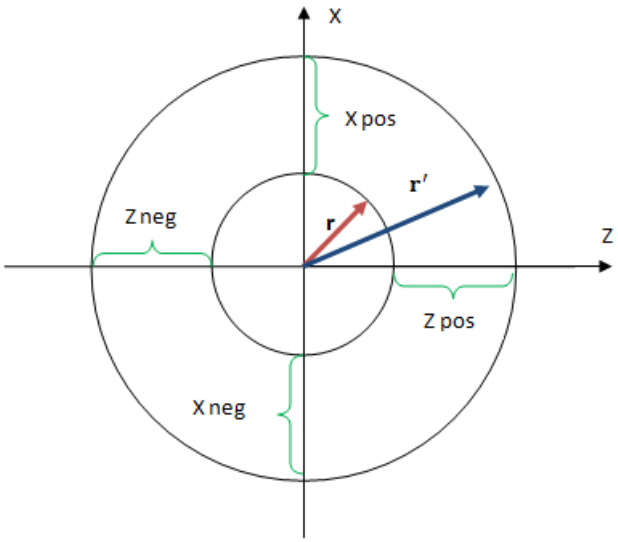

Figura 25: *Circonferenze per i margini di immunità al rumore*

In questo modo possiamo decidere di scartare tutti i movimenti della mano che sono in posizioni particolari rispetto alle due circonferenze. Scarteremo poi tutti i punti all'interno della circonferenza C di raggio r=100 mm, in quanto possono essere associati a movimenti involontari della mano, come quelli causati dalla stanchezza della posa. Inoltre va considerato un altro fattore; nel sistema il mouse non è uno strumento utilizzato dall'utente, ma è l'utente stesso. Se l'utente si allontana troppo dal sensore, ad esempio se deve rispondere ad un cellulare o aprire la porta possono verificarsi principalmente due situazioni.

- L'utente resta nell'area di monitoraggio del sensore ma la sua posizione essendo diversa da quella dell'origine degli assi viene associata ad un movimento e quindi l'utente effettuerà spostamenti non desiderati.

- L'utente esce dall'area di monitoraggio del sensore, il che comporta la perdita del punto mano, la chiusura della sessione e la richiesta di una nuova calibrazione.

Se si verifica la prima ipotesi, il sistema diventa inutilizzabile, inoltre affinché si verifichi la seconda ipotesi, la prima si verificherà comunque anche solo per pochi istanti. La possibilità di mettere il programma in pausa aiuta, ma non rende il problema trascurabile. Essendo l'utente libero di muoversi potrebbe facilmente finire in aree che non permettono una corretta navigazione, quindi creare una seconda circonferenza per restringere l'area di azione diventa un fattore di vitale importanza per rendere il sistema utilizzabile. Possiamo quindi scartare tutti i punti esterni alla circonferenza C' di raggio r'=300 mm.
Per lavorare nello spazio useremo un principio molto simile a quello appena esposto, andremo a creare due cubi uno dentro l'altro, come in figura 26.
Il margine di immunità al rumore, rispetterà le seguenti condizioni per gli assi X,Y,Z:

$$\begin{cases} 0 & se\ -100 < rotation < 100 \\ 0 & se\ -300 < rotation < 300 \\ & rotation\ \neq 0 \qquad altrove \end{cases} \qquad \begin{cases} 0 & se\ -100 < watch < 100 \\ 0 & se\ -300 < watch < 300 \\ & watch \neq 0 \qquad altrove \end{cases}$$

$$\begin{cases} 0 & se\ -100 < walk < 100 \\ 0 & se\ -300 < walk < 300 \\ & walk\ \neq 0 \qquad altrove \end{cases}$$

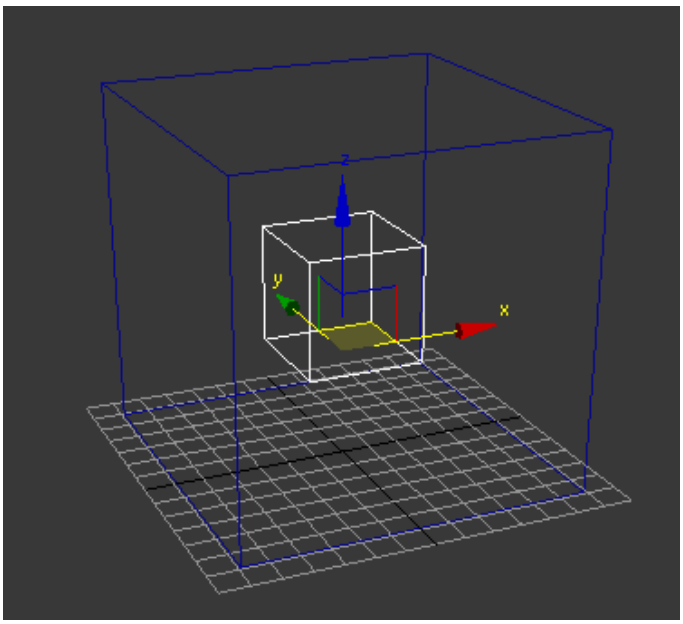

Figura 26: *Una rappresentazione a tre dimensioni della possibile area d'azione*

Anche se abbiamo creato un area di gioco adeguata, restano comunque da risolvere altri problemi relativi all'usabilità del sistema. Va ancora considerato cosa può succedere se l'utente esce dall'area che il sensore è in grado di monitorare. Il punto mano verrà perso e la sessione non verrà immediatamente chiusa, seguirà uno stato in cui la sessione è in attesa di un gesto di refocus che durerà diversi secondi. Se il gesto non si verifica la sessione verrà chiusa e l'utente sarà costretto a ripetere la fase di calibrazione. Va quindi valutato come gestire il punto che era stato scelto precedentemente come origine del sistema di riferimento. Prima di affrontare questo nuovo problema, possiamo tradurre in codice C++ i sistemi ottenuti dalle nostre considerazioni:

```
//Come vogliamo ruotare a destra o sinistra?
void Bridge::SetRotation(const XnPoint3D& stPoint){

      //calcolo rotation positivo o negativo
      rotation = (startPoint[0] - stPoint.X);
      //Setto l'area di gioco dell'utente
      if (rotation > -100 & rotation <100)
            rotation = 0;
      if (rotation < -300 || rotation >300)
            rotation = 0;

      //cambio segno per avere il movimento desiderato
      nel mondo 3D
      rotation = rotation*-1;
}

// Vogliamo avanzare oppure indietreggiare?
void Bridge::SetWalk(const XnPoint3D& stPoint){

      //calvolo il valore di walk positivo o negativo
      walk = (startPoint[2] - stPoint.Z);
      //Setto l'area di gioco dell'utente
      if(walk < 100 & walk > -100) walk = 0;
      if(walk > 300 || walk < -300) walk =0;
}

//Come vogliamo ruotare in su o in giù?
void Bridge::SetWatch(const XnPoint3D& stPoint){

      //calcolo il valore di watch
      watch = (startPoint[1] - stPoint.Y);
      //Setto l'area di gioco dell'utente
      if (watch > -100 & watch <100) watch = 0;
      if (watch < -300 || >300) watch = 0;

      //cambio segno per avere il movimento desiderato
      nel mondo 3D
      watch = watch *-1;
}
```

Come avevamo già introdotto la gestione dello startPoint ha un ruolo importante ed è necessario spendere qualche minuto per decidere come gestirlo in relazione alle calibrazioni sucessive. Se l'utente uscirà al di fuori della sfera di raggio di 30 cm i suoi movimenti vengono considerati nulli, ma può sempre riportare la mano all'interno della prima sfera e continuare ad interaggire con l'applicativo. Qualora la mano uscisse dal raggio d'azione del sensore o finisse in un punto cieco, (ad esempio l'utente mette la propria mano dietro la schiena) il sensore smetterà di tracciarla ed eliminerà il punto della mano. Questo potrà essere riassegnato in almeno tre possibili modi.

- Per mezzo del gesto di refocus "RaiseHand"
  sessionManager.Initialize(&context, "Wave,Click", "RaiseHand");

- Per mezzo del gesto di refocs "Wave" (o Click)
  sessionManager.Initialize(&context,"Wave,Click", "Wave");

- Se la sessione è terminata, con il gesto di focus Wave o Click

Ora bisogna decidere se con la riassegnazione del punto bisogna modificare anche l'origine degli assi. La risposta affermativa, anche se è la più ovvia e concettualmente la più giusta, non è sempre la soluzione più adeguata.
Il problema è proprio nei diversi gesti che si usano. Il Wave si basa sul riconoscimento di un movimento, quindi fornisce all'utente la possibilità ed il tempo di portare la mano in una posizione per lui comoda. RaiseHand principalmente si basa solo sul riconoscimento della figura della mano, qualsiasi sia la sua posizione.
Sebbene RaiseHand è uno strumento estremamente veloce e preciso per l'individuazione della mano, non permette all'utente di creare un sistema di riferimento comodo per la sua navigazione. Ad esempio supponiamo che l'utente metta la mano dietro la schiena per mandare l'applicazione in pausa, il sensore smetterà di tracciare il punto ed invierà al server l'informazione di bloccare il sistema di navigazione. Ipotiziamo poi che dopo un istante l'utente voglia riprendere il controllo dell'interazione. Per prima cosa dovrà portare la mano davanti al sensore ed utilizzando RaiseHand, questa verrà immediatamente individuata anche se è in una posizione troppo bassa, troppo alta o troppo laterale per garantire all'utente di avere un buon sistema di navigazione.

Una soluzione immediata al problema sarebbe quella di non riassegnare uno startPoint se stiamo ricalibrando la mano per la n-esima volta, ma mantenere in memoria il vecchio punto iniziale come riferimento.

```
if(ricalibrare == true){

	//Se è la prima calibrazione
	if(iCalib==0){
	m_Bridge->SetStartPoint(ptProjective.X,ptProjective.Y,
		ptProjective.Z);
	}
	else
	{
	//Se stiamo ricalibrando
	m_Bridge->SetStartPoint(m_Bridge->GetStartX(),
		m_Bridge->GetStartY(), m_Bridge->GetStartZ());
	}

	ricalibrare = false;
	iCalib++;
}
```

In questo modo possiamo utilizzare i dati forniti dalla prima calibrazione certamente più affidabile perché basata su gesti più precisi. Si intuisce immediatamente che la soluzione presentata non è quella ottima, in quanto si basa sull' ipotesi che la prima calibrazione dia sempre un sistema di riferimento comodo. Quest'ipotesi non può essere sempre verificata, l'utente potrebbe rendersi conto troppo tardi di essersi posizionato male. Spostare il sensore oppure nascondere la mano e sperare in una seconda ricalibrazione non lo aiuteranno ad interagire meglio con il programma.
Se fosse stata fatta una scelta del genere per gestire lo startpoint il sistema avrebbe presentato un errore progettuale e sarebbe diventato ancora una volta scomodo ed inutilizzabile. Si deve quindi pagare con un tempo di calibrazione leggermente più lungo la possibilità di dare all'utente i mezzi per ricalibrare l'origine del suo sistema di navigazione, se quello che ha ottenuto è troppo scomodo da usare.
Una soluzione può essere quella di usare il gesto Wave o Click come gesto di refocus.

```
//Chiamata per quando abbiamo perso il tracking delle mani
void XN_CALLBACK_TYPE NoHands(void* UserCxt)
{
	printf("Quick refocus\n");
	ricalibrare = true;
	//aggiorno lo stato
	g_SessionState = QUICK_REFOCUS;
}
```

```
//Nel main
XnStatus rc = sessionManager.Initialize(&context, "Wave,Click",
"Wave,Click");

//In pointDrawer
if(ricalibrare == true){
        //Se bisogna ricalibrare
        m_Bridge->SetStartPoint(ptProjective.X,ptProjective.Y,
              ptProjective.Z);
        ricalibrare = false;
        }
```

Dato che i due gesti Wave e Click si basano sul movimento della mano, e non sul riconoscimento della sua figura, entrambi avranno un margine di errore di cui abbiamo tenuto conto nella creazione del primo cubo per il margine di immunità ai disturbi.

Per ultima cosa oltre al riconoscimento di una mano è stata fornita anche la possibilità di monitorarne due. Durante la fase di tracking verranno memorizzati sul Bridge solo i movimenti sull'asse Z della seconda mano, questo ulteriore strumento è stato inserito per aiutare l'utente a svolgere operazioni più complesse all'interno dell'ambiente grafico. In futuro si potrà pensare di sviluppare un nuovo sistema di navigazione oppure creare gesti ad hoc basati sull'interazione delle due mani.

3.2.3 Interagire nell'ambiente grafico

Prima di spiegare il tipo d'interazione che possiamo avere con l'ambiente grafico utilizzato, verrà spiegato brevemente il procedimento che ha permesso di trasferire lo stato dell'oggetto Bridge su un oggetto KinectData nel programma Java. Il motore grafico lavorerà lato Server, il primo compito che deve svolgere è ricevere il continuo flusso di dati che il Client invierà. Le informazioni relative allo stato dell'utente sono scritte all'interno di una stringa che l'oggetto Bridge ha opportunamente preparato.

Ad esempio la stringa potrebbe contenere i seguenti valori:

"143.222 -93.100 12.444 0.0000 false false false 2.000".

Questi identificano rispettivamente la possibile posizione su asse Z, X e Y relative alla prima mano, la pozione sull'asse Z della seconda mano, lo stato del Push detector, del gesto Up e del gesto Backward ed infine la posizione dello Slider. La Classe KinectData si occuperà pertanto di recuperare la stringa e scomporla in tutte le sue parti.

Utilizzando i valori ricevuti, l'oggetto KinectData cambierà costantemente il suo stato in modo da essere al corrente ad ogni istante di quale movimento viene fatto dall'utente. Questi dati verranno recuperati in seguito dalla classe KinectDataInterpreter che si occuperà di interpretare i valori float e boolean contenuti nella stringa per interagire con l'ambiente virtuale. La classe KinectDataInterpreter andrà utilizzata opportunamente al fine di soddisfare tutti i requisiti esposti nelle prime pagine di questo capitolo.
Verranno sorvolate le informazioni riguardanti la creazione del Socket per l'architettura Client-Server e ci concentreremo fondamentalmente sulle considerazioni riguardanti l'utilizzabilità del sistema. Principalmente si desiderava permettere all'utente di navigare ed interagire con una sola mano, ma questo in fase di sviluppo ha portato il sorgere di continui conflitti. Come abbiamo spiegato precedentemente movimenti come Slider e Wave non possono esser utilizzati insieme per navigare in un menù ed uscire da esso in quanto sono molto simili e potrebbero verificarsi contemporaneamente causando conflitti e l'uscita involontaria dal menù di scelta.
Un problema simile si ha con la selezione di oggetti. Originariamente si era scelto di cominciare l'interazione su un oggetto con il gesto Up (un movimento rettilineo verso l'alto che terminava in un punto) ma questo causava anche il movimento della telecamera che superata la soglia di 10 cm cominciava a far ruotare il punto di vista dell'utente non permettendogli di interagire con un oggetto. Questo problema è relativamente raro, durante la fase di test si è verificato a un soggetto su cinque. Un altro fattore da prendere in considerazione era la possibilità di permettere agli utenti di raccogliere oggetti posti a terra, cioè in un punto che può essere visibile solo facendo ruotare la telecamera. Se l'oggetto ad esempio è sul pavimento l'utente deve portare la mano in basso per far ruotare la telecamera in quella direzione.
Si può facilmente intuire che il gesto Up non è più adatto ad interagire con l'oggetto posto sul pavimento, in quanto con un piccolo movimento della mano la telecamera verrà riportata in alto e l'interazione sarà persa. Il gesto Push non aiuterebbe la situazione in quanto il movimento in avanti porterebbe l'utente ad avanzare come succederebbe con il gesto Wave che lo farebbe spostare lateralmente. Se si vuole offrire all'utente la possibilità avere una visione totale dell'ambiente e vogliamo anche fornendogli i mezzi per interagire con oggetti posti in posizioni "scomode" una mano non riesce più a soddisfare tutte le richieste e non consente di fornire un prodotto utilizzabile.

Per tutti questi motivi si è deciso di inserire nel sistema di navigazione la seconda mano. Questa non è sempre necessaria, non influisce sulla navigazione dell'utente nell'ambiente virtuale, potrà essere calibrata solo prima di interagire con un oggetto e verrà usata solo per tale scopo. La figura 27 mostra come la mano sinistra dell'utente scende per abbassare la telecamera, mentre la mano destra attende di poter interagire con l'oggetto.

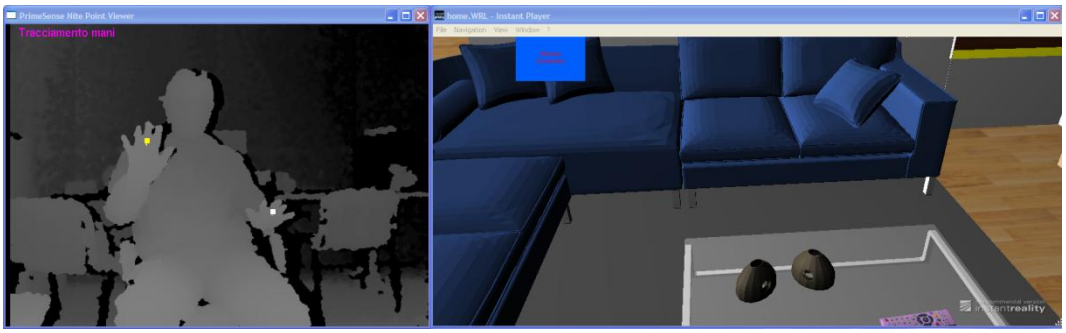

Figura 27: *Interazione con oggetti non facilmente raggiungibili*

L'uso della seconda mano si basa sul riconoscimento di un gesto simile al Push, ma questo è stato creato ad hoc e non si basa sull'uso del PushDetector messo a disposizione dalla PrimeSense. Allo stesso modo possono esser realizzati altri gesti per applicazioni future.
La classe KinectDataInterpreter è stata quindi fornita dei seguenti metodi:

- Navigation: controlla ad ogni istante il parametro Walking, Rotation e Looks della classe KinectData. Questi rappresentano gli spostamenti sull'asse X,Y,Z della mano. Il metodo navigation utilizzerà Walking per incrementare il valore deltaD, permettendo all'utente di avanzare. Rotation verrà usato per far ruotare la telecamera sull'asse Y e Looks per far ruotare la telecamera sull'asse X.

```
DeltaY = DeltaD = DeltaAngle = 0.0f;
DeltaX_axis = DeltaY_axis = DeltaZ_axis = 0.0f;

// Camminata in avanti o indietro
if (kd.getWalking() != 0) {
      DeltaD = kd.getWalking() / 300;
}

// Rotazione intorno all'asse Y
if(kd.getRotation() != 0)
{
      DeltaY_axis = 1.0f;
      DeltaAngle = (-0.06f*kd.getRotation())/300;
}
```

```
// Rotazione intorno all'asse X
if (kd.getLooks() != 0)
{
      DeltaX_axis = 1.0f;
      DeltaAngle = (-0.06f * kd.getLooks()) / 300;
}
```

Essendo kd.getWalking() un valore che l'utente può aumentare con il movimento della mano all'interno dei margini di navigazione il valore di DeltaD sarà compreso tra $\pm 0{,}33$ e ± 1. Fornirà la velocità di navigazione. In figura 28 l'utente avanza portando la mano in avanti.

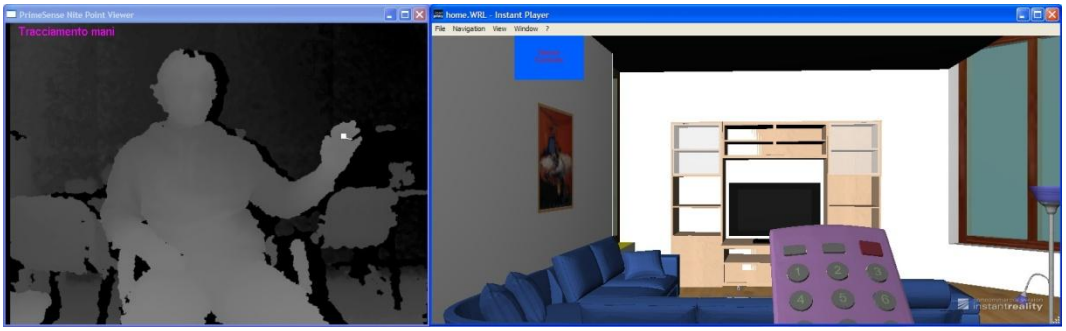

Figura 28: *Istante di navigazione all'interno dell'ambiente virtuale*

- changeState: serve per passare dallo stato di navigazione allo stato di interazione con gli oggetti. Inizialmente si utilizzava il gesto Up per iniziare o terminare un'interazione. Per diversi motivi già mensionati il gesto è stato sostituito introducento la seconda mano. Il metodo controlla il valore dell'attributo S_push dell'oggetto Kinectdata. S_push sarà true solo se l'utente aveva la mano nell'origine del sistema di coordinate e con un gesto rapido la portata in avanti sull'asse Z e successivamente è tornato in zero. Se questo movimento non si verifica o è troppo lento, il valore di S_push ricevuto dal Bridge sarà false. Se al contrario si verifica, allora l'utente ha deciso di interagire o di uscire dall'interazione (se questa era già cominciata). L'uso della seconda mano non influisce sui movimenti della prima, inoltre potrà e dovrà esser monitorata solo se necessario.

- Interaction: questo metodo si basa sull'uso dello Slider ed il risultato della sua esecuzione è presentato in figura 29. Controlla il valore di Slider che l'oggetto KinectData ha recuperato dalla stringa. Se il valore è 0 permetterà all'utente di spostarsi sull'oggetto a sinistra, se il valore è 2 l'utente si sposterà sull'oggetto a destra. Inoltra controlla se il Kinect data

è a conoscenza di un eventuale gesto Push dell'utente. Se si verifica cambierà lo stato di choice da "nothing" ad "action".

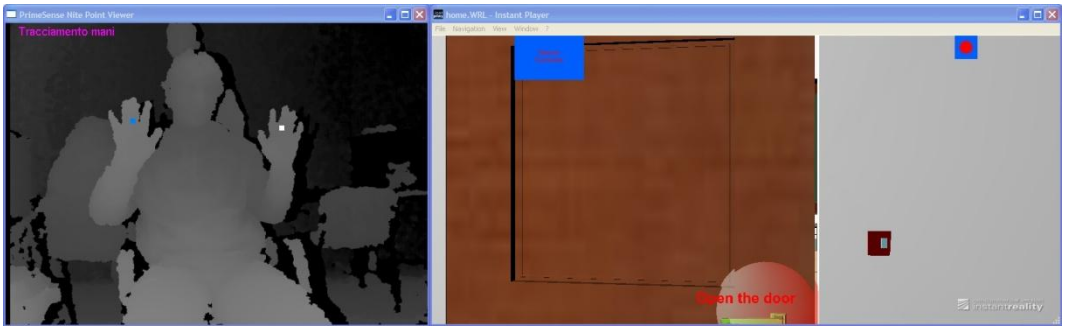

Figura 29: *Interazione con gli oggetti per mezzo di feedback visivi in scena*

- Action: Questo metodo permette semplicemente di usare l'oggetto con il quale stiamo interagendo. Per farlo controllerà semplicemente se il KinectData ha registrato un gesto Push.

Capitolo 4
Test e valutazione dell'interazione

4.1 Beta Test

Il collaudo del software (18) in informatica è il procedimento utilizzato per individuare la mancata correttezza, completezza o affidabilità delle componenti software in fase di sviluppo. Consiste nel porre il software da collaudare in esecuzione da solo oppure con altro software di servizio e valutare se sono rispettati i requisiti. Il collaudo non deve essere confuso con il debuging, con il profiling o con il benchmarking.

Più precisamente il "beta testing" è una fase di prova di un'applicazione software che viene messo a disposizione di utenti meno esperti e confidando nelle loro azioni imprevedibili si spera che questi portino alla luce eventuali errori (bug) o incompatibilità dell'applicazione software. Questa fase può anche essere svolta da personale specializzato o da semplici amatori chiamati beta tester. Alla fine del beta testing il programma è considerato completo e distribuibile. Si parlerà a questo punto di versione "build" del programma. Gli eventuali errori trovati dopo la commercializzazione potranno esser corretti per mezzo dei service pack, distribuiti dal produttore. Il lavoro di testing che è stato fatto per questa tesi, non prevedeva come obbiettivo ultimo di ottenere un prodotto pronto per la commercializzazione, ma solo assicurarsi che il sistema di interazione realizzato fosse utilizzabile per sviluppi futuri, andando a ridurre il più possibile gli eventuali malfunzionamenti, cioè comportamenti del software difformi dai

requisiti. In particolare, il testing del software è stato effettuato più volte durante lo sviluppo del progetto di tesi, il che ha permesso di correggere errori, cambiare alcune caratteristiche del progetto ed effettuare nuovamente la fase di testing per confrontare i nuovi risultati con i vecchi. Le considerazioni derivanti dai test sono state inoltre utilizzate per individuare la soluzione adottata per il programma e descritta nei capitoli precedenti. Il sistema di interazione con un mondo 3D è stato testato da 5 utenti così scelti: due studenti di Ingegneria Elettronica, due studenti di Ingegneria Informatica ed uno studente esterno alla facoltà di Ingegneria con poca pratica con la tecnologia. Il test che gli utenti hanno dovuto affrontare consisteva nell'usare le proprie mani per navigare nell'ambiente 3D proposto ed interagire con gli oggetti posizionati al suo interno. Per ogni checkpoint raggiunto veniva registrato un tempo. Alla fine della prova sono stati confrontati i tempi per vedere se il sistema era utilizzabile, inoltre è stato richiesto all'utente quali fossero i problemi o le difficoltà riscontrati. I checkpoint che gli utenti hanno dovuto raggiungere sono stati i seguenti:

1) raggiungere la porta dello studio e posizionarsi con la telecamera davanti ad essa. Per raggiungere tale posizione l'utente doveva far uso di tutti i movimenti messi a disposizione per la navigazione: avanzare, indietreggiare e ruotare.
2) Interagire con la porta aprendo l'accesso allo studio.
3) Interagire con l'interruttore della luce accendendola
4) Interagire con l'interruttore della luce spegnendola.

Di seguito verranno mostrati alcuni grafici accompagnati da brevi descrizioni per mostrare i diversi tempi ottenuti. Tutti gli utenti sono stati in grado di completare tutti i checkpoint con tempi complessivi variabili da 50 a 118 secondi. Nella figura 30 il grafico mostra tempi alti per un percorso relativamente breve. Il risultato peggiore non è stato ottenuto dall'utente esterno alla facoltà, che ha completato il test in 73 secondi. Gli utenti che hanno effettuato questa prima prova avevano ricevuto solo una spiegazione dei comandi ed avevano preso praticità con le demo NITE per capire l'interazione con il sensore Kinect. Il test fa riferimento al sistema di navigazione ed interazione basato su una sola mano. La difficoltà maggiore nel primo test era nella navigazione in quanto non riuscivano a prendere un'immediata praticità con il sensore.

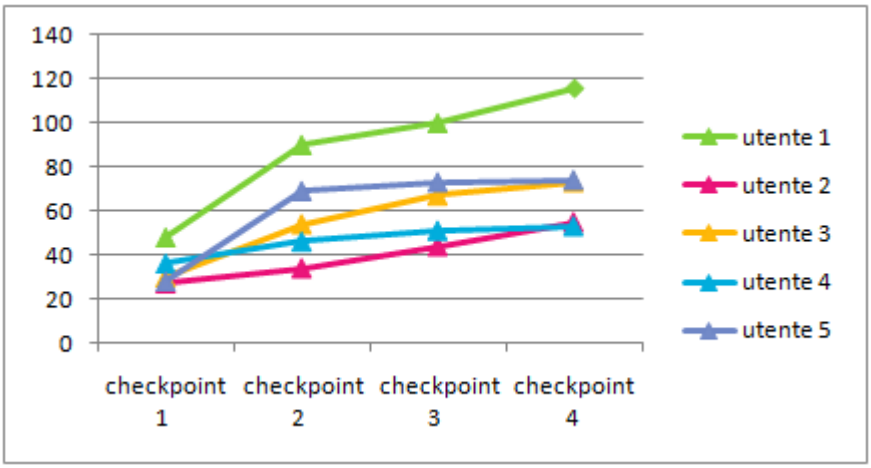

Figura 30: *Prima prova, i tempi di esecuzione sono suddivisi in quattro fasi*

La seconda prova, i cui risultati sono in figura 31 ha mostrato dei tempi nettamente migliori per la maggior parte degli utenti, che sono riusciti a prendere dimestichezza con il sistema di navigazione. Questo test però ha portato alla luce un malfunzionamento del software. Sebbene l'utente numero due, che aveva eseguito il primo test con un ottimo tempo, non abbia avuto problemi durante la fase di navigazione nell'ambiente, egli ha riscontrato un problema durante l'interazione con gli oggetti. Il gesto Up che permetteva l'inizio dell'interazione con un oggetto veniva associato anche al movimento della telecamera. Il conflitto non permetteva quindi a questo utente di interagire con gli oggetti. Gli altri utenti non hanno riscontrato il bug, tuttavia il loro ottimo risultato è dovuto a pura fortuna.

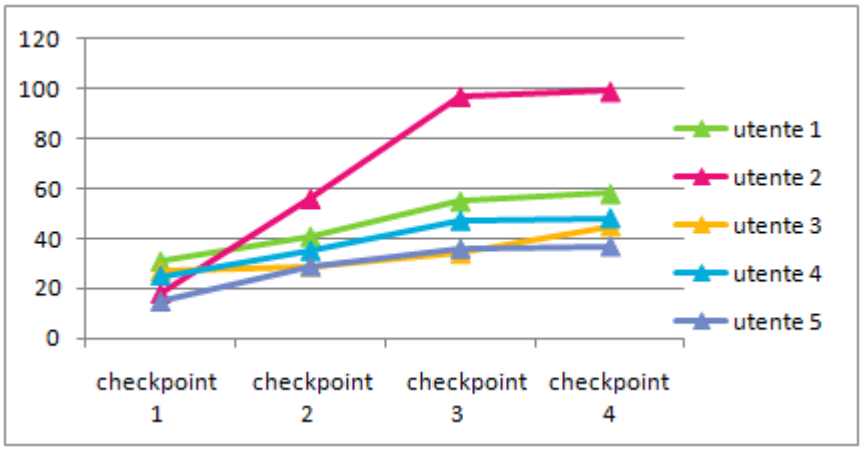

Figura 31: *Seconda prova, il tempo elevato del secondo utente è dovuto ad un malfunzionamento*

Il gesto Up di interazione, per non essere confuso con quello di movimento della telecamera, deve essere svolto all'interno dei 10 cm del margine di disturbo creato per la navigazione. Se il movimento Up esce al di fuori di quel margine il sistema diventa inutilizzabile.

4.2 Test con modifiche sul sistema d'interazione

I risultati precedenti hanno mostrato dei tempi di esecuzione che tendevano a migliorare con la familiarità che l'utente sviluppava nei confronti del sensore. Il malfunzionamento che si è verificato con il secondo utente, ha determinato un intervento di modifica al software. Con la nuova riprogettazione è stata inserita la possibilità di utilizzare la seconda mano. Questa idea, inizialmente scartata perché creava conflitti sui controllori dei gesti, è stata rielaborata. La soluzione prevede che i controllori dei gesti NITE siano attivi solo sulla prima mano, mentre sulla seconda è stato creato un controllo del gesto ad hoc. Dopo la riprogettazione gli utenti hanno effettuato nuovamente il test composto dai quattro checkpoint precedentemente descritti.

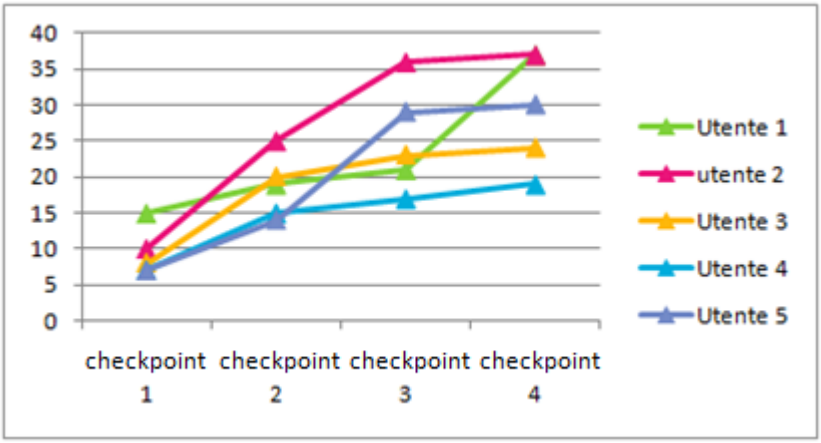

Figura 32: *Terza prova, il tempo complessivo medio è di 30,6 secondi*

Come si può vedere dai grafici riportati nelle figure 32 e 33 i tempi sono nettamente migliorati rispetto alle prime prove. Questo miglioramento si può attribuire a due fattori: al nuovo sistema di interazione con gli oggetti che è gestito dalla seconda mano e dal fatto che gli utenti hanno comunque preso maggiore familiarità con il nuovo sistema. La difficoltà principale che hanno avuto gli utenti

è stata capire quale movimento della mano era legato a particolari interazioni nel mondo virtuale.

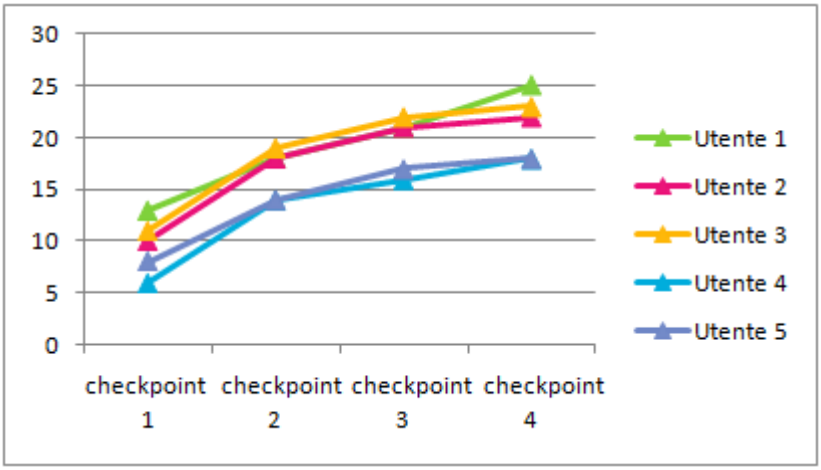

Figura 33: *Quarta prova, il tempo complessivo medio è di 21 secondi*

Nel primo test il tempo peggiore è stato di circa 120 secondi, mentre nell'ultimo test è stato di 25 secondi, questo implica che durante la fase di test e riprogettazione il tempo peggiore si è ridotto di 95 secondi. Inserire dei feedback visivi nella scena aiuterebbe l'utente a coordinare meglio i gesti nel mondo 3D e gli garantirebbe un apprendimento ed un coordinamento tra gesti e interazione più rapido.

Successivamente è stato condotto un secondo test per vedere se il sistema realizzato era utilizzabile per gestire quelle situazioni in cui veniva richiesto il movimento della visuale in prima persona verso alto o in basso e l'interazione con gli oggetti. Il test è stato svolto nel seguente modo. Gli utenti dovevano recarsi nel salotto all'interno dell'ambiente virtuale e posizionarsi davanti la televisione; a questo punto dovevano calibrare la seconda mano ed abbassare la prima mano affinché la telecamera inquadrasse il telecomando del televisore riposto sul tavolino. Interagendo con esso dovevano prenderlo e successivamente riporlo sul tavolo. Nel complesso il compito prevedeva quindi tre punti di controllo.

Il sistema è risultato utilizzabile ed ha permesso di soddisfare le richieste degli utenti in un tempo abbastanza immediato. Sul risultato di alcune prove hanno influito però la perdita del punto della mano, ad esempio quando la mano durante la navigazione si avvicina al viso di utenti con capelli lunghi. Un altro problema si è riscontrato quando l'utente utilizzava il sensore stando ad una distanza troppo elevata da esso. I suoi movimenti molto probabilmente non venivano percepiti correttamente ed i punti assegnati alle mani correvano il rischio di essere invertiti.

Non potendo modificare gli algoritmi OpenNI e NITE, un feedback visivo inserito nel mondo virtuale o la soluzione di mettere in pausa l'applicazione aiuterebbe l'utente ad orientarsi meglio in questi casi. Nelle figure 34 e 35 sono riportati alcuni risultati riguardanti il secondo test.

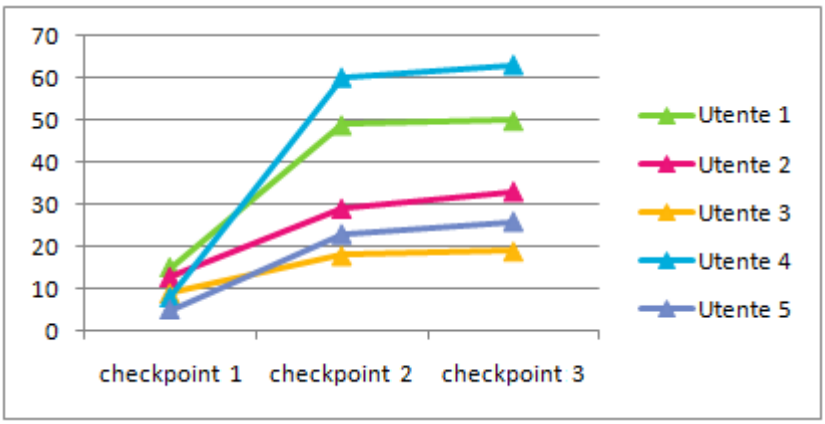

Figura 34: *Prima prova del secondo test.*

Si può notare in figura 34 che l'utente 4 ha eseguito il compito con un tempo sensibilmente più alto in quanto ha riscontrato il problema dell'inversione dei punti. In figura 35 si evidenzia come nelle successive prove riducendo la distanza dal sensore non ci sono stati problemi d'inversione dei punti.

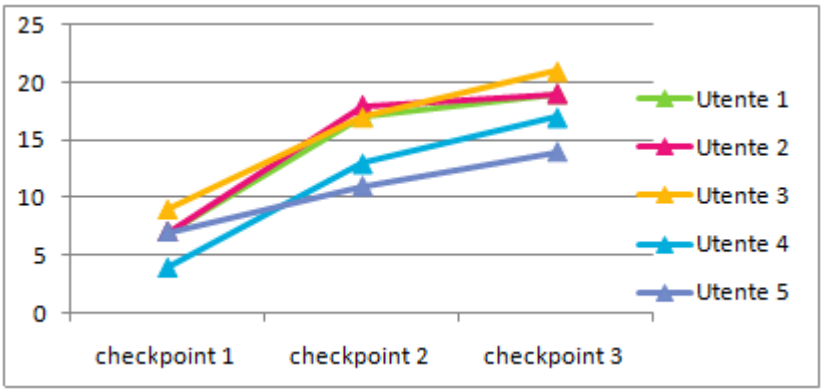

Figura 35: *Seconda prova, i tempi migliorano*

Dai test condotti è emerso che gli utenti non hanno riscontrato particolari difficoltà a familiarizzare con il sistema di navigazione ed interazione e non hanno trovato difficile il suo utilizzo. Particolare attenzione va posta all'utilizzo della mano da parte dell'utente: affinché il sistema funzioni correttamente la mano deve avere il palmo rivolto verso il sensore e deve essere aperta, in modo da permettere agli algoritmi OpenNI e Nite d'identificare le dita e riconoscere la mano come

tale. Il sistema potrebbe anche rispondere ai movimenti di una mano chiusa o con le dita serrate, ma i risultati non sarebbero ottimali in quanto l'identificazione della mano non è precisa. Per evitare inversioni dei punti si deve inoltre evitare di tenere le mani vicine se si è ad una distanza elevata dal sensore. Va sempre considerato che l'intero applicativo si basa sull'uso di ottime librerie Open source, ma pur sempre instabili e messe a disposizione su Internet per effettuare dei beta test.

Capitolo 5
Conclusioni e sviluppi futuri

5.1 Discussione del sistema realizzato

In questo lavoro di tesi è stato sviluppato un mouse a tre dimensioni, in grado di soddisfare i requisiti dell'utente per la navigazione e l'interazione in ambienti virtuali. La soluzione presentata soddisfa adeguatamente i requisiti richiesti.
Il ruolo fondamentale dell'applicativo è gestito dal Client che potrà in futuro essere interfacciato ad altri apllicativi utilizzando una struttura Client-Server come quella presentata in questa tesi.
Il sistema di navigazione ed interazione si basa sull'uso di librerie ancora instabili sulle quali attualmente ricercatori ed appassionati investono molto tempo e denaro. Il risultato di questo lavoro potrà essere migliorato inserendo il tracking del corpo in fase di calibrazione. Tale soluzione richiederà un maggiore tempo per intraprendere la navigazione ma permetterà d'assegnare i punti della mano ad un determinato scheletro consentendo a più utenti (rappresentati da avatar) di navigare nella stessa scena.
Senza l'inserimento dello scheletro potranno esserci problemi sull'uso della seconda mano individuata se non si usa il gesto di focus adeguato; la calibrazione della seconda mano da parte di un secondo utente potrebbe togliere all'utente principale la possibilità di interagire con gli oggetti. Tutte le altre mani in scena invece non influiranno in alcun modo perché la funzionalità d'interazione è stata attribuita solo alla seconda mano individuata.
La probabilità che questo evento si verifichi non è molto elevata se viene utilizzato un adeguato gesto di inizio sessione. Il secondo utente per essere

riconosciuto deve intenzionalmente effettuare il gesto Wave o click per interferire nel sistema di navigazione ed interazione. Il gesto click però deve essere molto preciso per essere riconosciuto. Il solo uso del gesto click come inizio sessione ridurrebbe la probabilità d'errore rendendola simile alla probabilità che un collega ci tolga il mouse di mano mentre stiamo scrivendo un'e-mail.
Il sistema comunque ha risposto bene ai vari test fornendo spunti per apportare modifiche in fase di sviluppo, inoltre il costante miglioramento nei tempi di esecuzione dei compiti riscontrati nel beta test ha mostrato che gli utenti non hanno eccessiva difficoltà ad utilizzare questo tipo di tecnologia e a prenderci familiarità. L'assenza di una periferica utilizzabile con le mani non rappresenta un problema: in un prossimo fututo mouse e tastiere potrebbero essere sostiuiti con periferiche di nuova generazione basate su interazione gestuale.
Le librerie ed il sensore utilizzati in questo lavoro di tesi possono essere impiegati in tutti gli ambiti in cui l'interazione gestuale rappresenta un valore aggiunto.
Nei prossimi mesi saranno disponibili versioni più aggiornate e più stabili delle librerie che metteranno a disposizione controlli dei movimenti basati su algoritmi più sofisticati. Molto probabilemte esse permetteranno anche il riconoscimento e l'assegnazione di un punto alle dita di una mano (attualmente disponibile con l'uso di ROS). Verrà fornita inoltre la possibilità di poter lavorare con i comandi vocali.

5.2 Sviluppi futuri dell'applicazione

In queasta sezione viene fornita una breve lista di idee per i possibili sviluppi futuri che riguardano miglioramenti all'applicazione presentata in questo lavoro di tesi. Gli sviluppi sono preposti in particolare a coloro che vogliono affiancare l'uso del sensore allo studio della grafica computazionale.

- Inserire il tracking dello scheletro affiancandolo al riconoscimento della mano per ottenere un sistema più stabile.

- Inserire un sistema di collisioni nell'ambiente virtuale e realizzare ambienti più ampi da visitare con maggiori oggetti con cui interagire.

- Effettuare nuovi test per valutare se i controlli basati sul movimento della mano possono essere ulteriormente migliorati, realizzando ad hoc nuovi

gesti che possono essere riconosciuti. (Si veda come esempio il gesto Push associato alla seconda mano).

- Realizzare un videogioco utilizzabile su internet con possibilità di utenti multipli.

5.2.1 Sviluppi nell'ambito della Robotica

In questa sezione vengono presentati i possibili sviluppi che hanno come base di partenza l'uso del sensore e il software sviluppato per il client, che potranno essere integrati in altri campi di studio che differiscono dalla grafica computazionale. Le classi sviluppate lato client riescono a riconoscere un sostanzioso alfabeto di gesti, ma solo alcuni di questi sono stati realmente utilizzati in quanto trovavano un riscontro pratico nell'applicazione realizzata. Tutti gli altri gesti vengono monitorati e registrati ma sono rimasti inutilizzati. Sebbene fosse possibile toglierli, è stato scelto di lasciarli per agevolare gli sviluppi futuri. Quest'alfabeto di gesti e il tracking della mano che ha permesso di realizzare il mouse in 3 dimensioni presentano diverse possibilità di utilizzo:

- manovrare a distanza veicoli come l'AR drone o altro, realizzando il sistema lato server per interfacciarsi al veicolo volante. L'AR drone ad esempio è un quadricottero dotato di due telecamere. Il drone invia i dati ad un applicativo eseguito su iPhone ed iPod touch permettendo all'utente di manovrare il drone. Il sistema può essere portato su PC ed interfacciato con il sensore. Il software lato client realizzato fornirebbe un sistema di navigazione in 3 dimensioni che attualmente non è disponibile sui sistemi touch screen che si basano sul 2D.

- manovrare a distanza sistemi meccanici in grado di trasportare carichi pesanti. Per fornire un esempio concreto, il sensore è stato impiegato per muovere i riflettori di un palcoscenico. Questo sistema offre un supporto alla sicurezza dei tecnici delle luci che devono sostare sulle impalcature solo il tempo necessario al montaggio delle diverse componenti meccaniche.

- utilizzare i dati inviati via client per gestire i movimenti di un robot umanoide come il Nao. Il movimento della mano in avanti, indietro, a

destra o a sinistra potrà essere usato come input per far eseguire dei movimenti già sviluppati come camminare, indietreggiare, voltarsi a destra o a sinistra. I gesti Push o Wave potranno esser associati a movimenti delle braccia. La soluzione non prevede un tracking del corpo ma il solo uso del sensore come telecomando.

5.2.2 Evoluzione del sistema

Per ultimi, ma non meno importanti, vengono presentati quei possibili sviluppi che prevedono come base di partenza l'uso del sensore e le librerie OpenNI e NITE, ma richiedono un nuovo sviluppo del software. Il progetto proposto in questa tesi potrà essere utilizzato come guida per il programmatore, che troverà numerosi commenti e chiarimenti nel codice sorgente. Tali sviluppi comprendono i seguenti:

- collegare il sensore ad un robot mobile come un iRobot Roomba e con l'uso di ROS utilizzarlo per permettergli d'individuare ed evitare ostacoli.

- utilizzare il tracking del corpo con i programmi Autodesk MotionBuild e 3D studio Max per realizzare effetti speciali.

- usare il tracking del corpo per muovere robot umanoidi come il Nao. Andrebbe fatto uno studio adeguato della cinematica diretta ed inversa per associare i movimenti dei giunti dello scheletro ai movimenti di giunto del robot.

- utilizzare la fase di segmentazione che precede la fase di tracking del corpo come base per sviluppare un sistema che monitorizzi e studi le traiettorie delle persone in un ambiente urbano.

- associare al punto della mano il puntatore di un mouse degli attuali sistemi operativi per navigare su desktop, aprire cartelle, usare browser ed usare un sistema che simuli una tastiera per scrivere testi.

Per maggiori spunti si possono visitare la comunità OpenNI oppure kinecthacks.

Appendice

Ruotare un Cubo in OpenGl con Kinect

Quella che segue è una semplice applicazione basata sull'interazione tra il Kinect e l'ambiente grafico OpenGL. È quella che si può definire un "Hello World".
Si è cercato di mettere assieme le conoscenze raccolte sul sensore Kinect e sulle librerie OpenNI e NITE per interagire per mezzo del sensore sulla rotazione di un Cubo. La visualizzazione grafica del cubo è stata fatta utilizzando le librerie OpenGL (figura 36) e per la sua rotazione è stata utilizzata la funzione glRotatef che veniva richiamata nella funzione Draw ogni volta che il cubo doveva essere ridisegnato nella scena. La funzione glRotatef riceve in ingresso quattro parametri, un angolo e gli assi intorno ai quali avviene la rotazione. Per semplicità è stato deciso di far ruotare il cubo solo intorno all'asse Y e l'angolo che gli viene passato per la rotazione dipende dalla posizione della mano dell'utente.

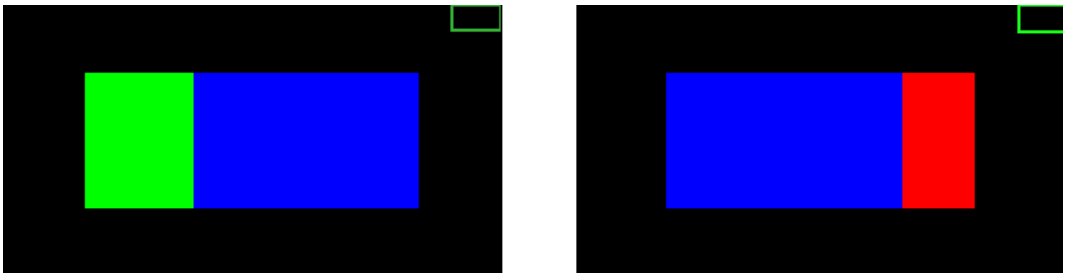

Figura 36: *Rotazione del cubo con Kinect, vista frontale.*

Sono state utilizzate le librerie NITE per riconoscere il gesto Wave ed inizializzare la sessione. Nel momento in cui la sessione è attiva compare un

rettangolo verde sullo schermo e inizieremo a monitorare tutti i punti che la mano avrà nel tempo. Monitorare i punti però non basta ad interagire con il cubo, in quanto solo quando il gesto Circle viene riconosciuto il cubo riesce a ruotare.
Nel momento in cui viene monitorato il gesto Circle i punti della mano sono riportati lungo una circonferenza di diametro unitario.
Se tracciamo una retta che passa per l'origine e per il punto che identifica la mano, possiamo trovare un angolo che si crea con l'ascissa del piano cartesiano, che aumenterà con il movimento della mano verso l'alto.
Possiamo inoltre tenere costantemente traccia del punto mano sulla circonferenza salvando le sue coordinate all'interno di un vettore.

```
//Creo un vettore XnV3DVector (XnFloat fX, XnFloat fY, XnFloat fZ)
XnV3DVector vec(0.5, 0.5, 0);
//Riporto il mio punto sulla circonferenza
vec += XnV3DVector(sin(g_fCubeAngle),-cos(g_fCubeAngle),0) * 0.3;
```

Osservando la figura 37 si può notare che la mano girando lungo la circonferenza toccherà un punto massimo ed un punto minimo, da questo ricaveremo l'angolo per la rotazione del cubo come:

$$\theta_{rot.} = 360 * \frac{X_{tracciato}}{X_{max}+X_{min}} \quad \text{con} \quad X_{max} + X_{min} = 1$$

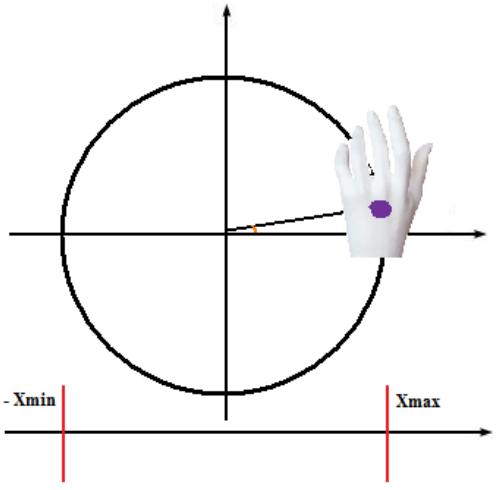

Figura 37: *Movimento della mano con moto circolare*

Con questa semplice considerazione e l'utilizzo delle librerie OpenNI e NITE è stato quindi possibile realizzare una semplice applicazione per l'interazione con un oggetto in un ambiente 3D.
Questa applicazione non ha un reale valore pratico, ma ha consentito di concretizzare gli studi sulle librerie prima di confrontarsi su un sistema più complesso. I sorgenti commentati con ulteriori spiegazioni si possono trovare con i sorgenti del progetto di tesi.

Bibliografia

(1). Interazione uomo-macchina - Alan Dix, Janet Finlay, Gregory D. Abowd, Russell Beale – McGraw-Hill (2004)

(2). Understanding motion capture for computer animation and video games - Alberto Menache - (2000)

(3). Analisi del movimento - Jacquelin Perry, M. G. Benedetti – Elsevier (2005)

(4). Natural Interaction - http://www.naturalinteraction.org/

(5). Autodesk Education Suite for Industrial Design - http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=13163791-

(6). PrimeSense - http://www.primesense.com/ -

(7). Cella di Peltier - http://it.wikipedia.org/wiki/Cella_di_Peltier -

(8). Beamforming principi generali - http://www.worldlingo.com/ma/enwiki/it/Beamforming-

(9). Learning OpenCV: computer vision with the OpenCV library - Gary Bradski, Adrian Kaehler – O'Reilly (2008)

(10). OpenKinect - http://openkinect.org/wiki/Main_Page

(11). OpenNI sito ufficiale - http://www.openni.org/

(12). Licenza LGPL- http://it.wikipedia.org/wiki/GNU_Lesser_General_Public_License

(13). Prime Sensor™ NITE 1.3 Controls

(14). VRML: http://it.wikipedia.org/wiki/VRML

(15). Guida all'uso VRML da http://xml.html.it/guide/lezione/1810/che-cos-e-il-vrml/

(16). Coordinate omogenee: http://it.wikipedia.org/wiki/Coordinate_omogenee

(17). Spazio proiettivo: http://it.wikipedia.org/wiki/Spazio_proiettivo

(18). Collaudo del software: http://it.wikipedia.org/wiki/Collaudo_del_software