Alma Mater Studiorum · Università di Bologna
Campus di Cesena

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Informatica

Kinect e OpenNI a supporto delle NUI (Natural User Interface) applications

Elaborato in Fondamenti di Informatica A

Relatore:
Chiar.mo Prof.
VIROLI MIRKO

Presentata da:
MILZONI ALESSANDRO

Sessione 2°
Anno Accademico 2013 - 2014

DEDICA*:*

A tutti coloro che mi hanno permesso
di raggiungere questo obbiettivo ...

Introduzione

Le nuove tecnologie stanno diventando sempre più naturali e intuitive. Le persone utilizzano gesti e parole per interagire con i propi PC e dispositivi, tali modi naturali di interagire con le tecnologie rendono il loro funzionamento più semplice. Le NUI (Natural User Interface) applications si concentrano di agevolare l'impiego dei futuri paradigmi informatici in maniera del tutto naturale.

In questa tesi si è voluto fare uno studio generale su questo nuovo tipo di interfaccia concentrandosi soprattutto sul framework OpenNI creato appositamente per lo sviluppo delle NUI application arrivato alla nuova versione la 2.0, cercando di capire se effettivamente questa libreria con le sue funzionalità possa aiutare e favorire gli sviluppatori.

Come caso di studio mi sono concentrato sull'applicazione illustrata nella tesi dell' Ing. Simone Costanzi, sulla parte sviluppata sfruttando la vecchia versone di OpenNI 1.5 per la rilevazione degli utenti e della loro attenzione, confrontandola con una nuova applicazione sviluppata da me sfruttando questa volta le nuove funzionalità introdotte dalla nuova versione di OpenNI 2.0 e verificare se effettivamente si presentano miglioramenti funzionali e dal punto di vista delle prestazioni.

I miglioramenti percepiti sono molti, dall'intruduzione di nuove funzionalità ad eventi che ci informano della presenza di uno o più nuovi utenti, ad una più semplice e intuitiva stesura del codice grazie all'introduzione di nuove API fino ad un miglioramento delle performance percepito da una esecuzione più fluida dell'applicazione.

Questa tesi è organizzata come segue:

Primo Capitolo : Vengono esposti i concetti principali delle NUI (Natural User Interface) dalla loro evoluzione ai campi di utilizzo.

Secondo Capitolo : Data l'importanza che ha il Kinect in questo progetto, viene presentata una panoramica su questo sensore. Vengono descritte le caratteristiche tecniche e vengono dati alcuni cenni principali sull'hardware e sul software.

Terzo Capitolo : In questo capitolo viene introdotto il framework OpenNI 2.0 utilizzato per lo sviluppo di applicazioni che sfruttano le NUI, riportando i concetti principali della guida di programmazione (pubblicata dalla casa produttrice) con spiegazione del funzionamento delle varie classi utilizzate. In più viene fatta una panoramica anche sul Middelware Nite 2.0 che si appoggia su OpenNI 2.0.

Quarto Capitolo : In questo capitolo vengono riportate le differenze riscontrate dalla versione 1.5 alla 2.0 di OpenNI con panoramica sui vantaggi e miglioramenti riscontrati nella nuova versione.

Quinto Capitolo : Nell'ultimo capitolo viene riportato un esempio pratico sull'utilizzo di questa nuova versione di OpenNI. Descrivendo le funzionalità dell'applicazione sviluppata, la sua architettura e confrontandola con un'altra applicazione sviluppata con la versione precedente del framework mettendo in evidenza le differenze tra le due versioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

NUI (Natural User Interface)

Le Natural User Inteface (NUI) consentono l'interazione con i nuovi dispositivi in modo del tutto naturale come il movimento delle mani o il ricoscimento vocale in modo da rendere più facile e intuitivo l'utilizzo del dispositivi anche da parte degli utenti più inesperti.

Molto suggestiva e interessante è la tendenza degli ultimi anni di "buttare via" mouse e tastiera e di poter interagire direttamente con il dispositivo. Scene memorabili in film come Minority Report e Avatar ci suggeriscono un futuro in cui, invece di sforzarsi a cliccare, trascinare o digitare le informazioni su uno schermo si potranno manipolare con le mani e la voce in modo del tutto naturale per l'utente.

Mentre l'idea in sè è abbastanza antiquata, le tecnologie per sostenere le NUI, e il campo di studio attorno a loro, è relativamente nuovo.

Nonostante le prime NUI siano già entrate nella nostra vita quotidiana, come display touch screen e dispositivi di riconoscimento vocale, con l'evoluzione di nuove tecnologie come il Kinect avremo un nuovo impatto sulla vita quotidiana in cui le persone potranno interagire con i vari dispositivi con le stesse modalità con cui interagiscono tra di loro.

Vi è una larga convinzione che questa nuova tecnologia trasformerà i nostri modi

Figura 1.1: NUI (Natural User Interface)

di fare. Fino ad ora, abbiamo sempre dovuto adattarci ai limiti della tecnologia e di conformare i nostri modi di lavorare ai computer ad una serie di convezioni e procedure arbitrarie. Con le NUI saranno i dispositivi informatici che per la prima volta dovranno adattarsi alle nostre esigenze e l'uomo inizierà a utilizzare la tecnologia nel modo che è più comodo e naturale per lui.

Inoltre queste nuove tecnologie porteranno nuovi benefici, come la possibilità di usare strumenti informatici avanzati che prima erano utilizzati solo da esperti con competenze altamente specializzate saranno a disposizione di chiunque. Ancora più importante con le NUI le persone che non possono avere le competenze di base di alfabetizzazione saranno in grado di sfruttare alcuni dei vantaggi della nuova tecnologia oppure la possibilità di rendere accessibili i PC alle persone con disabilità fisiche e cognitive.

1.1 Evoluzione

Le NUI sono state descritte come "la prossima fase evolutiva dopo il passaggio dall'interfaccia a riga di comando (CLI) all'interfaccia grafica utente (GUI)".

Nella CLI, gli utenti dovevano utilizzare un mezzo artificiale per immettere i dati, la tastiera, e usare una serie di input in codice con una sintassi rigorosa, ricevendo i dati in uscita sotto forma di testo scritto. In seguito, quando venne introdotto il

mouse e l'interfaccia grafica, gli utenti potevano più facilmente interagire con il sistema attraverso i movimenti del mouse, permettendo di lavorare maggiormente con contesti ad oggetti e con il contenuto attivo visualizzato sullo schermo.

Un rapido esempio per fare capire la filosofia di un'interfaccia grafica è l'utilizzo dello strumento "drag" ("trascinare"). Si dispone sul desktop di una finestra che intendiamo spostare, la cosa più intuitiva è prenderla e trascinarla dove più ci interessa. Questo è proprio quello che s'intende effettuare, cliccando con il mouse sulla finestra, spostandola letteralmente.

Come si può notare la GUI è di più facile utilizzo, anche per i meno esperti di computer, rispetto alla CLI. Inoltre non richiedono competenze avanzate, e ciò ha permesso alla massa di avvicinarsi all'utilizzo del computer, facendone diventare un mezzo di largo e facile uso, a differenza dei primi anni '80, il cui utilizzo era riservato ad una piccola nicchia ed èlite d'utenti. Le NUI invece consente agli utenti di manipolare i contenuti in modo più diretto usando movimenti più naturali, azioni e gesti. Poichè la NUI è così veloce da imparare, l'aggettivo "intuitivo" viene usato da molti per descrivere come gli utenti interagiscono con esso.

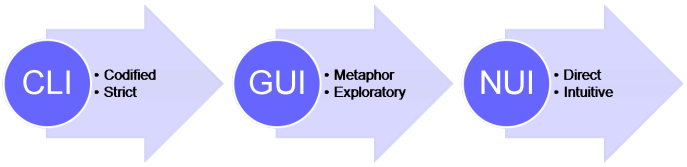

Figura 1.2: Evoluzione delle interfacce utente

1.1.1 Batch computing (1940s)

[1] La storia delle Interfacce Utente ha inizio nel lontano 1940 in modo piuttosto rudimentale. Nelle elaborazioni Batch non vi era quasi nessuna interazione tra l'utente e il computer a parte il comando di input iniziale, che era nella maggior parte dei casi fatto meccanicamente. Batch computing gradualmente si trasformarono e arrivarono dispositivi con Interfaccia a riga di comando che diventarono popolari nel 1980 - '90.

Figura 1.3: Batch computing (1940s)

1.1.2 Command line Interface Shell (1970s)

CLI è un tipo di interfaccia in cui l'utente digita i comandi sulla tastiera e riceve indietro una risposta dal sistema attraverso un display, l'interazione è limitata ad un range di comandi predefiniti oltre ai quali il sistema risponde con un errore. Nonostante questo le CLI continuano a coesistere nei sistemi operativi moderni. Rimangono ancora una parte molto integrante nei sistemi basati su Unix - Linux per la risoluzione di problemi avanzati.

1.1.3 Graphical User Interface GUI (1980s)

La necessità di rendere l'interazione con il computer ancora più attraente portò allo sviluppo delle Interfacce Grafiche - GUI, alla fine del ventesimo secolo. Le GUI permettono tutt'ora all'utente di scegliere i comandi, avviare pogrammi, visualizzare elenchi di

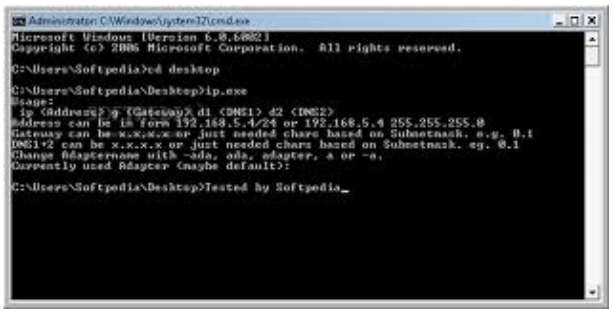

Figura 1.4: Command line Interface Shell (1970s)

file e altre opzioni puntando e cliccando con un mouse a semplici icone visualizzate sullo schermo.

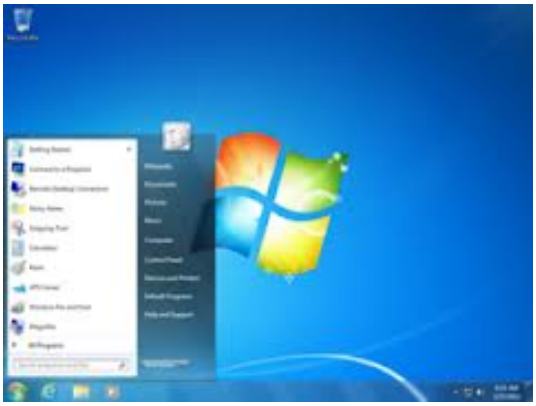

Figura 1.5: Graphical User Interface GUI (1980s)

1.2 Primi esempi di interfacce NUI

1.2.1 Touch screen interfaces

Le interfacce touch screenconsentonono agli utenti di interagire con i controlli e le applicazioni in modo più intuitivo e diretto di una interfaccia basata su cursore. Ad esempio invece di spostare un cursore per selezionare un file e fare clic per aprirlo basterà

toccare una rappresentazione grafica del file per aprirlo. Smartphone e tablet in genere consentono questa Interfaccia.

1.2.2 Gesture recognition systems

Questi sistemi monitorano i movimenti dell'utente e traducono i suoi movimenti in istruzioni da eseguire sul dispositivo. Nintendo Wii e sistemi di gioco di movimento come PlayStation Move lavorano attraverso accelerometri e giroscopi situati su controller che misurano inclinazione, rotazione e accelerazione di essi. Ancora più intutive sono le NUI che riconoscono gesti specifici e li traducono in azioni.

Kinect di Microsoft, per esempio, è un sensore di movimento per la console di gioco Xbox 360 che permette agli utenti di interagire con i movimenti del corpo, gesti e comandi vocali. Molto interessante è la possibilità di usare questo dispositivo per poter interagire con i computer.

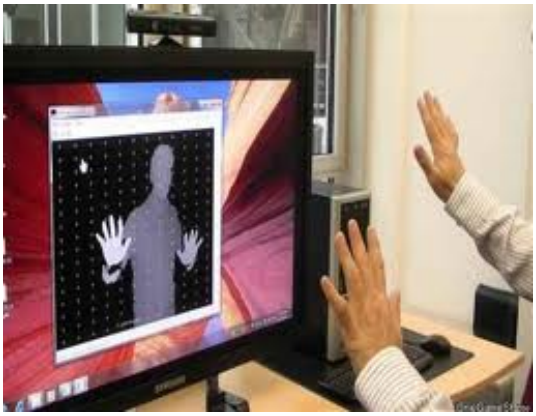

Figura 1.6: Gesture recognition systems

1.2.3 Speech recognition

Il riconoscimento vocale consente agli utenti di interagire con il sistema attraverso comandi vocali. Il sistema identifica le parole e le frasi pronunciate e li converte in un formato leggibile dalla macchina per l'interazione. Applicazioni di riconoscimento vocale sono utilizzate soprattutto nell'ambito telefonico.

1.2.4 Gaze tracking

Interfacce Gaze tracking consentono agli utenti di guidare un sistema attraverso il movimento degli occhi.

A marzo 2011, Lenovo ha annunciato di aver prodotto il primo computer portatile eye-controllato. Il sistema di Lenovo unisce una sorgente di luce infrarossa con una macchina fotografica per cattutrare riflessi degli occhi. Un software calcola l'area dello schermo che si sta guardando e utilizza tali informazioni come input.

1.2.5 Brain-machine interfaces

Questo tipo di interfacce leggono segnali neurali e utilizzano programmi per tradurre quei segnali in azioni. BCI rendono possibile per chi è paralizzatoa poter far funzionare un computer, una sedia a rotelle motorizzata o altro.

1.3 Modello NUI a 3 livelli

Come è possibile vedere nell'immagine sottostante ogni Natural User Interface può essere suddivisa in 3 livelli o parti.

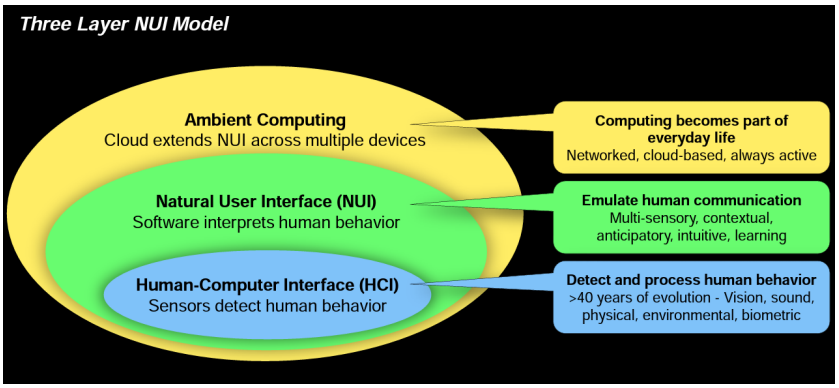

Figura 1.7: A - Modello NUI a 3 livelli (Presa dal sito di developer summit)

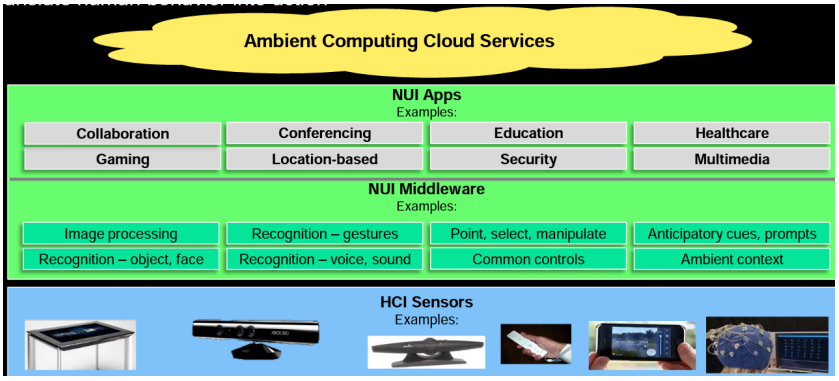

Figura 1.8: B - Modello NUI a 3 livelli (Presa dal sito di developer summit)

1.3.1 Livello 1

Questo livello è composto dai Devices e Software utilizzati per rilevare ed elaborare i comportamenti dell'utente: sensori, videocamere, microfoni...

1.3.2 Livello 2

Questo livello è composto da Middleware e Framework che traducono il comportamento dell'utente in azioni da svolgere sul dispositivo.

1.3.3 Livello 3

Questo livello ci permette di interagire con i nostri dispositivi ovunque noi siamo in modo del tutto naturale.

Capitolo 2

KINECT

Microsoft Kinect (inizialmente conosciuto con il nome Project Natal)[2], è un accessorio per Xbox 360 sensibile al movimento del corpo umano. Esso rende il giocatore stesso controller della console senza l'uso di alcuno strumento, a differenza dei concorrenti come Nintendo Wii o Sony Playstation.

Figura 2.1: Kinect

Sebbene in origine il dispositivo Kinect fu pensato esclusivamente per Xbox 360, Microsoft prevede di rendere nel prossimo futuro disponibile l'uso della periferica ai PC dotati del nuovo sistema operativo Windows 8. Microsoft ha pero dichiarato di rilasciare gratuitamente, durante l'estate 2011, i driver ufficialii per poter utilizzare Kinect nel proprio Personal Computer, dimostrando così di voler portare quanto prima la tecnologia

di Kinect anche sui sistemi operativi Windows attualmente disponibili, favorendo lo sviluppo di varie applicazioni tra il mondo degli sviluppatori di software.

2.1 Storia

Kinect è stato annunciato al pubblico il 1 giugno 2009 durante la conferenza stampa della Microsoft all'E3 2009 (Electronic Entertainment Expo) con il nome Project Natal, poi rinominato Kinect alla presentazione ufficiale all'E3 2010.

Il 13 giugno 2010 Microsoft ha rivelato per la prima volta il vero nome del dispositivo, ovvero Kinect. Quest'ultimo è in vendita dal 4 novembre 2010 in America e dal 10 novembre in Europa, ed è possibile usarlo su un qualsiasi modello di XBOX 360.

L'hardware di Kinect si basa su tecnologie della 3DV, una compagnia israeliana specializzata in tecnologie di riconoscimento dei movimenti tramite videocamere digitali che Microsoft ha prima finanziato e poi acquisito nel 2009, e sul lavoro della israeliana PrimeSense, che ha poi dato in licenza la tecnologia a Microsoft.

Il software di Kinect è stato, invece, sviluppato internamente dai Microsoft Game Studios e, più precisamente, dai programmatori della Rare, la quale ha dovuto cancellare altri progetti migliori per dedicarsi interamente alla periferica.

L'uscita di Kinect ha provocato un grande sommovimento nella comunità di sviluppo libero di software per PC e Mac. Una moltitudine di programmatori è al lavoro sul reverse engineering sulla periferica, allo scopo di trovare nuove modalità di utilizzo di un dispositivo che si configura come il primo di una serie di sistemi che potrebbe davvero portarci ad un futuro alla Minority Report.

2.2 Campi di utilizzo

Kinect è uno strumento nato come componente aggiuntivo per la consolle XBOX 360, quindi il contesto principale rimane quello dei videogiochi. Alcuni esempi in commercio che utilizzano Kinect come unico controller sono Kinect Adventures, Kinect Animals ed il gioco di ballo Dance Central.

Il costo relativamente basso insieme alle funizionalità di body-tracking che il dispositivo offre, ha fatto smuovere ed incuriosire la massa di sviluppatori software. Dopo qualche mese infatti il Web si è popolato di una moltitudine di applicazioni non strettamente legate al contesto dei videogames. Tra queste si possono citare programmi di visualizzazione di immagini, di riconoscimento del volto, plugin per software già esistenti e addirittura prototipi di riproduzione di una persona attraverso l'utilizzo di due Kinect.

Grazie a questo sensore è possibile eliminare mouse, tastiere e telecomandi: persone disabili potrebbero utilizzare questi dispositivi per abbattere numerose barriere che impediscono loro l'utilizzo della tecnologia.

Il sensore può essere utilizzato anche nell'ambito della robotica, ad esempio utilizzando la visione artificiale per far muovere degli automi.
Inoltre potrebbe essere utilizzato per far volare un elicottero o per far muovere un piccolo veicolo, evitando ostacoli mediante la creazione di una mappa 3D dell'ambiente.
Il dispositivo permette anche di risparmiare enormi budget per la realizzazione un sistema di motion capture.
Infine si potrebbero avere applicazioni anche per l'intrattenimento e nel campo della medicina.

La prima cosa che bisogna fare per lavorare con questi sensori è analizzarne l'hardware e il supporto fornito agli sviluppatori. In secondo luogo bisogna scoprire i driver e le funzionalità fornite dalle librerie.

2.3 Hardware

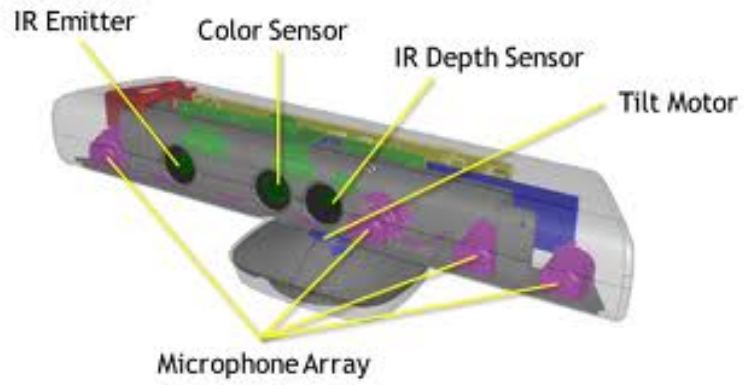

Figura 2.2: Componenti Kinect

Kinect è dotato di telecamera RGB, sensore di profondità a raggi infrarossi composto da un proiettore a infrarossi e da una telecamera sensibile alla stessa banda. La telecamera RGB ha una risoluzione di 640 x 480 pixel, mentre quella a infrarossi usa una matrice di 320 x 240 pixel.

É presente anche di un insieme di microfoni utilizzato dal sistema per la calibrazione dell'ambiente in cui ci si trova, mediante l'analisi della riflessione del suono sulle pareti e sull'arredamento. In tal modo il rumore di fondo e i suoni del gioco vengono eliminati ed è possibile riconoscere correttamente i comandi vocali.

La barra del Kinect è motorizzata lungo l'asse verticale e segue i movimenti dei giocatori, orientandosi nella posizione migliore per il riconoscimento dei movimenti. 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 usurabili.

Dato che le immagini vengono elaborate direttamente sul Kinect, 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.

Un particolare difficile da notare è la cella di Peltier posta tra l'IR e la barra metallica che svolge il ruolo di sistema di raffreddamento.

Tra la telecamera RGB e il proiettore IR è situato anche un piccolo LED di stato.

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.

Figura 2.3: Componenti del dispositivo Kinect

Per far sì 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.

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.

Ora che è stato descritto il sistema visivo ed uditivo del dispositivo bisogna capire come avviene l'elaborazione dei dati.

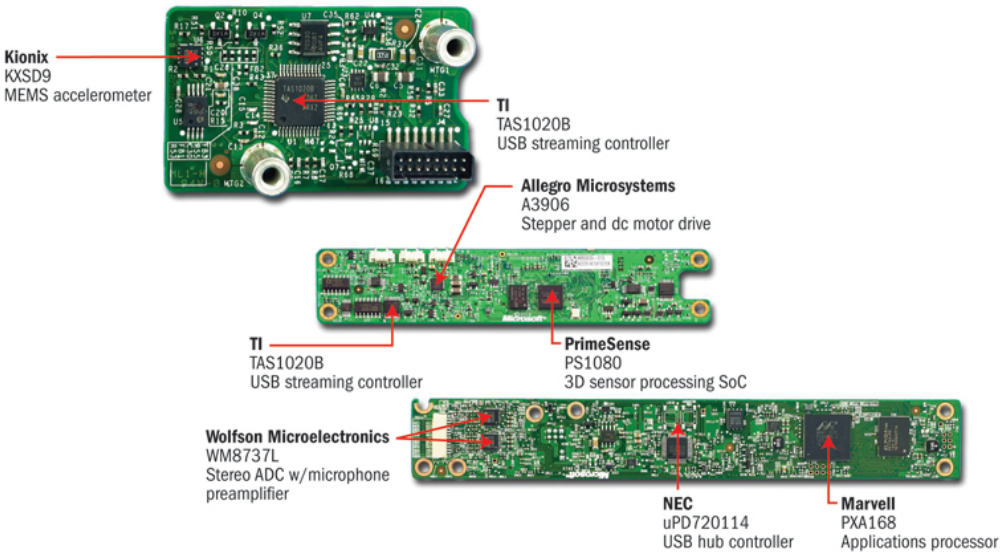

Figura 2.4: Scheda A - Scheda B - Scheda C

La scheda madre ha sei chip. Osservando la scheda C della figura , 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 - SPI flash (H1026567 XBOX1001 X851716-005 Gepp);
- SoC per il controller dell'interfaccia della macchina fotorafica (Marvell AP102);
- SDRAM DDR2 512 megabit (Hynix H5PS5162FF).

Nella scheda B sono montati:

- 2 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 la scheda A dispone di un controller audio USB frontale e centrale (TI TAS1020B) e sul lato sinistro della scheda si può vedere un accelerometro (Kionix MEMS KXSD9) che probabilmente è utilizzato come stabilizzatore d'immagine.

2.4 Caratteristiche tecniche

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; questi parametri di precisione vengono forniti direttamente da Microsoft.

Nella seguente tabella sono riportate le caratteristiche tecniche di Microsoft Kinect.

Campo visivo (in gradi)	58°H, 45°V, 70°D
Risoluzione x/y (a 2 m dal sensore)	3 mm
Risoluzione z (a 2 m dal sensore)	10 mm
Range di lavoro	0.8 m - 3.5 m
Interfaccia	USB 2.0
Consumo	2.25 W
Immagine di profondità	320 x 240 pixel
Immagine a colori RGB	640 x 480 pixel
Frame-rate	30 fps
Stream audio	4 canali 16 bit (fc 16KHz)

Tabella 2.1: Caratteristiche tecniche del Kinect

2.4.1 Requisiti Hardware

Requisiti minimi per poter usare il Kinect su PC:

- 32 bit (x86) or 64 bit (x64) processor

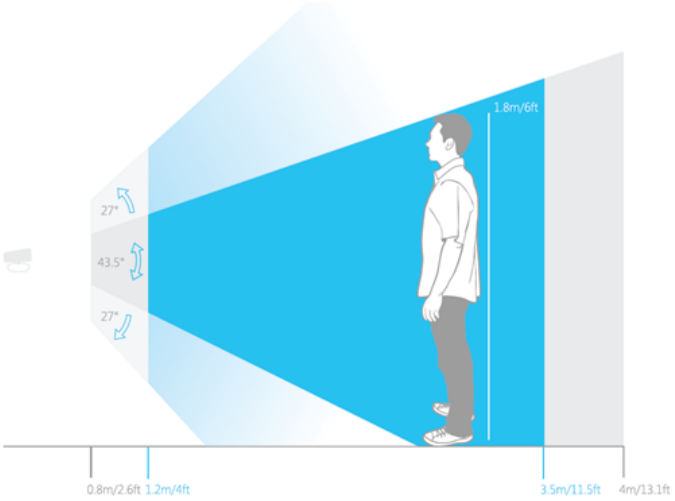

Figura 2.5: Campo di vista del Kinect

- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM

2.5 Kinect Development Software

Ad oggi sono presenti 4 principali Librerie di sviluppo Software per Kinect:

- OpenNI
- Microsoft's Kinect for Windows SDK
- OpenKinect
- CLNUI

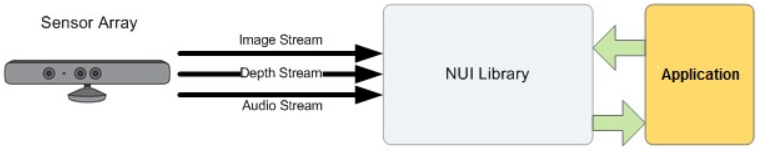

Figura 2.6: Interazione hardware e software con applicazioni

Questi framework consentono di accedere alle funzioni audio, video e ai sensori di profondità del Kinect attraverso diverse API.

Le applicazioni sviluppate per il Kinect non fanno uso dei dati provenienti direttamente dal sensore, ma si appoggiano a queste librerie le quali forniscono agli sviluppatori un maggior livello di astrazione e quindi una più semplice interpretazione dei dati provenienti dal sensore. In più queste librerie permettono anche di interagire con il dispositivo Kinect in modo da inviargli comandi da eseguire richiamando semplici metodi.

In particolare queste librerie permettono:

- Accesso ai flussi dati dei sensori di profondità, colore e audio.
- Tracciamento dello scheletro di una persona per creare programmi basati sui gesti.
- Capacità di processare l'audio con un nuovo sistema di rimozione del rumore e cancellazione dell'eco, identificando la fonte sonora e integrandola con le funzioni di riconoscimento vocale.
- Interagire con il dispositivo.

2.6 Kinect for Windows SDK Architecture

Vediamo un po più nello specifico i componenti dell'architettura dell' SDK della Microsoft arrivato alla versione 1.7:

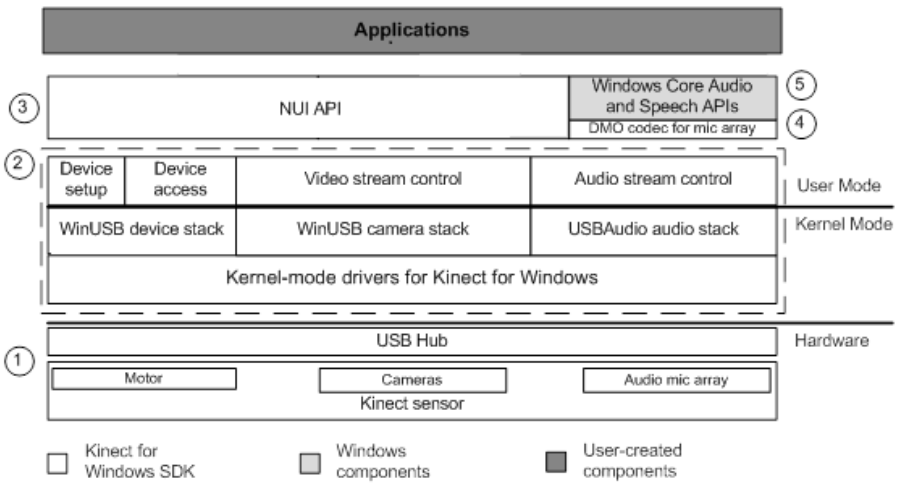

Figura 2.7: Kinect for Windows Architecture

1 Hardware L'hardware comprende i sensori visti in precedenza è l'hub USB che permette il loro collegamento al pc.

2 Microsoft Kinect drivers

a) Permettono l'accesso all'array di microfoni con le API Audio standard di Windows.

b) Forniscono gli stream della video camera e dei sensori di profondità.

c) Forniscono la possibilità di utilizzare più device contemporaneamente.

3 NUI API Un insieme di API che permettono di recuperare i dati dai sensori di immagine e di controllare il device stesso.

4 KinectAudio DMO Estende le funzionalità dell'array di microfoni supportato in Windows 7 per fornire le funzionalità di Beamforming (mappatura sonora dell'area) e localizzazione della sorgente sonora.

5 Windows 7 standard APIs Le API audio, speech e media presenti in Windows 7 e Microsoft Speech.

Nonostante le grandi funzionalità, questo framework è utilizzabile soltanto sotto piattaforme Microsoft. Per questo motivo ora ci concentreremo su un altro framework indipendente dalla piattaforma utilizzata.

Capitolo 3

OpenNI 2.0

OpenNI ovvero Open Natural Interaction è un framework sotto licenza GNU GPL indipendente dalle piattaforme di lavoro e multi-linguaggio che definisce le API per scrivere applicazioni che usano le Natural User Interface. 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.

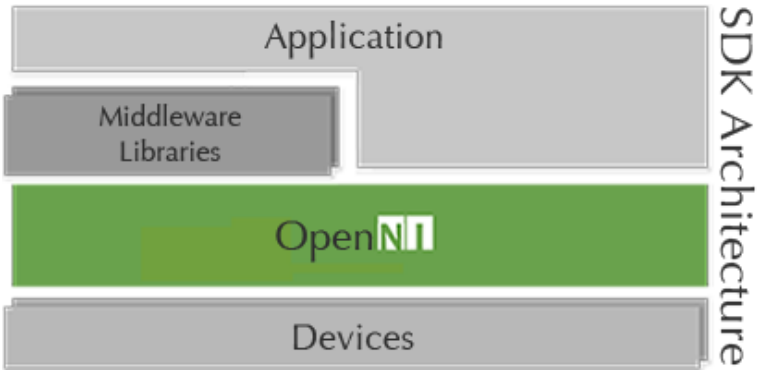

Figura 3.1: OpenNI Architecture (Presa dal sito di OpenNI)

Il middleware 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 tra i dispositivi fisici e componenti middleware.

Il livello più alto rappresenta il software che fa uso di OpenNI implementando le Natural User Interface.

Il livello centrale rappresenta l'interfaccia OpenNI con le sue capacità di comunicare con i vari sensori e con le funzionalità disponibili (Skeleton Tracking, Hand Tracking, 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:

- sensore 3D
- fotocamera RGB
- dispositivo audio (un microfono o un array di microfoni)
- Kinect

OpenNI ha rilasciato i driver e i propri codici sorgente per far sì che le sue librerie vengano implementate su più architetture possibili e su qualsiasi sistema operativo, in modo da accelerare l'introduzione di applicazioni che sfruttano le Natural User Interface sul mercato.

Con l'uscita del 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

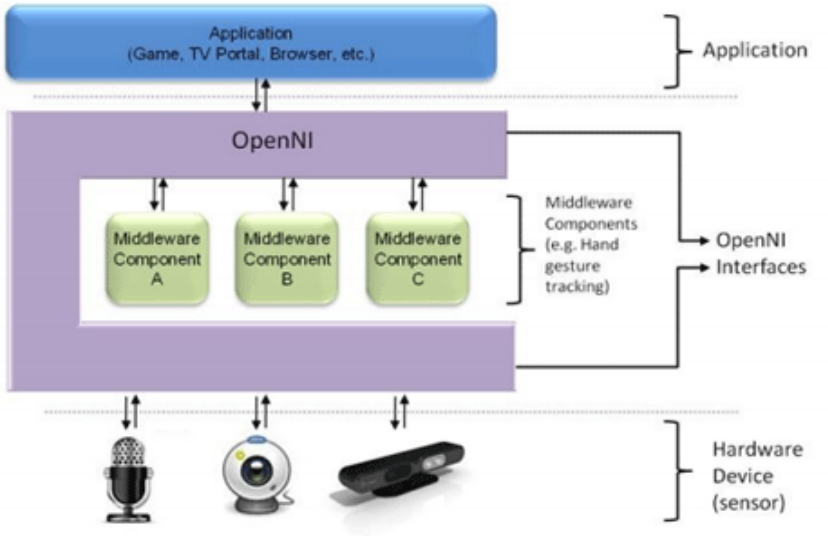

Figura 3.2: Astrazione dei livelli di funzionamento di OpenNI.

qualsiasi sensore ha solo facilitato l'uso del dispositivo Microsoft.

3.1 Middelware e NITE 2.0

OpenNI è dunque un livello di astrazione che consente di nascondere ai livelli superiori (nonchè ai moduli interni) la complessità tecnologica dei singoli sensori, di modo che non è strettamente legato a Kinect, bensì si pone come una soluzione software adatta ad essere utilizzata con qualunque tipo di tecnologia con un grado qualunque di complessità.

Inerentemente agli argomenti trattati in questa tesina, OpenNI non si limita a permettere di muovere il Kinect e catturare le immagini, bensì è corredato anche di un middleware di motion tracking chiamato NITE (sviluppato anch'esso da PrimeSense), per il riconoscimento delle mani e del corpo della persona.

La combinazione OpenNI NITE si basa sull utilizzo delle librerie OpenGL per visualizzare lo scheletro dell'utente. Il tracciamento dello scheletro avviene tramite una

posizione "chiave", ovvero lutente deve rimanere in posizione eretta, allargare le braccia e posizionare gli avambracci con un a ngolo di novanta gradi circa rispetto alle braccia.

Il suo scopo è quello di rendere più facile la vita agli sviluppatori introducendo un livello di astrazione superiore, in modo che possano concetrarsi sullo scopo specifico della lora applicazione da sviluppare. Infatti la libreria OpenNI restituisce dati a basso livello come mappe di profondità, mappe di colori, audio e quant'altro. NITE invece lavora ad un livello superiore e fornisce funzionalità aggiuntive e facilita la creazione di applicazioni di controllo basato sul movimento delle mani dell'utente e sul rilevamento dello scheletro.

3.2 Componenti del Middelware

Componenti del Middelware:

Full body point analysis middleware: 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.

Hand point analysis middleware: è un componente software che elabora i dati sensoriali, individua la sagoma di una mano assegnando un punto alla posizione del palmo.

Gesture detection middleware: è un componente software che identifica gesti predefiniti (Push, Wave, Circle) associandoli ad eventi.

Scene Analyzer middleware: 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.

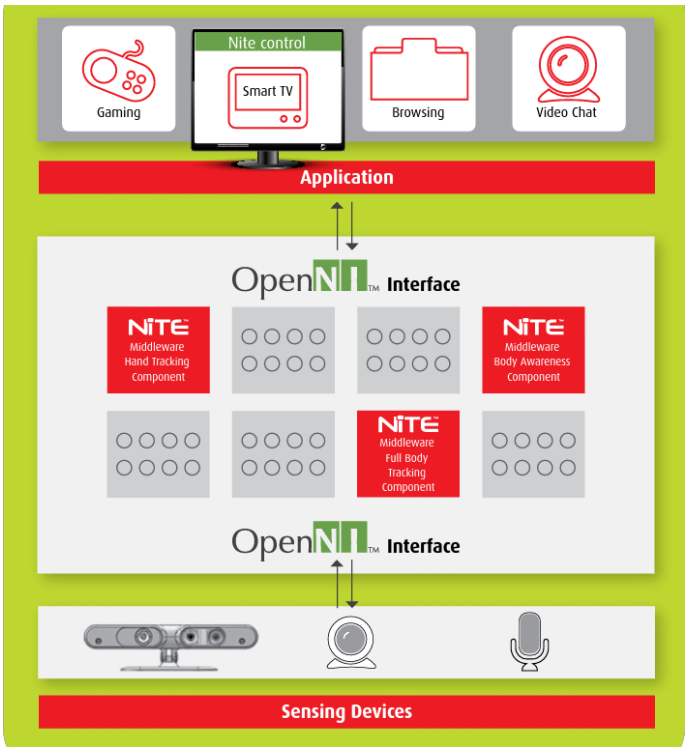

Figura 3.3: Componenti NITE.

3.3 Funzionamento del sensore

[5] Per capire come funziona la periferica è possibile dividere il sistema in tre sotto blocchi: il monitoraggio dei movimenti, il riconoscimento vocale ed il motore.
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 RGB.
La prima cosa che la periferica fa è creare una mappa di profondità della scena separando l'utente dagli oggetti inanimati.

A seconda della distanza dal sensore, 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 creare la mappa di profondità 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 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 3.4: Fascio di raggi infrarossi per la creazione della mappa di profonditá

L'immagine acquisita dal sensore viene fatta passare in diversi filtri, in modo tale che il dispositivo possa capire cosa è una persona e cosa non lo è. L'intero sistema segue delle linee guida, riguardanti la conformazione generale del corpo. Questo permetterà in fase di calibrazione di non confondere gli oggetti con le persone. Non tutte le persone hanno però la stessa conformazione fisica, inoltre spesso vengono utilizzati indumenti larghi o cappelli. Per questo vengono inseriti tra le linee guida degli algoritmi di riconoscimenti di 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, l'algoritmo 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 pronti per essere usati a tale scopo.

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 trovata nell'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 agisce con la telecamera per capire dove si sta creando una possibile fonte di suoni intorno all'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

Figura 3.5: Posizionamento array di microfoni

sono in ascolto in ogni momento rendendo il sistema Kinect open-mic.

A questo punto rimane da capire come funziona il sotto blocco 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.

3.3.1 Calcolo della mappa di profondità

Le immagini di profondità semplificano molti problemi di computervision e di interazione come ad esempio:

- rimozione del background e segmentazione della scena;
- tracking di oggetti e persone;
- ricostruzione 3D degli ambienti;
- riconoscimento della posa del corpo;
- implementazione di interfacce basate su gesti.

La mappa di profondità della scena è un'immagine M di dimensione mxn, in cui ciascun pixel p(x, y) codifica la distanza nella scena 3D del punto (x,y) dal sensore.

In letteratura esistono molte tecniche per calcolarla e le più utilizzate sono:

La stereo triangolazione che calcola la profondità di un oggetto combinando le immagini catturate da due telecamere;

La tecnica time of flight che invece utilizza solo una telecamera calcolando la distorsione che subisce un segnale luminoso proiettato sugli oggetti;

La proiezione di pattern tecnica utilizzata sul Kinect.

Questa ultima tecnica utilizza un sistema di visione stereo costituito da una coppia proiettore-telecamera. Nella scena viene proiettato un pattern luminoso (infrarosso) noto e la profondità degli oggetti è calcolata studiando la sua distorsione sugli oggetti.

É possibile implementare questa tecnica con varie tecnologie:

- proiezione di linee e studio della loro curvatura sugli oggetti: non molto veloce e soggetta a disturbi quando gli oggetti sono in movimento;
- proiezione di pattern 2D periodici e studio del loro scostamento quando colpiscono gli oggetti: l'informazione 3D è ottenuta in real-time ma non è in grado di lavorare su lunghe distanze per via della distorsione del pattern;
- proiezione di pattern 2D pseudo-casuali: anche in questo caso i pattern sono 2D ma la loro casualità permette di ottenere accurate mappe 3D in real-time con un sistema molto semplice ed economico.

Nel sensore Kinect la mappa di profondità della scena viene costruita utilizzando la tecnica della proiezione di pattern pseudo-casuali, mediante un sistema di visione stereo costituito da un proiettore IR e da una telecamera sensibile alla stessa banda.

Questa tecnologia è stata brevettata nel 2005 da Zalevsky, Shpunt, Maizels e Garcia, sviluppata e integrata in un chip dalla compagnia israeliana PrimeSense.

Questa tecnica si basa su 3 elementi principali:

1. proiettore di pattern pseudo-casuali IR;

2. telecamera IR (in tecnologia CMOS);

3. unità di controllo (chip PS1080).

Il proiettore è molto semplice ed economico ed è costituito da un emettitore di raggi IR e da un generatore di pattern che devia tali raggi nella scena imprimendo ad essi angolazioni pseudo-casuali.

Una volta proiettato il pattern, la telecamera acquisisce l'immagine IR della scena contenente il pattern distorto e la invia all'unità di controllo che costruisce così la mappa di profondità della scena.

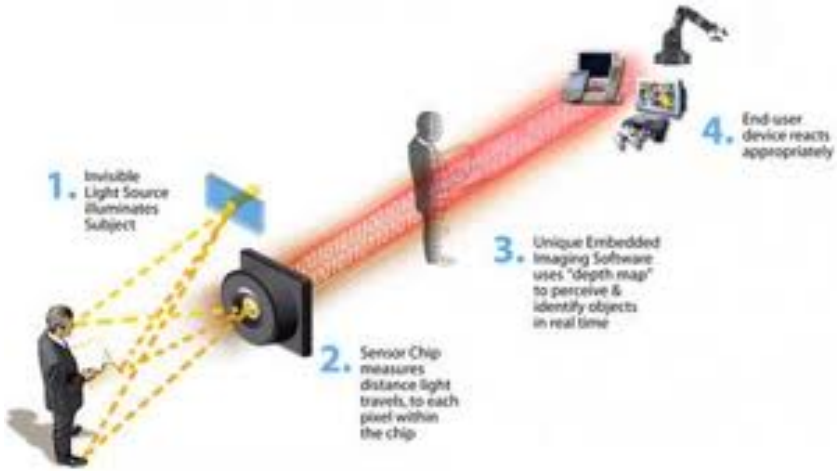

Figura 3.6: Tecnica per il calcolo della mappa di profondità.

Con questo sistema è necessario acquisire una singola immagine e quindi utilizzare un singolo algoritmo di matching per determinare la profondità degli oggetti (dato che l'altra immagine è costituita dal pattern originale che è noto).

L'unità di controllo, conoscendo la struttura del pattern proiettato, calcola lo scostamento fra i punti proiettati e quelli ripresi dalla telecamera determinando in questo modo la mappa di profondità della scena.

La dimensione dei punti proiettati, la loro forma e orientazione non è costante ma dipende dalla distanza dal sensore.

Il brevetto individua tre differenti tipologie di punti per tre differenti regioni dello spazio come mostrato in figura 3.7:

Una prima regione R1 (0.8 - 1.2 m) in cui si ha la massima risoluzione, una seconda regione R2 (1.2 - 2 m) con una buona accuratezza e una terza regione R3 (2 - 3.5 m) dove l'accuratezza è scarsa.

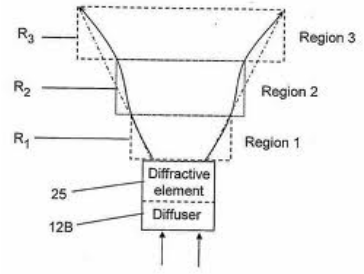

Figura 3.7: Regioni dello spazio individuate dal brevetto PrimeSense.

PrimeSense non ha solo sviluppato un nuovo sistema di acquisizione della mappa 3D della scena, ma soprattutto una tecnologia capace di elaborare questi dati realizzando molti task di processing 3D. Infatti il chip PS1080 ha al suo interno molte funzionalità di processing per il tracciamento, per la ricostruzione della scena e per il riconoscimento di gesti.

3.4 OpenNI Programmer's Guide

[3] Questa sezione fornisce una panoramica sulle API fornite dall' SDK di OpenNI 2.0.

3.4.1 Overview

Le API di OpenNI 2.0 forniscono l'accesso ai sensori del Kinect. Queste consentono a un'applicazione di inizializzare il sensore e ricevere dati di profondità, RGB e flussi video a infrarossi dal dispositivo, forniscono anche una singola interfaccia unificata per tutti i sensori.

Ottenere l'accesso ai flussi di profondità necessita l'utilizzo di quattro classi principali. Questo elenco è inteso come una breve introduzione. Ognuna di queste classi

saranno discussi in dettaglio nella propia sezione:

1) openni::OpenNI - fornisce un unico punto di ingresso statico per le API. Consente inoltre di accedere ai dispositivi, agli eventi correlati ai dispositivi e alle informazioni di errore e di versione. Necessario per consentire di connettersi a un dispoditivo;

2) openni::Device - fornisce un' interfaccia per ogni singolo sensore collegato al sistema. Richiede OpenNI per essere creato e inizializzato. Fornisce l'accesso ai vari stream di dati;

3) openni::Stream - permette l'accesso a un singolo stram di dati ottenuto da un dispositivo specifico. Richiede FrameRefs.

4) openni::FrameRef - permette l'accesso ai metadati relativi a un singolo stream di dati.

In aggiunta a queste classi principali sono presenti altre varie classi di supporto e varie strutture per la presenza di specifici tipi di dati. É fornita anche una classe Recorder per la memorizzazione di flussi di dati su file. Sono fornite anche classi di ascolto per gli eventi che OpenNI e le classi di Stream possono generare.

Gli Stream di dati possono essere letti con uno dei due metodi base: Loop Based ed Event Based. Entrambi questi metodi saranno esposti in dettaglio più avanti.

3.4.2 Classe OpenNI

La prima delle principali classi che compongono OpenNI 2.0 è OpenNI::OpenNI. Questa classe fornisce un punto di ingresso statico per le API. É usata per fornire l'accesso a tutti i dispositivi del sistema. Inoltre rende disponibili eventi di connessione e disconnessione dei dispositivi, oltre a fornire funzioni che consentono l'accesso a tutti

i flussi di dati basato sul polling.

Accesso ai sensori

La funzione OpenNI::initialize() inizializza tutti i driver dei sensori disponibili e esegue la scansione del sistema per rilevare i dispositivi disponibili. Qualsiasi applicazione che utilizza OpenNI dovrebbe richiamare questa funzione prima di ogni altra.

Una volta che è stata eseguita la funzione di inizializzazione, sarà possibile creare oggeti di tipo Device e usarli per comunicare con il sensore specifico. La funzione OpenNI::getDeviceInfoList() restituisce una lista di tutti i dispositivi disponibili collegati al sistema.

Quando l'applicazione deve terminare, la funzione OpenNI::shutdown() dovrebbe essere richiamata per spegnere tutti i driver dei sensori collegati al sistema.

Accesso agli stream dati

Una tecnica di polling per l'accesso agli stream dati può essere implementato utilizzando la funzione OpenNI::waitForAnyStream(). Questa funzione richiede un elenco di stream come uno dei suoi argomenti di Input. Quando viene chiamata questa funzione, l'applicazione si blocca fino a quando uno qualsiasi degli stream in lista dispone di nuovi dati. Quindi restituisce un codice di stato e indica quale stream ha reso disponibile i nuovi dati. Questa funzione può essere utilizzata per implementare un ciclo di polling per nuovi dati disponibili.

Accesso ai sensori tramite eventi

La classe OpenNI fornisce la possibilità di accesso ai vari sensori collegati attraverso gli eventi. La funzione OpenNI::adListener() e OpenNI::removeListener() consentono di aggiungere e rimuovere oggetti alla event listener gestita da OpenNI. La classe

openni::OpenNI::Listener fornisce la possibilità per rispondere a tali eventi.

OpenNI definisce 3 eventi:

onDeviceConnected un evento onDeviceConnected viene generato ogni volta che un nuovo dispositivo viene collegato ed è disponibile;

onDeviceDisconnected un evento onDeviceDisconnected viene generato quando un dispositivo viene rimosso dal sistema;

onDeviceStateChanged un evento onDeviceStateChanged viene chiamato ogni volta che vengono modificate le impostazioni del dispositivo.

Tutti e tre gli eventi offrono un puntatore ad un oggetto OpenNI::DeviceInfo. Questo oggetto può essere utilizzato per ottenere informazioni e per l'identificazione del dispositivo a cui si riferisce l'evento. Inoltre, l'evento onDeviceStateChanged fornisce un puntatore a un oggetto DeviceState che può essere utilizzato per visualizzare il nuovo stato del dispositivo.

L'accesso agli stream tramite eventi sarà descritto nella sezione che riguarda la Classe Stream.

Informazione di Errore

La funzione OpenNI::getExtendedError() fornisce una traduzione leggibile dei codici di stato numerici. Molte funzioni del SDK hanno come tipp di ritorno "Status". Quando si verifica un errore, lo stato conterrà un codice che può essere registrato o visualizzato da un utente. Traducendo il codice in una stringa di descrizione utilizzando la funzione getExtendedError() sarà possibile capire chi ha generato l'errore.

Informazione di Versione

Le informazioni di versione sono fornite da OpenNI::getVersion(). Questa funzione restituisce la versione delle API con cui l'applicazione attualmente interagisce.

3.4.3 Classe Device

La classe OpenNI::Device fornisce un' interfaccia per ogni singolo dispositivo hardware collegato al sistema.

Lo scopo fondamentale dei dispositivi è quello di fornire degli Stream di dati. L'oggetto Device è utilizzato per collegare e cnfigurare i vari dispositivi e quindi da questi ottenere gli Stream dati.

Prerequisiti per connettere un Device

Prima di poter utilizzare la classe Device e collegarla ad un dispositivo hardware, il dispositivo deve essere fisicamente collegato al sistema e i driver devono essere installati correttamente. I driver per i sensori PrimeSense vengono installati insieme a OpenNI 2.0.

É inoltre necessario che la funzione openni::OpenNI::initialize() sia stata richiamata dopo aver collegato i dispositivi. Questa funzione come detto in precedenza inizializza i driver e rende disponibili le API agli eventuali dispositivi collegati.

Costruttore

Il costruttore per la classe Device non ha argomenti, e non collega l'oggeto a un dispositivo hardware fisico. Esso crea semplicemente l'oggetto in memoria in modo che altre funzioni possono essere richiamate.

Funzione Device::open()

La funzione Device::open() serve per collegare un oggetto Device a un dispositivo hardaware fisico. La funzione open() accetta un solo argomento l'URI di uno dispositivo specifico. Questa funzione restituisce un codice di stato che indica se la funzione è stata eseguita con successo o se si è verificato qualche errore.

Per un utilizzo più semplice di questa funzione viene utilizzata la costante openni::ANY_DEVICE come URI. Utilizzando questa costante l'oggetto Device si connetterà a un qualsiasi dispositivo hardware attivo nel sistema. Ciò è molto utile quando si può presumere con sicurezza che c'è esattamente un dispositivo hardware attivo collegato al sistema.

Se più sensori sono collegati al sistema, allora si deve prima chiamare la funzione OpenNI::getDeviceInfoList() per ottenere un elenco di tutti i dispositivi attivi. Quindi, trovare il dispositivo desiderato nella lista, e ottenere il suo URI chiamando la funzione DeviceInfo:getUri(). Utilizzare l'output di questa funzione come parametro richiesto dalla funzione open(), al fine di aprire quel dispositivo specifico.

Funzione Device::close()

La funzione close() chiude correttamente il collegamento con il dispositivo hardware. Come buona pratica, qualiasi collegamento ad ogni dispositivo che viene aperto deve essere chiuso. Questo lascierà i driver e il dispositivo hardaware in uno stato corretto in modo che le appliczioni future non avranno difficoltà a connettersi ad esso.

Funzione Device::isValid()

La funzione isValid() può essere utilizzata per determinare se vi è attualmente un dispositivo attivo collegato ad un oggetto Device.

Ottenere informazioni su un dispositivo

É possibile ottenere informazioni di base su un dispositivo. Le informazioni disponibili comprendono nome, vendor string, uri e USB VID-PID. La classe openni::DeviceInfo è prevista per contenere tutte queste informazioni. Essa fornisce delle funzioni getterper ogni informazione disponibile. Per ottenere l'oggetto DeviceInfo specifico per un dato dispositivo bisogna richiamare la funzione getDeviceInfo().

Un dispositivo può essere costituito da più sensori. Ad esempio il Kinect ha un sensore a infrarossi, un sensore a colori e un sensore di profondità. Gli stream di dati possono essere aperti su un qualsiasi sensore esistente.

É possibile ottenere un elenco dei sensori disponibili da un dispositivo. La funzione Device::hasSensor() può essere utilizzata per interrogare se un dispositivo offre un sensore specifico. Sensori possibili sono:

SENSOR_IR - sensore video a infrarossi - IR;

SENSOR_COLOR - sensore videoa colori - RGB;

SENSOR_DEPTH - sensore video di profondità.

Se il sensore desiderato è disponibile, allora la funzione Device::getSensorInfo() può essere usata per ottenere informazioni specifiche su di esso. Queste informazioni saranno contenute all'interno di un oggetto SensorInfo il quale fornisce funzioni getter per il tipo di sensore e per un array che contiene le modalità video disponibili. Le informazioni riguardo le modalità video saranno a sua volta contenute nella classe VideoMode.

Funzionalità specifiche per i Device - Registrazione

Alcuni dispositivi producono sia stream di profondità sia stream a colori. Di solito, questi flussi sono prodotti con due diverse telecamere fisiche. Poichè le telecamere sono situate in punti separati nello spazio, forniscono immagini della stessa scena da due angolazioni diverse. Ciò si traduce in un unica immagine che sembra avere una posizione

apparentemente diversa dello stesso oggetto.

Se le relazioni geometriche tra le due telecamere e la distanza dell'oggetto in questione sono entrambi noti, allora è possibile trasformare matematicamente una delle due immagini per farla sembrare ripresa dallo stesso punto di osservazione dell'altra telecamera. Ciò consente di sovrapporre un'immagine ad un'altra, ad esempio per colorare un'immagine di profondità con i colori di un'immagine RGB. Questo processo è chiamato Registrazione.

Alcuni dispositivi sono in grado di eseguire queste operazioni via hardware. Se possiede quese capacità allora ci saranno dei dispositivi hardware per abilitarlo o disabilitarlo.

L'oggetto Device fornisce la funzione isImageRegistrationSupported(); per verificare se il dispositivo specifico collegato supporti la Registrazione. Se è supportata, allora la funzione getImageRegistrationMode() può essere utilizzata per richiedere lo stato attuale di questa funzione, e setImageRegistrationMode() può essere usata per impostare un valore specifico. L'enumerato openni::ImageRegistrationMode fornisce i possibili valori che possono essere passati alle due funzioni get e set:

IMAGE_REGISTRATION_OFF - la funzione di registrazione hardware viene disabilitata;

IMAGE_REGISTRATION_DEPTH_TO_IMAGE - l'immagine di profondità viene trasformata per avere lo stesso punto di vista di un'immagine RGB.

Si noti che, poichè i due sensori avranno zone dove il loro campo di vista non si sovrappone, ci sarà generalmente una zona da un lato della mappa di profondità non raffigurato come risultato desiderato. É anche comune vedere "ombre" o "buchi" nella mappa di profondità dove ci sono spigoli improvvisi nello spazio reale. Questo è causato dal fatto che sono presenti errori di approsimazione durante la trasformazione dell'immagine.

Funzionalità specifiche per i Device - FrameSync

Quando entrambi gli stream di profondità e di colore sono disponibili, è possibile che i singoli fotogrammi di ogni flusso non siano esattamente sincronizzati tra loro. Questo può essere causato da una leggera differenza di frame rate o da una leggere differenza di tempo di arrivo del frame.

Alcuni dispositivi forniscono la capacità di eseguire la sincronizzazione hardware dei due frame, al fine di ottenere fotogrammi che sono separati l'uno dall'altro al massimo di un tempo t garantito. Solitamente, tale tempo massimo è molto inferiore al tempo tra due fotogrammi successivi. Questa capacità è indicata come FrameSync.

Per attivare o disattivare questa capacità basta richiamare la funzione setDepthColorSyncEnabled().

Funzionalità specifiche per i Device - File Devices

OpenNI 2.0 offre la possibilità di collegare l'uscita di un dispositivo ad un file (denominato file ONI, e di solito hanno l'estensione .oni). Questo collegamento potrà eventualmente includere tutti i flussi prodotti dal dispositivo, insieme a tutte le impostazioni abilitate al momento della registrazione del dispositivo.

Fatta eccezione per lievi differenze nel modo in cui vengono inizializzati (passando un nome di un file alla funzione Device::open()), l'utilizzo di un device fisico o di un File Device sono indistinguibili.

Questa funzionalità può essere molto utile per algoritmi di debug. Scene dal vivo sono generalmente difficili o impossibili da riprodurre esattamente. Utilizzando questa funzionalità lo stesso input può essere utilizzato per più di un algoritmo, consentendo di fare vari controlli e misurazioni di prestazioni. Questa funzionalità può essere anche utile per il test automatizzato di applicazioni, e per situazioni in cui le telecamere disponibili sono insufficienti per tutti gli sviluppatori di un progetto.

La classe PlaybackControl viene utilizzata per accedere a qualsiasi funzionalità specifica per i file devices. Vedere la sezione seguente per ulteriori informazioni. Per facilitare la scrittura di codice di uso generale che si occupa di entrambi i casi, dispositivi fisici e file device, è stata fatta la funzione Device::isFile() che permette alle applicazioni di determinare se un dispositivo è stato creato da un file prima di tentare di usare la classe PlaybackControl.

3.4.4 Classe PlaybackControl

Ci sono alcune azioni che sono possibili solo quando si ha che fare con un file device. Queste funzioni includono: cercare all'interno di un flusso, determinare quanto tempo è durata una registrazione, "looppare" la registrazione e cambiare la velocità di riproduzione. Queste funzionalità sono state introdotte nella classe PlaybackControl.

Inizializzazione

Per utilizzare la classe PlaybackControl, è necessario prima creare un'istanza della classe Device e inizializzarla con un file. Una volta che un file device è stato creato, è posibile acquisire l'oggetto interno PlaybackControl. La funzione Device::isFile() può essere utilizzata per determinare se un dato device è statocreato da un file o no.

Ricerca all'interno di una registrazione

Due funzioni sono forniti per consentire ricerca all'interno di una registrazione.

La funzione PlaybackControl::seek() richiede un puntatore VideoStream e un frameID come Input. Imposta quindi la riproduzione della registrazione al fotogramma indicato.Se ci sono più flussi in una registrazione, tutti i flussi verranno impostati nello stesso punto indicato.

La funzione PlaybackControl::getNumberOfFrames() può essere utilizzata per determinare quanto tempo dura una registrazione. Questo è utile principalmente per determinare gli obiettivi per la funzione di ricerca. Essa vuole come Input un VideoStream e restituisce il numero di fotogrammi della registrazione per il flusso specificato.

Playback Speed

É possibile variare la velocità di riproduzione di una registrazione. Ciò è utile per testare un algoritmo con un ampio insieme di dati di input, in quanto permette di ottenere risultati più velocemente.

Per impostare la velocità è stata creata la funzione PlaybackControl::SetSpeed() che come Input vuole un valore in virgola mobile. Il valore di ingresso viene interpretato come un multiplo della velocità con cui è stata effettuata la registrazione. Ad esempio, se la registrazione è stata fatta a 30 fps, e un valore di 2,0 è stato passato alla funzione SetSped(), allora il flusso viene riprodotto a 60 fps. Se viene passato un valore pari a 0,5, il flusso viene riprodotto a 15 fps.

L'impostazione della velocità a 0,0 farà si che il flusso scorra alla velocità impostata nel sistema che ospita l'applicazione. Impostare la velocità a -1 farà si che il flusso venga letto manualmente dall'applicazione. Impostare la velocità a -1 e leggere manualmente il flusso in un ciclo, sarà simile a impostare la velocità a 0,0.

La funzione PlaybackControl::getSpeed() fornisce il valore più recente con cui è stato impostato setSpeed() (vale a dire, il valore della velocità attivo).

Playback Looping

Un sensore fisico normalmente continua a fornire dati indefinitamente, mentre un device file ha solo un numero finito di fotogrammi. Questo può essere problematico

quando si tenta di utilizzare una registrazione per simulare un sensore fisico, poichè il codice delle applicazioni sviluppato per affrontare sensori fisici non gestisce generalmente la fine di una registrazione.

Per superare questa difficoltà, è stata fatta una funzione di loop per la riproduzione ciclica della registrazione. La funzione PlaybackControl::setRepeatEnabled() può essere utilizzato per attivare la riproduzione ciclica della registrazione. Se viene passato il valore TRUE alla funzione la registrazione avrà inizio dal primo fotogramma e dopo la lettura dell'ultimo fotogramma ricomincerà la riproduzione dal primo fotogramma della registrazione. Se viene passato il valore FALSE la registrazione sarà riprodotta normalmente.

PlaybackControl::getRepeatEnabled() può essere utilizzato per interrogare il valore di ripetizione corrente.

3.4.5 Classe VideoStream

La classe VideoStream rappresenta tutti gli stream di dati creati dalla classe Device. Consente di avviare, arrestare e configurare stream di dati specifici.

Creazione e Inizializzazione

Chiamando il costruttore predefinito della classe VideoStream verrà creato un oggetto vuoto non inizializzato. Prima dipoter essere utilizzato, questo oggetto deve essere inizializzato con la funzione VideoStream::create(). Questa funzione richiede un Device inizializzato e valido. Una volta creato, è necessario chiamare la funzione VideoStream::start() per avviare il flusso di dati. La funzione VideoStream::stop() è fornita per fermare il flusso di dati.

Lettura dei dati basata sul polling

Una volta che un VideoStream è stato creato, i dati possono essere letti direttamente con la funzione VideoStream::readFrame(). Se sono disponibili nuovi dati, questa funzionefornisce l'accesso al più recente VideoFrameRef generato dal VideoStream. Se nessun nuovo frame è ancora pronto, allora questa funzione si bloccherà fino a quando un nuovo frame è pronto.

Si noti che se la lettura avviene da una registrazione con Looping spento, questa funzione si blocca per sempre una volta che è stato raggiunto l'ultimo fotogramma.

Lettura dei dati basata su eventi

É anche possibile leggere i dati da un VideoStream attraverso gli eventi. Per fare questo, è necessario creare una classe che estende la classe VideoStream::Listener. Questa classe deve implementare una funzione chiamata onNewFrame(). Una volta creata una istanza di questa classe bisogna passarla alla funzione VideoStream::addListener(). Quando un nuovo frame è disponibile, la funzione onNewFrame() del nostro ascoltatore sarà richiamta. Sarà necessario chiamare anche readFrame().

Ottenere informazioni su videostream

Le classi SensorInfo e VideoMode sono previste per tenere traccia delle informazioni sui videostream. Un VideoMode contiene informazioni sul frame rate, risoluzione e formato pixel di un VideoStream. SensorInfo contiene il tipo di sensore utilizzato per produrre lo Stream Video, e un elenco di oggetti VideoModel. Scorrendo l'elenco di oggetti VideoMode, è possibile determinare tutte le possibili modalità per la produzione di un dato sensore VideoStream.

Utilizzando la funzione VideoStream::getSensorInfo() si ottengono le informazioni del sensore corrispondente a un determinato VideoStream.

Campo visivo

Sono fornite alcune funzioni per determinare il campo di vista del sensore utilizzato per creare un certo VideoStream. Utilizzando le funzioni getVerticalFieldOfView() e getHorizontalFieldOfView() si può determinare il campo visivo utilizzato per creare lo stream.Questo valore verrà riportato in radianti.

Valore minimo e massimo dei pixel

Per gli stream di profondità è spesso utile conoscere i possibili valori minimi e massimi che un pixel può contenere. Utilizzando le funzioni getMaxPixelValue() e getMinPixelValue() si otengono queste informazioni.

Configurazione VideoStream

É possibile configurare un VideoStream impostando il frame rate, la risoluzione e il tipo di pixel di un determinato stream. Per fare questo, si utilizza la funzione setVideoMode().

Cropping

Se un dato sensore supporta questa funzione, la VideoStream fornisce un mezzo per poterla utilizzare. La funzione VideoStream::isCroppingSupported() determinare se un sensore supporta il Cropping.

Se il Cropping è supportato, la funzione setCropping() è usata per abilitarlo e configurare le impostazioni di Cropping desiderate. La funzione resetCropping() può essere utilizzata per riattivare il Cropping di nuovo. La funzione getCropping() può essere utilizzata per ottenere le impostazioni di Cropping correnti.

Mirroring

Il mirroring fa si che il VideoStream appaia come se visto allo specchio, vale a dire, l'immagine viene trasformata riflettendo tutti i pixel lungo l'asse verticale. Per attivare o disattivare il mirroring si utilizza la funzione VideoStream::setMirroringEnabled(). Passando TRUE per attivare il mirroring o FALSE per disattivarlo. L'impostazione di mirroring attuale può essere interrogato tramite la funzione getMirroringEnabled().

3.4.6 Classe VideoFrameRef

La classe VideoFrameRef contiene tutti i dati relativi a un singolo fotogramma letto da un VideoStream. É la classe base che utilizza VideoStream per restituire un nuovo fotogramma. Essa fornisce l'accesso al'array sottostante che contiene i dati del frame, oltre ai metadati necessarie per lavorare con il frame.

Oggettidella classe VideoFrameRef sono ottenuti richiamando la funzione VideoStream::readFrame().

Dati di tipo VideoFrameRef possono provenire da telecamere a infrarossi, telecamere RGB o da telecamere di profondità. Se necessario, la funzione getSensorType() può essere utilizzata per determinare quale tipo di sensore a generato il frame. Restituisce un enumerato SensorType che fornisce un codice per ogni possibile tipo di sensore.

Accesso ai dati del Frame

La classe VideoFrameRef include la funzione VideoFrameRef::getData(), che restituisce un puntatore diretto ai dati di frame sottostanti.

Metadati

Diversi metadati sono presenti nei frame per facilitare il lavoro sui frame stessi.

Dati di Cropping

La classe VideoFrameRef contiene le informazioni di Cropping per il VideoStream utilizzato per ottenere il frame. É possibile determinare l'origine della finestra di Cropping, dimensione della finestra di Cropping e se è stato abilitata quando è stato generato il frame. Le funzioni includono: getCropOriginX(), getCropOriginY() e getCroppingEnabled(). La dimensione della finestra di Cropping sarà uguale alla dimensione della cornice di Cropping se è stata abilitata, quindi il metodo di determinazione è lo stesso del metodo di determinazione della risoluzione del frame.

TimeStamp

Ogni frame sarà timbrato con un timestamp. Questo valore è misurato in microsecondi da qualche valore di zero arbitrario. La differenza tra il timestamp di due fotogrammi dello stesso flusso sarà il tempo trascorso tra questi fotogrammi. Tutti i flussi nello stesso dispositivo sono garantiti per utilizzare lo stesso valore zero, quindi le differenze tra i timestamp possono anche essere utilizzati per confrontare fotogrammi da diversi flussi.

L'implementazione corrente di OpenNI 2.0 è avviare il timestamp zero come il tempo del primo fotogramma del flusso.

Frame Indexes

Oltre al timestamp, i frame sono forniti di un indice sequenziale. Questo è utile per determinare la sequenza dei fotogrammi e per sapere quanti fotogrammi sono passati tra due specifici frame. Se due flussi sono stati sincronizzati utilizzando la funzione Device::setColorDepthSync() si può facilmente capire quali frame sono in relazione tra loro.

Se la sincronizzazione non è abilitata, quindi non è possibile abbinare i frame. É utile utilizzare il timestamp per determinare quali frame sono in relazione tra di loro.

Video Modes

VideoFrameRef::getVideoMode() può essere utilizzata per determinare le impostazioni del sensore che ha generato quel determinato frame. Queste informazioni includono il formato dei pixel, la risoluzione dell'immagine, il frame rate della telecamera.

Data Size

La funzione getDataSize() può essere utilizzata per determinare la dimensione di tutti i dati contenuti nel'array dell'immagine. Questo è utile se avete bisogno di allocare un buffer per memorizzare il fotogramma, o una serie di fotogrammi. Notare che questa è la dimensione dei dati totali dell'array. Utilizzando VideoMode::getPixelFormat() per determinare la dimensione dei singoli elementi dell'array.

Risoluzione dell'immagine

Per comodità, le funzioni getHeight() e getWidth() sono fornite per determinare facilmente la risoluzione del frame. Questi dati potrebbero essere ottenuti anche con VideoFrameRef::getVideoMode().getResolutionX() e VideoFrameRef::getVideoMode().getResolutionY(), ma questi valori sono utilizzati molto frequentemente, quindi le chiamate sopra sarebbero inefficienti e scomode.

Validità dei dati

La funzione VideoFrameRef::isValid() è fornita per determinare se un frame contiene effettivamente dati validi. Questa funzione restituisce false se non sono presenti dei frame.

Tipo di sensore

Il tipo di sensore utilizzato per generare il frame può essere determinato chiamando getSensorType(). Ciò restituirà un enumerato SensorType che assegna costanti per ogni possibile tipo di sensore. I valori possibili sono:

SENSOR_IR - per i frame generati da un sensore a infrarossi IR;

SENSOR_COLOR - per i frame generati da un sensore RGB;

SENSOR_DEPTH - per i frame generati da un sensore di profondità.

3.4.7 Classe Recorder

Una semplice classe Recorder viene fornita per facilitare la registrazione dei dati in un file VideoStream ONI.I file ONI sono lo standard di OpenNI per registrare l'output di un sensore su un file. Essi possono contenere uno o più flussi di informazione (ad esempio uno stream di profondità e uno stream RGB registrati simultaneamente da un sensore PrimeSense). Essi contengono anche le impostazioni del dispositivo utilizzato per creare tali informazioni.

Ci sono tre passaggi fondamentali per la creazione di un Recorder.

In primo luogo, è necessario costruire il Recorder chiamando il costruttore predefinito.

In secondo luogo, è necessario chiamare la funzione Recorder::create() su quel Recorder, e fornire un nome di file dove registrare. Eventuali errori nella creazione e nella scrittura del file verranno restituiti come codici di stato dalla funzione create().

In terzo luogo, è necessario fornire i flussi di dati da registrare. Questo viene fatto usando Recorder::attach() per collegare il Recorder a un dato VideoStream. Se si desidera registrare più di un flusso, è sufficiente chiamare attach() più volte, una per ogni VideoStream da aggiungere.

Registrazione

Dopo aver collegato lo stream al file, la registrazione non avrà inizio fino a quando non viene chiamata la funzione Recorder::start(). Una volta chiamata la funzione start() ogni frame generato dallo stream collegato viene scritto nelfilo ONI. Una volta terminata la registrazione, bisogna chiamare la funzione Recorder::stop(). Chiamando la funzione Recorder::destroy() si libera la memoria utilizzata dal Recorder e si garantisce che il file venga scritto sul disco in modo corretto.

Riproduzione

I file ONI possono essere riprodotti da molte applicazioni che utilizzano OpenNI. Per connettersi a loro direttamente dal codice dell'applicazione, bisogna aprire un file device e leggere da esso (vedi la sezione Device). Controlli di riproduzione aggiuntivi sono accessibili mediante l'oggetto PlaybackControl dal vostro file device (vedere la sezione PlaybackControl).

3.4.8 Classi di supporto

Oltre alle principali classi di OpenNI, sono presenti un certo numero di classi di supporto che servono principalmente per incapsulare dati. Esse sono descritte brevemente qui.

DeviceInfo

Questa classe registra le impostazioni di configurazione del device, tra cui il nome del dispositivo, URI, USB VID-PID, descrittori e nome del fornitore.

SensorInfo

Questa classe memorizza le impostazioni di configurazione che si applicano a un dato sensore. Un "sensore" in questo contesto è o una telecamera a infrarossi, telecamera RGB o telecamera di profondità. Un dispositivo può contenere diversi sensori.

VideoMode

Questa classe memorizza la risoluzione, framerate e il formato dei pixel di un fotogramma. É utilizzata da VideoStream per impostare e monitorare le impostazioni, da VideoFrameRef per monitorare queste impostazioni, e da SensorInfo di fornire un elenco di tutti i modi validi.

CameraSettings

Memorizza le impostazioni di una telecamera RGB. Consente di abilitare o disabilitare il bilanciamento automatico del bianco e l'esposizione automatica.

Version

Memorizza la versione del software. Utilizzata da OpenNI per riferire la sua versione. Può essere utilizzata anche da tutte le applicazioni che desiderano adottare lo stesso schema di controllo delle versioni o per specificare le versioni OpenNI richiesta.

RGB888Pixel

Tale struttura memorizza un singolo valore di pixel colorato.

Array

OpenNI fornisce una semplice classe Array che avvolge le matrici primitive che contengono dati di immagine.

Coordinate Conversion

Una classe di conversione di coordinate viene fornita per consentire la conversione tra coordinate reali e coordinate di profondità.

3.5 NITE 2.0

[4] Diamo qualche approfondimento sulle funzionalità della libreria NITE 2.0. Come già accennato sopra NITE è un toolbox che opera sulla libreria OpenNI. Sebbene OpenNI sia in grado di monitorare i movimenti della mano, NITE mette a disposizione degli algoritmi per facilitare i programmatori non solo ad individuare un punto che identifichi il palmo della mano, ma anche ad elaborare i dati provenienti dal sensore in maniera tale da trarre delle informazioni più accurate sugli spostamenti effettuati dalla mano.

La libreria è caratterizzata da particolari oggetti, controlli, che elaborano ed analizzano dati inerenti alla mano e al suo spostamento al fine di individuare specifici movimenti.

I controlli supportati da NITE sono i seguenti:

Push Detector - Questo controllo gestisce l'evento Push che consiste nel muovere la mano verso il sensore e tirarla indietro. Il gesto Push può essere usato, per esempio, per selezionare un oggetto o aprire una cartella.

Swipe Detector - Rileva il gesto Swipe sia verso l'alto, il basso, a sinistra o a destra. Lo Swipe è un movimento breve in una specifica direzione dopo il quale la mano si ferma. Ad esempio, tale gesto potrebbe essere usato per sfogliare le pagine di un libro.

Steady Detector - Il controllo cerca di riconoscere quando la mano è ferma per un determinato lasso di tempo. Il gesto Steady avviene appunto quando la mano è completamente ferma o quasi e la sua varianza delta è circa zero. Questo gesto è utile per gli altri controlli, per far sì che il successivo evento parta da mano ferma.

Wave Detector - Identifica il movimento ondulatorio della mano. Il gesto Wave consiste in un un certo numero di cambiamenti di direzione della mano entro un certo timeout. Di solito sono necessari quattro cambiamenti di direzione per rilevare un Wave.

Circle Detector - Questo controllo rileva movimenti circolari della mano. Affinchè il gesto venga riconosciuto è necessario che la mano compia un giro completo sia in una direzione che nell'altra. La direzione positiva è considerata quella in senso orario, mentre quella in senso antiorario è considerata negativa.

SelectableSlider1D - Il controllo riconosce uno scorrimento della mano in una delle tre direzioni degli assi X,Y, Z, cioè sinistra-destra, sopra-sotto, vicino-lontano. Lo scorrimento è diviso in parti uguali in un certo numero di aree e ogni area definisce un singolo oggetto. Pu essere utilizzato per creare menu in cui ogni oggetto corrisponde ad una opzione del menu.

SelectableSlider2D - Il controllo rileva uno scorrimento della mano in due direzioni sul piano X-Y.

Tutti questi oggetti estendono dall'oggetto Point Control il quale riceve gli hand point attivi ad ogni frame, tenendo in questo modo traccia di tutti i punti (rappresentati in coordinate rapportate allo schermo) dello spostamento della mano. Ogni sottoclasse poi analizza questi punti notificando qualora si verificasse quel determinato spostamento per cui la classe è specializzata.

Anche l'oggetto Point Control può scatenare delle notifiche in particolare con la creazione di nuovi punti, con il movimento dei punti e la scomparsa dei punti.

Le notifiche vengono gestite con un meccanismo ad eventi.

Classe nite::UserTracker - Fornisce tutte le funzionalità relative alla segmentazione della scena, il monitoraggio scheletro, rilevamento della posa e il tracciamento degli utenti.

Classe nite::HandTracker - Fornisce tutte le funzionalità relative al monitoraggio della mano, incluso il rilevamento dei gesti.

Capitolo 4

Da OpenNI 1.5 a OpenNI 2.0

Questo capitolo vuole spiegare il salto che si è avuto dalle vecchie versioni di OpenNI, con particolare attenzione dalla versione OpenNI 1.5.2, alla nuova versione del framework la 2.0. Questa sezione è stata sviluppata sia come aiuto al porting delle applicazioni che utilizzano le vecchie API, oltre a capire i nuovi concetti introdotti con la nuova versione di OpenNI.

4.1 Overview

OpenNI 2.0 rappresenta un importante cambiamento nella filosofia di fondo di OpenNI. Un' attenta analisi delle precedenti API, nella versione 1.5, ha rilevato che molte caratteristiche sono raramente o mai utilizzate dagli sviluppatori. Nel progettare la nuova versione 2.0, PrimeSense ha cercato di ridurre notevolmente la complessità delle API. Alcune caratteristiche sono state ridisegnate attorno alle funzionalità di base di maggior interesse per gli sviluppatori. Di particolare importanza è che OpenNI 2.0 semplifica enormemente l'interfaccia per comunicare con i sensori di profondità.

L'architettura plug-in del vecchio OpenNI è stata rimossa completamente. OpenNI 2.0 ora fornisce delle API per comunicare con i sensori tramite i driver sottostanti. Anche i middleware come NITE per l'interpretazione delle informazioni ora sono

disponibili come pacchetti standalone in cima a OpenNI.

Il seguente diagramma di alto livello confronta l'architettura di OpenNI - NITE 2.0 con l'architettura di OpenNI - NITE 1.5.

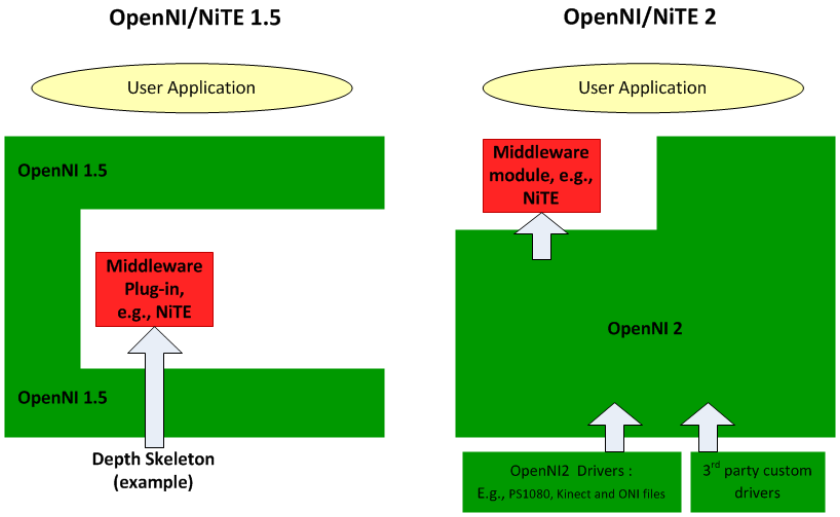

Figura 4.1: Confronto di architettura OpenNI - NITE 2 con OpenNI - NiTE1.5.

Semplificando le interfaccedi base che gli sviluppatori utilizzano effettivamente PrimeSense spera che il lavoro dei fornitori di middleware di terze parti risultino facilitati a implementare i loro prodotti basandosi su OpenNI.

Semplificazione dei dati

OpenNI 1.5 aveva un'ampia varietà di tipi di dati complessi. Ad esempio, le mappe di profondità erano avvolte nei metadati. Questo ha reso più complicato lavorare con molti tipi di dati che, nella loro forma grezza, sono semplicemente degli array. OpenNI 2 perviene a questo con le seguenti soluzioni:

- Unifica le rappresentazione dei dei dati IR, RGB e di profondità,

- Fornisce l'accesso algli aray sottostanti,
- Elimina i metadati non utilizzati o non pertinenti,
- Prevede la possibilità di non dover aver bisogno di un doppio buffer per i dati in arrivo.

A causa del modo con cui venivano gestiti i frame in OpenNI 1.5, il vecchio SDK era costretto all' uso di un doppio buffer per tutti i dati proveniente dal sensore. Ciò ha creato problemi di attuazione, di complessità e di prestazioni. La semplificazione dei dati in OpenNI 2.0 prevede la possibilità di non dover aver bisogno di un boppio buffer per i dati in arrivo.

Passaggio da Data Centric a Device Centric

Il disegno complessivo delle API ora può essere descritto come device centric piuttosto che data centric. Alcuni concetti relativi ai dati sono stati eliminati a favore di un modello molto più semplice che fornisce un accesso semplice e diretto ai dispositivi sottostanti e hai dati che producono. Le funzionalità fornite da OpenNI 2.0 in generale sono le stesse di quelle fornite da OpenNI 1.5, a cambiare sono i metodi di accesso alle funzionalità di OpenNI.

Più facile da imparare e da capire

Si prevede che le nuove API saranno più facili da imparare e da utilizzare per i programmatori. OpenNI aveva quasi un centinaio di classi, oltre a più di cento strutture e enumerazioni di dati di supporto. In OpenNI 2.0, questo numero è stato ridotto acirca una dozzina, con un' altra dozzina di dati di supporto. Le funzionalità di base delle API possono essere rappresentate da 4 classi princpali (viste nel capitolo precedente):

- openni::OpenNI

- openni::Device
- openni::VideoStream
- openni::VideoFrameRef

Purtoppo, questa riprogettazione ha richiesto di rompere la compatibilità con OpenNI 1.5. La decisione di fare questo non è stata affatto presa con leggerezza, tuttavia, si è ritenuto necessario al fine di raggiungere gli obiettivi di progettazione delle nuove API.

Accesso ai dati di profondità tramite eventi

In OpenNI 1.5 e prima, i dati di profondità erano reperibili soltanto attraverso un ciclo che aspettava l'arrivo di un nuovo frame. Fino a quando un nuovo fotogramma non era pronto il thread si bloccava. OpenNI 2.0 ha ancora questa funzionalità, ma fornisce anche funzioni per l'accesso ai dati di profondità tramite gli eventi.

4.2 Classe OpenNI

Sostituisce la classe Context, openni::OpenNI fornisce le stesse funzionalità complessive della classe Context di OpenNI 1.5. Qualsiasi applicazione basata su OpenNI 2.0 deve partire eseguendo la funzione di inizializzazione di questa classe.

In OpenNI 1.5 per accedere ai dati di profondità bisognava passare attraverso nodi, generatori e alberi. In OpenNI 2.0 questo è stato semplificato con un modello nel quale i sensori hardware sono rappresentati da oggetti della classe Device i quali forniscono degli stream dati che a loro volta sono composti da FrameRefs sequenziali (singoli fotogrammi). La classe OpenNI è responsabile dell'inizializzazione dei driver hardware e rende i dispositivi fisici accessibili dagli oggetti Device.

Notare che la classe openni::OpenNI è stata implementata come un insieme di funzioni statiche e a differenza della classe Contex in OpenNI 1.5 essa non ha bisogno di essere istanziata. Basta chiamare la funzione initialize(); per rendere le API pronte all'uso.

Varie funzionalità

I vari tipi di errore implementati in OpenNI 1.5 tramite la classe EnumerationErrors sono stati sostituiti con semplici codici di ritorno. La traduzione di questi codici in stringhe leggibili viene gestita dalla classe openni::OpenNI. Questa sostituisce la classe Xn::EnumerationErrors di OpenNI 1.5.

OpenNI aggiunge i seguenti eventi legati ai dispositivi:

- evento generato dall'aggiunta di un nuovo Device,
- evento generato dalla rimozione di un Device,
- evento generato dalla riconfigurazione di un Device.

4.3 Classe Device

Il vecchio approccio ha avuto una certa simmetria nel modo in cui si accedeva ai dati, ma ignorava la realtà sottostante non facendo distinzione dei casi in cui i dati erano prodotti da un flusso video RGB, un flusso IR, o un flusso di profondità.

OpenNI 2 è passato a una metafora che imita molto più da vicino le situazioni di vita reale. Così facendo, sono riusciti a ridurre la complessità di interagire con l'hardware, i dati che produce, e il middleware che agisce su tale hardware.

Alla base della nuova gerarchia c'è la classe Device. Un dispositivo può essere rappresentato da:

- un dispositivo hardware fisico che produce flussi di dati effettivi,
- un file contenente una registrazione presa da un dispositivo fisico.

La classe Device viene utilizzata per connettersi a un dispositivo, configurarlo, e per ottenere flussi video (implementati come oggetti VideoStream). Per ogni dispositivo vi è un flusso per i frame a colore, IR, e di profondità.

La classe Device in OpenNI 2.0 sostituisce la classe Device in OpenNI 1.5, con la sostituzione delle classi Depth, IR e Image Generator.

4.4 Classe VideoStream

La classe VideoStream sostituisce DepthGenerator, IRGenerator, e ImageGenerator come fonte diretta dei dati dal sensore. Esso fornisce i mezzi per avviare la generazione dei dati, leggere i singoli fotogrammi, ottenere informazioni circa gli stream e configurare i flussi. Essa incapsula anche le funzionalità precedentemente contenute nelle classi CroppingCapability, MirrorCapability, e alcune delle funzionalità della classe GeneralIntCapability.

Poichè le classi Generator sono state eliminate, i dati sono ottenuti direttamente dai dispositivi. Per creare un flusso video, è sufficiente creare un oggetto VideoStream, e chiamare la funzione openni::VideoStream::Create() la passandogli un oggetto Device valido che fornisca i dati.

4.5 Classe VideoFrameRef

In OpenNI 1.5, i dati venivano memorizzati in una complessa gerarchia di classi. C'erano classi separate per i frame IR, immagine e di profondità.

In OpenNI 2.0 questa complessità è stata ridotta utilizzando un unica classe VideoFrameRef per incapsulare tutti i dati relativi a un singolo fotogramma , indipendentemente dal tipo. I metadati sono stati ridotti al minimo indispensabile per lavorare con i frame.

4.6 Funzionalità per il riconoscimento e NITE 2.0

In OpenNI 1.5 era stata fornita una serie di interfacce per il rilevamento dei gesti, tracciamento dello scheletro, rilevamento della mano e per la rilevazione degli utenti le quali potevano essere sfruttate dai middleware plugin per implementare interfacce specifiche. NITE 1.0 è stato implementato a questo scopo.

In OpenNI 2.0 questo approccio è stato completamente cambiato. Ogni tentativo di standardizzare l'interfaccia tra il middleware e le applicazioni è stato abbandonato.

Tuttavia NITE risulta ancora previsto con le stesse funzionalità. Ora però è un pacchetto autonomo con delle propie API. Così ora ci sono 2 programmi separati da installare, uno per OpenNI e uno per NITE.

Body Tracking

Tutte le funzionalità relative alla Scene Segmentation, Skeleton Tracking, Pose Detection e User Tracking ora sono gestite da una singola classe UserTracker. Ogni frame riguardante lo UserTracker viene memorizzato in un oggetto della classe UserTrackerFrameRef dalla quale si ottengono informazioni di segmentazione scena, di scheletro e di posa. I dati specifici di un determinato utente vengono memorizzati in un oggetto di tipo UserData. Questi oggetti sono ottenuti mediante funzioni get(); dalla classe UserTrackerFrameRef.

UserTracker incorpora anche la possibilitá dello Skeleton Tracking. Per avviare il monitoraggio di un determinato utente basta richiamare una funzione, nite::UserTracker::startSkeletonTracking().

Una volta che il monitoraggio è iniziato, i dati dello scheletro saranno disponibili come un oggetto nite::Skeleton, ottenuto da UserData richiamando la funzione nite::UserData::getSkeleton().

La classe Skeleton fornisce anche un semplice accesso ad un array di articolazioni. Posizione e orientamento di ogni articolazione vengono memorizzati nell'oggetto nite::SkeletonJoint, che possono essere ottenuti utilizzando la funzione nite::Skeleton::getJoint().

Hand Tracking

Anche il tracciamento della mano ora è eseguito tramite l'API HandTracker. Queta API include anche le chiamate per il rilevamento dei gesti.

L' HandTracker deve essere inizializzato con la funzione nite::HandTracker::create() e il monitoraggio può essere avviato tramite nite::HandTracker::startHandTracking().Tutti i punti della mano vengono incapsulati in un oggetto HandTrackerFrameRef. I gesti riconosciuti vengono memorizzati come tipo GestureData.

Capitolo 5

Caso Applicativo

Come caso applicativo mi sono concentrato sull'applicazione di Simone Costanzi (nel proseguo sarà chiamata applicazione base), sviluppata in occasione della sua tesi di laurea dal nome "REALIZZAZIONE DI UN PROTOTIPO DI SCHERMO PERVASIVO ADATTATIVO", questa applicazione è in grado di riconoscere una persona qualora questa gli porga attenzione e successivamente adattare i suoi contenuti interattivi (attraverso interfaccia gestuale) in base alla determinata persona.

La mia attenzione è ricaduta su questa applicazione in quano come sensore per il riconoscimento è stato utilizzato il Kinect descritto in precedenza e come framework si è utilizzato OpenNI 1.5.

Io mi sono concentrato sulla prima parte dell'applicazione, in particolare sul componente che si occupa del riconoscimento degli utenti individuati dal sensore Kinect e sulla base delle informazioni fornite da questo strato si è in grado di definire quando una persona voglia interagire con lo schermo "aggiornando" l'applicazione.

Il mio compito è quello di evolvere questo componente, in particolare cercando di riconoscere più di una sola persona che vuole interagire con il sensore in quanto l'applicazione base era ristretta all'interazione di una sola persona alla volta e soprattutto portare l'applicazione dalla vecchia versione 1.5 di OpenNI alla nuova 2.0 verificando se

effettivamente si riscontrano miglioramenti generali nell'utilizzo della nuova versione del framework.

Nell'applicazione base la classe che si occupa di questo ha il nome di GestureManager la quale crea e gestisce tutti gli strumenti che interpretano i dati provenienti dal sensore: il Depth Generator (per il recupero delle informazioni sulla profondità), l'UserGenerator (per identificare gli utenti nella scena) con la capacità di riconoscere lo scheletro (SkeletonCapability) in modo da avere la posizione delle parti del corpo che interessano. Per ogni frame vengono aggiornate le posizioni di tutti i giunti dello scheletro presi in considerazione (testa, collo, spalle, gomiti, mani, torso, anche, ginocchia, piedi; alcuni servono solamente per la visualizzazione dello scheletro). Tutte le posizioni dei giunti sono espresse (il centro del sistema di riferimento è il sensore con l'asse z uscente dal sensore) e misurate in millimetri.

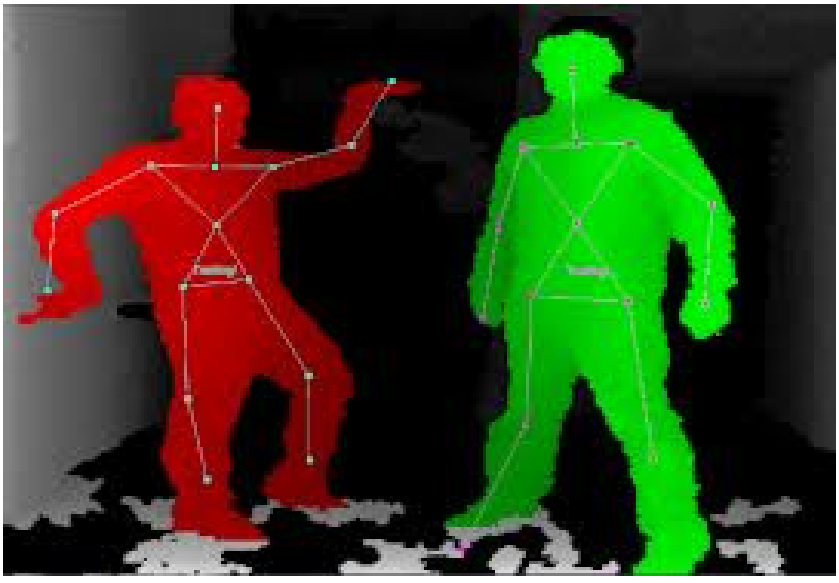

Figura 5.1: Riconoscimento degli utenti nell'applicazione .

5.1 Funzionamento Precedente

Prima di tutto bisogna creare e configurare un oggetto della classe GestureManager (che deriva da KinectManager) passandogli un oggetto di tipo Context, GestureManager app = new GestureManager(context, frame); e avviarlo app.run();.
La maggior parte del funzionamento di questo thread sta nel metodo run():
Con le funzioni ImageGenerator.create(context); DepthGenerator.create(context); UserGenerator.create(context); vengono creati i vari stream mentre con context.startGeneratingAll(); vengono avviati.
Con la funzione **context.waitAnyUpdateAll();** il thread si blocca finche non riceve un nuovo frame, da questo nuovo frame vengono estratte le informazioni riguardanti gli utenti presenti nella scena grazie alla classe UserGenerator, in particolare viene estratta una mappa dati la quale per ogni pixel viene assegnato un indice da 0 a n che indica se in quel determinato punto della mappa è presente una persona (gli indici con lo stesso valore maggiore di 0 indicano la sagoma di una persona, gli indici con valore 0 indicano che in quel determinato punto non è presente nessun utente) questa mappa dati poi viene salvata in un buffer,
SceneMetaData sceneMD = getUserGenerator().getUserPixels(0);
ShortBuffer scene = sceneMD.getData().createShortBuffer();,
dopodiche viene controllato ogni pixel e per quelli con valore maggiore di 0 viene salvato l'indice in una struttura dati (che elimina i duplicati):

```
while(scene.remaining() > 0){
short user = scene.get();
if (user != 0) set.add(""+user);}
```

Queste azioni vengono effettuate ogni volta che un nuovo frame viene registato dal sensore (quindi con una grossa quantità di codic eseguito), in aggiunta sono presente altre operazioni di controllano della struttura dati dove sono stati salvati i nuovi indici, per la precisione viene controllato se un utente gia registrato è uscito dalla scena, se un nuovo utente è stato riconosciuto o se è rimasto tutto come prima, avvisando di conseguenza i vari componenti e aggiornando lo stato dell'applicazione.

Poi con la funzione **getUserData(index);** si possono ottenere informazioni riguardanti la posizione dei vari giunti di un determinato utente identificato dall'indice, in modo da poter monitorare i suoi movimenti, riconoscere gesti e misurare alcuni parametri come la velocità e la direzione di alcune parti del corpo utili per il calcolo dell'attenzione dell'utente individuato.

L'attenzione dell'utente poi viene calcolata a parte attraverso lo studio della postura dell'utente. Considerando di trovarsi nelle vicinanze di un display, l' attenzione viene misurata utilizzando principalmente i seguenti parametri:

- velocità
- direzione
- direzione
- accelerazione
- distanza del soggetto
- posizione
- linearità del percorso del soggetto

Quindi un soggetto viene considerato attento al contenuto del display se la sua velocità, direzione, accelerazione e distanza diminuiscono, se è orientato con la testa e il corpo verso il display o se effettua un percorso lineare per avvicinarsi allo schermo.
Ulteriori discriminanti potrebbero essere:

- se il soggetto ha qualcosa in mano (ad esempio un cellulare o un giornale) che potrebbe interferire con l'attenzione da dedicare al display pubblico (diminuzione dell'attenzione).

- se il soggetto sta parlando con un'altra persona, che potrebbe distrarlo dal contenuto del display (diminuzione dell'attenzione).
- se è sbilanciato con il peso su una gamba e non guarda in direzione del display (diminuzione dell'attenzione).

La parte del progetto che si occupa di questo è costituita dalle seguenti clessi:

Kinect Manager: crea e gestisce tutti gli strumenti che interpretano i dati provenienti dal sensore: il Depth Generator (per il recupero delle informazioni sulla profondità), l'UserGenerator (per identificare gli utenti nella scena) con la capacità di riconoscere lo scheletro (SkeletonCapability) in modo da avere la posizione delle parti del corpo che interessano. Per ogni frame vengono aggiornate le posizioni di tutti i giunti dello scheletro presi in considerazione (testa, collo, spalle, gomiti, mani, torso, anche, ginocchia, piedi; alcuni servono solamente per la visualizzazione dello scheletro). Tutte le posizioni dei giunti sono espresse in coordinate mondo (il centro del sistema di riferimento è il sensore con l'asse z uscente dal sensore) e misurate in millimetri.

UserAnalizer: thread creato dal KinectManager al rilevamento di un nuovo utente in scena. A partire dai dati sulla posizione dello scheletro stima il livello di attenzione di un utente specifico, calcolando velocità, direzione delle spalle, parametri relativi alla postura e tempo di permanenza davanti allo schermo. Aggiorna infine l'interfaccia grafica.

5.2 Nuovo Funzionamento

Con la nuova versione di OpenNI le cose si semplificano enormemente migliorando la leggibilità, il funzionamento e l'efficenza dell'applicazione.

Per prima cosa inizializziamo OpenNI e Nite (non dobbiamo creare nuovi oggetto in quanto sono classi statiche)
OpenNI.initialize();
NiTE.initialize();,
colleghiamo un device fisico, in questo caso il Kinect, ad un oggetto di tipo Device **Device device = Device.open("URI");** controllando anche se il collegamento sia avvenuto senza problemi, in caso contrario visualizziamo un messaggio di errore, infine creiamo un oggetto di tipo UserTracker che ci fornirà tutte le funzionalità per la segmentazione della scena, il monitoraggio dello scheletro e il tracciamento degli utenti, **UserTracker tracker = UserTracker.create();**.

Una volta fatti questi passaggi di inizializzazione aggiungiamo un ascoltatore di eventi all'oggetto tracker **tracker.addNewFrameListener(this);** il quale ci avviserà quando un nuovo frame sarà disponibile richiamando automaticamente la funzione **public synchronized void onNewFrame(UserTracker tracker)** e passandogli se stesso come parametro.

La funzione onNewFrame() non farà altro che leggere il nuovo frame e controllare se sono stati riconosciuti nuovi utenti, in questo caso verrà avviato il tracciamento dello scheletro del nuovo utente in modo da monitorare i suoi movimenti:

```
for (UserData user : tracker.getUsers()) {
        if (user.isNew()) {
        // start skeleton tracking
        tracker.startSkeletonTracking(user.getId());}}
```

Se poi ci interessa ottenere informazioni rigurdanti i vari giunti di un utenti, come la posizione e la velocità, per la misurazione dell'attenzione dell'utente o per disegnare la sagoma basterà richiamare la funzione
JointType joint = user.getSkeleton().getJoint(JointType.HEAD) la quale ci restituirà un oggetto di tipo JointType con le informazioni riguardanti il giunto del

corpo specifico per il quale si è effettuata la richiesta nella funzione, in questo esempio quello della testa.

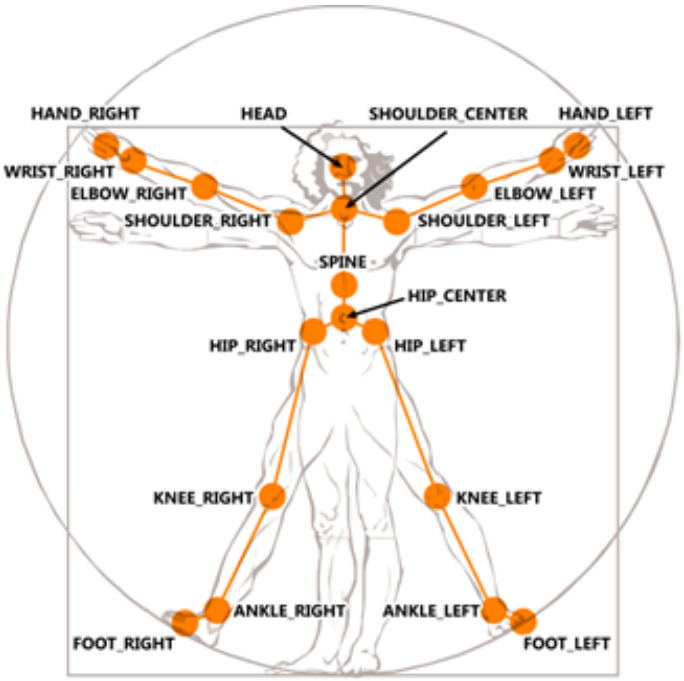

Figura 5.2: Giunti del corpo (Joint) riconosciuti dal sensore.

A questo punto abbiamo tutte le informazioni riguardanti gli utenti rilevati dal sensore per migliorare la nostra applicazione con funzionalità aggiuntive, nel mio caso per rendere l'applicazione più intuitiva ho riprodotto su schermo la sagoma e lo scheletro degli utenti rilevati colorandoli ognuno di un colore diverso e identificandoli con un ID univoco, il colore risulta più brillante per gli utenti più vicini al sensore e più scuri per quelli più lontani.

In più ho introdotto uno slider con il quale si può impostare il range (in millimetri con range massimo da 0 a 5000 mm) dentro al quale un utente se rilevato gli viene tracciato lo scheletro e visualizzato sullo schermo altrimenti se un utente viene rilevato al di fuori di questo range l'applicazione si accorge di questo utente ma non viene

tracciato (non vengono rilevate le posizioni dei vari giunti del corpo), questa funzionalità può essere utile per quelle NUI application che vogliono interagire con utenti non troppo lontani dal sensore cercando di misurare l'attenzione di questi utenti.

Infatti la seconda parte dell'applicazione consiste nella misurazione dell' attenzione degli utenti rilevati nella scena, gli utenti possono trovarsi in 3 stati, di "attenzione" (identificato dal colore rosso), di "possibile attenzione" (colore blu) e di "disattenzione" (colore nero) e a seconda dello stato in cui l'utente si trova l'indice che lo identifica cambia di colore.

Per trovarsi nello stato di "disattenzione" l'utente si deve trovare fuori dal range impostato dallo slider, per passare allo stato di "possibile attenzione" basta che l'utente entra nel range impostato, infine per passare allo stato di "attenzione" basta che l'utente alzi una delle due mani.

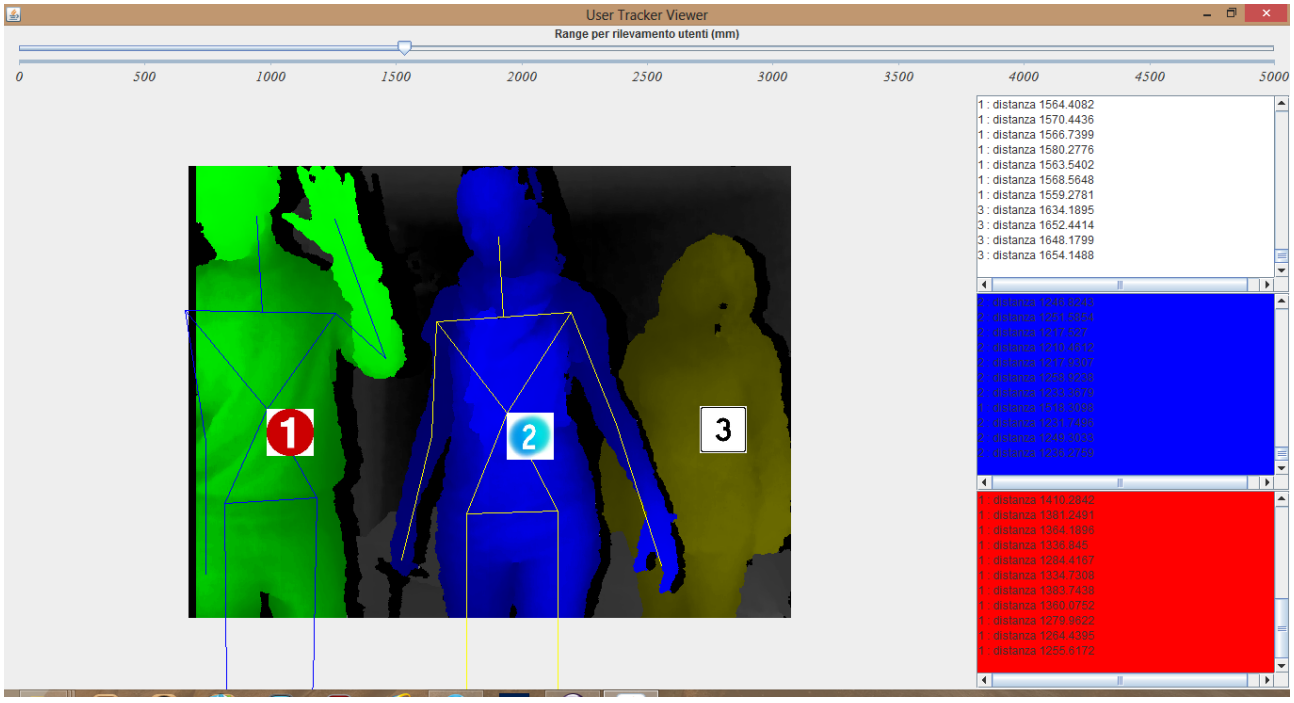

Figura 5.3: Screenshot dell'esecuzione dell'applicazione.

5.3 Descrizione dell'applicazione lato utente

L'utente intenzinato ad interagire con il sistema basta che si posizioni di fronte al sensore Kinect, questo consente di cominciare l'interazione con il sistema il quale accorgendosi della presenza di un nuovo utente aggiorna la sua interfaccia monstrando la sagoma della nuova persona riconosciuta e identificata da un indice ben visibile, la frequenza con cui aggiorna l'interfaccia è di 30 Hz cioè 30 fotogrammi al secondo una frequenza abbastanza elevata che permette di avere una buona senzazione di fluidità.

Dopo essersi posizionato di fronte al sensore, l'utente può trovarsi in 3 diversi stati di attenzione:

Disattenzione

L'utente è in questo stato quando inizialmente entrando nella scena ripresa dal sensore si trova ancora ad una distanza troppo lontana per poter interagire con il sistema oppure una volta terminato l'interazione con il sistema si allontana da questo, la distanza di confine è possibile impostarla grazie ad una slidebar la quale inizialmente è impostata a 5000 mm (5 metri).

Figura 5.4: SlideBar per impostare la distanza di confine.

In questo stato l'utente viene ripreso dal sensore ma viene mostrato a video solo la sua sagoma e non la posizione di tutti i suoi giunti in quanto non è intenzionato ad interagire con il sistema e quindi non ci preoccupiamo di monitorarne la posizione esatta.
L'identificazione avviene attraverso un indice di colore nero su sfondo bianco.

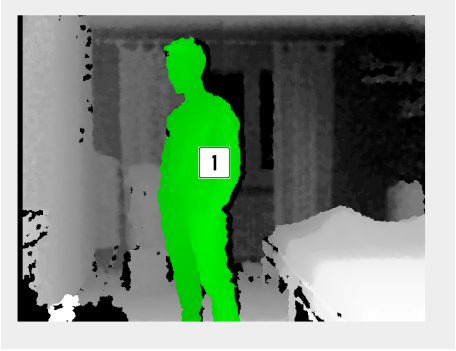

Figura 5.5: Stato di "Disattenzione".

Possibile Attenzione

L'utente incuriosito o intenzionato a interagire con il sistema che passa la distanza di confine impostata dalla slideBar passa nello stato di "Possibile Attenzione. In questo stato non è detto che l'utente voglia certamente interagire con il sistema infatti questo è uno stato intermedio dove l'utente può trovarsi anche per un semplice fatto di curiosità. Nonostante questo chi si trova in questo stato è identificato da un numero bianco dallo sfondo azzurro e oltre alla sua sagoma viene mostrata anche la struttura del suo scheletro in modo da poter determinare esattamente la posizione dei suoi giunti del corpo una volta che voglia interagire con il sistema.

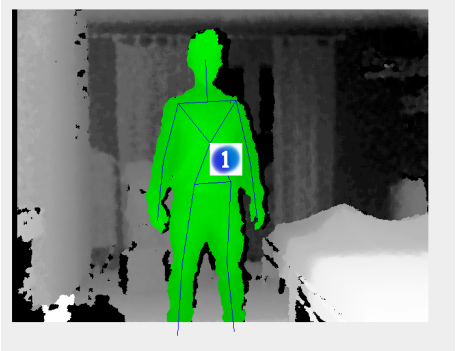

Figura 5.6: Stato di "Possibile Attenzione".

Attenzione

L'utente che si trova nello stato di "Possibile Attenzione" per passare allo stato di "Attenzione" per poter interagire con il sistema basta che alzi una delle due mani. A questo punto il sistema si accorge che uno o più utenti vogliono interagire con lui. Anche in questo stato viene mostrata la sagoma dell'utente con la relativa struttura del suo scheletro e l'identificazione avviene con un numero di colore bianco su sfondo rosso.

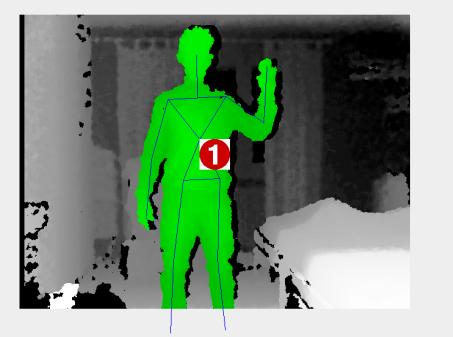

Figura 5.7: Stato di "Attenzione".

Oltra alle sagome degli utenti ripresi dal sensore l'interfaccia è costituita anche di 3 pannelli uno per ogni stato di attenzione che visualizza l' ID degli utenti presenti in quello stato con informazioni come la distanza dal sensore alla loro posizione.

5.4 Descrizione dell'architettura dell'applicazione

L'applicazione, come mostra la figura 5.8 è composta da 4 classi:

mainClass

Questa classe si occupa di inizializzare le classi statiche di OpenNI e NITE, di creare e inizializzare gli oggetti applicationViewer, sliderPanel e textArea oltre a controllare la

presenza di sensori Kinect collegati correttamente al sistema.

applicationViewer

Questa è la classe principale dell'applicazione che svolge il lavoro principale ed è quella che viene avvisata della presenza di un nuovo utente nella scena. Questa classe si occupa anche di disegnare la sagoma e lo scheletro degli utenti presenti nella scena oltra alla misurazione del stata di attenzione degli utenti rilevati.

sliderPanel

Questa classe si occupa della gestione del componente sliderBar per il settaggio della distanza di confine per la rilevazione degli utenti.

textArea

Infine questa classe si occupa della creazione di 3 JTextArea uno per ogni stato di attenzione e alla gestione delle informazioni da scrivere all'interno delle 3 aree di testo.

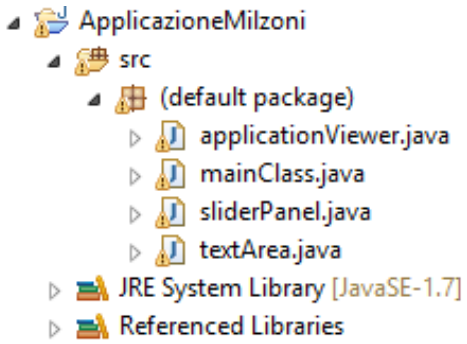

Figura 5.8: Architettura dell'applicazione.

5.5 Confronto

Come si può facilmente intuire con la nuova versione lo sviluppo dell'applicazione risulta più intuitivo e facile da capire per gli sviluppatori i quali non si dovranno più occupare di aspetti superficiali come accadeva nell'applicazione base dove una grossa parte di tempo veniva impiegata per la configurazione e del sensore e all'inizializzazione delle varie classi del framework, ma potranno concentrarsi completamente agli aspetti fondamentali della loro applicazione da sviluppare.

Un altro aspetto fondamentale sono le performance che nel caso della nuova versione aumentano notevolmente in quanto il lavoro di ricerca di un nuovo utente all'interno di un nuovo frame che prima veniva eseguire manualmente ora viene eseguita direttamente dal framework di OpenNI 2.0 e Nite 2.0 riducendo al minimo le operazioni da eseguire per effettuare questa ricerca e aumentando di conseguenza la velocità di eseguzione, in più grazie alla gestione degli eventi non è più necessario creare un thread apposito che si occupa di questa ricerca il quale era bloccato fino all'arrivo di un nuovo frame.

A questo punto possiamo concludere affermando che OpenNI 2.0 ha migliorato di molto il propio framework grazie all'introduzione di nuove funzionalità molto utili agli sviluppatori e sopratutto "ripulendolo" da aspetti superficiali e inutili presenti in precedenza che distraevano dallo scopo principale per cui è stato sviluppato.

Conclusioni e sviluppi futuri

Il successo di Kinect non è di certo solo dovuto alla nuova giocabilità introdotta dall'hardware e dai titoli usciti per Xbox, ma soprattutto per la sua possibilità di essere collegato a un PC e poter sviluppare applicazioni basate sulle Natural User Interface. Grazie anche alle nuove piattaforme di sviluppo supportate rendono più facile lo sviluppo di questa nuova tecnologia.
In questo lavoro abbiamo valutato OpenNI che grazie alle sue librerie e al Kinect risulta essere sufficiente e molto semplice da usare per lo sviluppo delle NUI applications.

Tuttavia questo è soltanto l'inizio di questa nuova fase di sviluppo tecnologico. Infatti Asus ha confermato di aver stretto un accordo con PrimeSense, l'azienda che ha collaborato con Microsoft per la progettazione e la realizzazione di Kinect, affinchè venga creato un dispositivo simile ma sui PC. Le poche notizie che si hanno a proposito affermano che la nuova periferica si chiamerà WAVI Xtion e potrà essere connessa al PC attraverso un trasmettitore wireless, per permettere all'utente di controllare con i gesti la musica, i film in riproduzione, i giochi e tante altre applicazioni, compresa forse anche la navigazione in Internet. Asus pensa di sviluppare delle applicazioni proprietarie che funzionino solamente con il dispositivo, creando cosi un market apposito: se così fosse, si potrebbe passare direttamente dal mouse alla " mano in aria", per cliccare ad esempio sui link ipertestuali o per scorrere le pagine di un documento, saltando direttamente la generazione dei touch screen sui PC casalinghi.

Anche Microsoft sta già guardando avanti ad una già possibile evoluzione di questa nuova tecnologia basata sull' EMG o meglio nota come elettromiografia, cioè la

registrazione dell'attività elettrica dei muscoli. Il progetto Wearable Electromyography - Based (EMG) Controller (controller indossabile basato sull'elettromiografia), è già sotto brevetto, come si può notare dalla pagina WIPO, ed ha come obiettivo la produzione di un dispositivo con il quale in futuro l'uomo possa interagire con i sistemi informatici, attraverso i segnali elettrici generati dai differenti muscoli del corpo. Una nuova interfaccia, quindi, che potrebbe nel giro di qualche decennio superare le tastiere, i mouse, i touchScreen ed i controlli vocali. Ma come funziona esattamente questa nuova interfaccia?

L'elettromiografia, in ambito medico, serve per registrare i segnali elettrici emessi dai muscoli durante il movimento e può essere eseguita o in modo invasivo, utilizzando degli aghi inseriti nelle fibre muscolari, oppure in modo non invasivo, servendosi di appositi elettrodi posizionati sulla superficie di particolari punti del corpo. Il primo metodo è sfortunatamente il più efficace, in quanto gli elettrodi esterni misurano variazioni di segnali elettrici non direttamente correlabili all'attività del muscolo, poichè quello stesso segnale deve passare attraverso il grasso corporeo e attraverso la pelle prima di giungere al sensore. Si potrebbe dunque pensare che Microsoft stia andando contro a indicazioni etiche come quella di infilare degli aghi nei muscoli delle persone, per permettere loro di comunicare con il pc. In realtà, BigM ricorre agli elettrodi di superficie, utilizzando però un piccolo trucco, che permette di superare i limiti di cui sopra: infatti, i ricercatori di Redmond ricorrono ad un numero di sensori molto più elevato di quelli normalmente necessari per misurare l'attività elettrica dei muscoli. In questo modo ottengono diversi segnali per uno stesso muscolo e successivamente, attraverso un processo automatico di localizzazione, si individua un sottogruppo di elettrodi dalla quale raccogliere i segnali più attendibili, per correlarli a particolari gesti o movimenti dell'utente. Così, si ottimizza l'affidabilità dell'apparecchiatura, migliorando la risposta dell'interfaccia agli input elettrici muscolari. Questi elettrodi possono essere poi anche integrati negli abiti oppure negli accessori di moda, come orologi od occhiali, che indossati permetterebbero di controllare ogni dispositivo elettronico compatibile.

Inoltre, l'interazione sarebbe a doppio senso: infatti, se tramite i segnali elettrici,

gli utenti possono controllare i propri gadget, allo stesso modo attraverso una stimolazione elettrica proveniente da questi stessi gadget sarebbe possibile, ad esempio, insegnare all'utente a suonare uno strumento oppure a compiere particolari gesti per uno sport.

Appendice A

Bill Gates e le NUI

Bill Gates ha tenuto una relazione sul potere di Natural User Interfaces per il suo compleanno presso l'Università di Washington lo scorso ottobre. Ha parlato delle potenzialità del Kinect e di questa nuova trasformazione tecnologica. Ecco un pezzo del suo intervento:

"Kinect is much more than just a cool video game technology - it is the most advanced example of a wave of new advances that are enabling people to interact with technology in entirely new ways.

One of the most important current trends in digital technology is the emergence of natural user interface, or NUI. We've had things like touch screen and voice recognition for a while now. But with Kinect, we are seeing the impact when people can interact with technology in the same ways that they interact with each other."[6]

Bibliografia

[1] www.wikipedia.org.

[2] Mascolo C., Capra L., Emmerich W., "Mobile Computing Middleware", Dept. of Computer Science, University College London, 2002.

[3] Documentazione OpenNI.

[4] Documentazione NITE.

[5] Simone Costanzi, "Realizzazione di un prototipo di schermo pervasivo adattativo"

[6] Bill Gates, relazione sulle NUI, ottobre 2012 all'università di Washington.

Ringraziamenti

Un sincero ringraziamento va a tutti coloro che mi hanno aiutato in vario modo a raggiungere questo importante traguardo.

Desidero ringraziare in primo luogo il prof. Mirko Viroli per la disponibilità e la cortesia con cui mi ha aiutato durante l'attività e la stesura di questa tesi.

Un sentito ringraziamento alla mia famiglia che con il suo sostegno morale ed economico mi ha permesso di raggiungere questo importante obiettivo.

Grazie a tutti gli amici con cui ho condiviso questi piacevoli anni di studio.

Sinceri ringraziamenti a tutti i miei amici, per il sostegno dato e la fiducia mai mancata.