Università degli studi di Cagliari

Facoltà di Scienze

Corso di Laurea Magistrale in Informatica

Modelling Feedback and Feedforward for Gesture Interfaces

Supervisor
Prof. Lucio Davide Spano

Candidate
Alessandro Carcangiu 49191

Anno Accademico 2014/2015

Title: Modelling Feedback and Feedforward for Gesture Interfaces

Supervisor: Prof. Lucio Davide Spano

Candidate: Alessandro Carcangiu (60/73/49191)

Abstract: The gesture interaction has spread hugely in the last decade, moving from a niche product to a daily use product. This is in part related to the technological evolution, which has allowed recognition systems to handle a large amounts of data in a relatively short time. However, most the gesture recognition systems lacks of reliable and efficient feedback and feedfoward mechanism, that assist the user during the interaction, for providing guidance to users and showing them the effects of their actions.

The goal of our thesis is to develop a model for the generation of continuous feedback and feedforward for systems and interfaces dedicated to the interaction through gesture or voice commands. this will help the user to explore and to learn all the gesture implemented, to understand what will be the status of the program following the execution of a certain gesture. Furthermore, users will be informed in real time on how the movements he's performing are recognized by the system.
To achieve this we must use a system that permitted to define the gesture such as a set of events. For this reason we used *GestIT*, a declarative framework for multiplatform gesture definition, which describe them as a temporal definition of smaller gestures.
This information is useful for generating feedback and feedfoward continuously, for showing the effects of the partial gesture detected on interface.

For demonstrating the soundness of the approach, we developed a proof of concept library that allows the developers to create and define gesture in a declarative manner, and from these gesture, generate feedback and feedforward according to our defined models.

A Gianlu

CONTENTS

Chapter

1 INTRODUCTION

Negli ultimi anni si è assistito a un crescente interesse verso i sistemi d'interazione, in ogni casa e ufficio troviamo ormai almeno un dispositivo touch. Così come stanno sempre più prendendo piede software e interfacce per il riconoscimento vocale. A questo si aggiungono anche i dispositivi per il tracciamento e il riconoscimento di gesture, metodo d'interazione che si sta diffondendo pian piano non solo nel mondo video-ludico (i prodotti *Virtuix Omni*[2] ne sono un esempio) ma anche negli uffici (con il *Leap Motion*[3]) e nelle case. Tale diffusione si può spiegare analizzando le caratteristiche delle gesture; Innanzitutto una gesture risulta essere più immediata da eseguire inoltre, a seconda del contesto, consente un maggior coinvolgimento e una miglior efficacia.

Questa massiccia diffusione dell'interazione diretta non è stata accompagnata però da un'altrettanta attenzione per lo sviluppo di architetture che consentano la generazione di feedback e feedforward adatti a tali sistemi. Infatti molti dei software che consentono questo tipo di interazione, in particolar modo per quelle basati sulle gesture, sono ancora sprovvisti di feedback e di feedforward efficaci. Questo è legato anche al fatto che le gesture, di per sé, non lasciano indietro informazioni rilevanti che possano essere veramente d'aiuto[1].
Una soluzione per risolvere questo problema può essere quello di utilizzare un modello dichiarativo delle gesture che consentirebbe di poter raccogliere i dati durante l'interazione. Successivamente questi dati potrebbero essere utilizzati per definire i sistemi di feedback e di feedforward.
Questi sistemi giocano un ruolo chiave nell'interazione. Difatti, in generale, la maggior parte degli utenti sono invogliati ad utilizzare un sistema solo se sono in grado di utilizzarlo in maniera semplice ed efficiente. A questo proposito basti pensare all'enorme espansione che ha avuto, negli ultimi vent'anni, il mercato dei computer e dei dispositivi elettronici. Le principali ragioni tecnologiche di questa diffusione sono legate non solo al miglioramento delle prestazioni e alla riduzione delle dimensioni, ma anche allo sviluppo di sistemi di feedback molto efficaci, che hanno aiutato gli utenti inesperti a prendere maggiore confidenza con le nuove tecnologie.
Ecco perché acquisisce una certa importanza lo sviluppo di architetture che possano

aiutare l'utente inesperto nell'imparare ad eseguire i movimenti richiesti, o nel conoscere in tempo reale quali saranno gli effetti sullo stato del programma della gesture che sta eseguendo. Infatti uno degli ostacoli principali di queste interfacce risulta essere, molto semplicemente, la difficoltà di apprendimento dei gesti da parte degli utenti finali a cui si accompagna anche il problema dell'ambiguità dell'input inviato dall'utente.

Queste sono le ragioni che ci hanno spinto a pensare e sviluppare delle architetture per generazione continua di feedback e feedforward; i nostri modelli di base sono altri sistemi sviluppati, per lo più, per interfacce a interazione indiretta ma che riteniamo ben adattabili al contesto di sistemi che consentano tracciamento dei movimenti dell'utente. A tal proposito per una rapida e semplice definizione delle gesture abbiamo deciso di utilizzare e implementare il framework *GestIT* adattandolo al nostro contesto; questo framework consente una definizione e un riconoscimento descrittivo delle gesture in maniera indipendente dal dispositivo di tracciamento utilizzato.
Per facilitare l'implementazione e la comunicazione tra questi due componenti abbiamo scelto di sviluppare una libreria, che consentirà agli utenti finali di per poter generare dei sistemi di feedback e di feedforward a partire dalle gesture che hanno definito attraverso gli strumenti messi a disposizione dalla libreria, minimizzando in questo modo il lavoro che dovrà fare il programmatore. Per semplificare ulteriormente il lavoro degli utenti finali, la libreria implementa delle componenti che supportano dei dispositivi di tracciamento quali *Kinect One* e *Leap Motion*. Queste componenti si occuperanno di stabilire la connessione con il dispositivo e di gestire i dati in arrivo, permettendo agli utilizzatori finali di concentrarsi maggiormente nell'utilizzo dei dati.

Nel capitolo dedicato allo stato dell'arte analizziamo nel dettaglio cosa si intende per riconoscimento delle gesture e dei comandi vocali, esaminando quali sono gli algoritmi, i dispositivi più famosi e le problematiche ad esse legate. Nello stesso capitolo andremo a studiare nel particolare cosa sono i feedback e i feedforward, quali sono i vantaggi che permettono di ottenere nell'ottica del riconoscimento delle gesture e, sempre sotto questo aspetto, quali sono i sistemi già sviluppati.
Nel capitolo 3 andremo ad analizzare, sotto ogni aspetto, l'implementazione e lo sviluppo di ciascuna componente della libreria, suddividendola in tre parti:

- la componente che si occupa di implementare il framework *GestIT*;
- quella che si occupa invece di generare i feedback e i feedforward;
- e infine la componente che implementa il supporto per *Kinect One* e *Leap Motion*.

Nel capitolo 4 esporremo la demo realizzata e i risultati ottenuti dai test, successivamente nel capitolo 5 esamineremo quali sono stati gli obiettivi raggiunti e quali potranno essere gli sviluppi futuri.

Chapter

2
STATE OF THE ART AND INSTRUMENTS

2.1 Human-Computer Interaction

L'interazione uomo-macchina (o *HCI*) è quella branca dell'informatica che si occupa di studiare e analizzare, sia da un punto di vista tecnologico che psicologico e fisico, l'interazione tra utenti e macchine. In questo caso per utenti intendiamo chiunque utilizzi un computer o uno strumento meccanico per eseguire un lavoro; mentre per macchina intendiamo una qualunque strumentazione che permette al suo utilizzatore di eseguire un certo lavoro.

Il principale obiettivo di questa branca è quello di ottimizzare e migliorare l'usabilità, la precisione e l'affidabilità di queste macchine e delle loro interfacce. L'interazione e l'interfaccia sono due concetti diversi: il primo rappresenta un modello astratto tramite il quale si interagisce con un dispositivo per compiere un determinato task, il secondo invece rappresenta una scelta tecnica di realizzazione di questo modello di interazione. L'interazione uomo-macchina è un campo di ricerca che interessa molte discipline:

Cognitive Science : è un settore di ricerca che coinvolge più discipline scientifiche; si occupa di studiare la mente e i suoi processi da più punti di vista: dall'apprendimento a basso livello e i meccanismi di decisione, ai circuiti neurali e alla logica ad alto livello.

Computer Vision : è la branca che si occupa di studiare nuovi metodi per l'acquisizione, l'elaborazione, l'analisi e la comprensione di immagini. La comprensione delle immagini si svolge con l'intento di trovare, di ottenere informazioni, esclusivamente attraverso i dati grezzi che descrivono l'immagine. Questi dati grezzi sono ottenuti mediante modelli costruiti a partire da nozioni di geometria, fisica e statistica. Un altro compito di cui si occupa la *Computer Vision* è quello di trovare nuovi modelli e di studiare la fattibilità di utilizzo nella vita reale di strutture dati molto grandi.

Design : è il settore che si occupa dell'organizzazione e della creazione di oggetti o sistemi. Nel contesto dell'interazione uomo-macchina il settore del *Design* è rappre-

sentato dall'industria del design, dalla grafica design, dal design dell'informazione e dell'interazione, e dal process-centered design.

Artificial Intelligence : è la branca della computer science che cerca di simulare il funzionamento e i processi mentali dell'uomo. Uno dei modelli più utilizzati è quello delle reti neurali, che come vedremo più avanti, sono delle reti formati da più livelli di nodi connessi tramite dei link pesati attraverso cui passa l'input.

Psycology : è il settore che studia ed analizza il comportamento della psiche umana. Nell'ambito della *Human-Computer Interaction* si occupa di studiare i meccanismi di memorizzazione e di funzionamento del sistema percettivo nell'uomo.

Il termine *HCI* è diventato di utilizzo generico già dagli anni '80. Pone le sue basi sulle ricerche effettuate negli anni '40, durante la Seconda Guerra Mondiale, allo solo scopo di produrre sistemi d'armi sempre più efficienti ed affidabili. Queste ricerche portarono a un'ondata di interesse nel campo dell'interazione uomo-macchina tra i ricercatori, tanto che nel 1949 venne fondata la Società di Ricerca in Ergonomia.
Dagli albori dell'informatica fino alla fine degli anni '70 i sistemi non erano dotati di interfacce ma solo di linee di comando, perciò risultavano difficili da utilizzare alla maggior parte degli utenti. Solo nei primi anni '80, con l'introduzione delle interfacce grafiche (basate sul *WIMP* e del concetto di manipolazione diretta mediante dispositivi di puntamento), si riuscì a sviluppare i primi sistemi user-friendly che permettevano all'utente di avere un feedback immediato delle azioni compiute.
Negli ultimi anni, con la diffusione dei dispositivi elettronici portatili e non, l'*HCI* ha acquisito un enorme importanza. Da questo consegue che oramai il successo di un prodotto commerciale dipende, in gran parte, dalla *user experience* offerta.

2.2 Gesture and Speech Recognition

L'interazione tra uomo e macchina può avvenire attraverso due tipi di dispositivi:

Dispositivi diretti : consistono sostanzialmente nell'elaborazione di dati inseriti direttamente dall'uomo, come nel caso dei comandi vocali o dei dispositivi touch. Questo tipo di dispositivi sono più veloci da utilizzare, non richiedono conoscenze pregresse dall'utente e soprattutto consentono un'interazione naturale con i sistemi. Commercializzati per la prima volta durante gli anni '80, non ebbero un grande successo anche per colpa dei limiti tecnologici dell'epoca; negli ultimi dieci anni però si è assistito ad un cambiamento: i gravi limiti tecnologici in termini di strumentazione e di potenza di calcolo sono stati superati, la maggior diffusione della tecnologia informatica nella vita quotidiana e le nuove tecniche adottate per la creazione di interfacce facili da usare hanno consentito l'enorme diffusione del linguaggio naturale nell'interazione con le macchine.

Figure 2.1: Esempio di interazione naturale

Dispositivi meccanici : o dispositivi indiretti, in cui l'utente invia dei comandi attraverso l'interazione con altri dispositivi, come nel caso del mouse o della tastiera. Questo tipo di dispositivi sono ancora molto diffusi e hanno il vantaggio di essere abbastanza precisi, ma richiedono una manutenzione costante e un certo addestramento per poter essere utilizzati efficacemente.

Un altro ambito di cui si occupa la *Human-Computer Interaction* è quello di studiare algoritmi e sistemi che permettano l'interpretazione e il riconoscimento del linguaggio naturale, che siano dei gesti o dei comandi vocali.

2.2.1 Gesture Recognition

La gesture recognition consiste nell'elaborazione e nell'interpretazione dei gesti e dei movimenti compiuti dall'utente durante l'interazione.

Applications

Attualmente il riconoscimento delle gesture viene utilizzato per scopi diversi e su una vasta gamma di gesti, che vanno dal linguaggio dei segni a quello delle espressioni facciali:

Sign language : costruire sistemi in grado di riconoscere[5], velocemente e con ridotti margini d'errore, la simbologia dei linguaggi dei segni consentirebbe anche a chi soffre di mutismo (o comunque di patologie che impediscano l'utilizzo della voce) di poter interagire in maniera naturale con i computer.

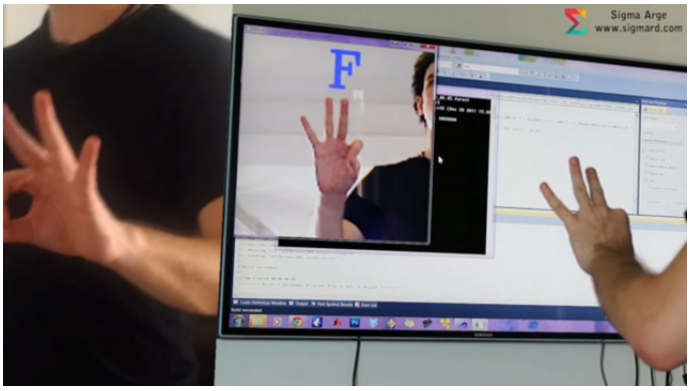

Figure 2.2: Un sistema dotato di camera può essere in grado di poter riconoscere facilmente il linguaggio dei segni.

Rehabilitation : analizzando e riconoscendo il movimento[7] di un arto compiuto dall'utente è possibile aiutare il paziente durante il suo processo di riabilitazione:

- il sistema indica all'utente quali sono i movimenti che deve eseguire;
- a partire dai movimenti eseguiti dall'utente, il software può aiutare il paziente a correggere gli eventuali movimenti scoordinati e

Ciò permetterebbe ai pazienti di poter fare riabilitazione direttamente da casa propria.

Figure 2.3: I dispositivi come la *Kinect*[6] possono dare il loro contributo anche in campo medico.

Face and eyes gesture : controllare dei sistemi tramite le espressioni facciali, o il movimento degli occhi, permetterebbe a chi non è fisicamente capace di usare mouse e tastiera di poter interagire con tali sistemi. Il tracciamento degli occhi, in particolare, può essere utile per controllare il movimento del cursore e per focalizzarsi sugli elementi presenti nell'interfaccia.

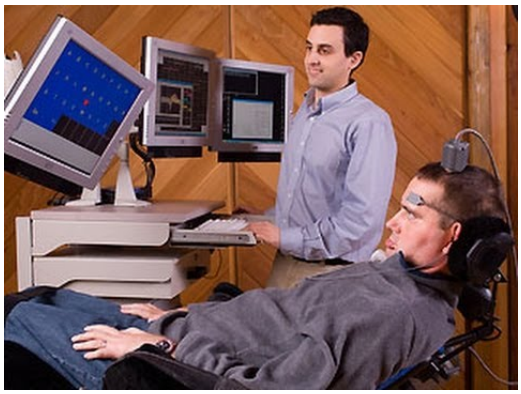

Figure 2.4: L'eyes-tracking permette alle persone affette da *ALS* (o comunque malattie neurodegenerative) di poter comunicare con l'ambiente.

Emotions : il riconoscimento dei gesti viene utilizzato anche nell'ambito della *emotional computation*[9]; sistemi simili vengono usati per addestrare i robot umanoidi, e in futuro potrebbero trovare utilizzo anche negli ospedali (nel reparto neonatale per esempio) o nelle strutture che richiedono uno stretto contatto con le persone.

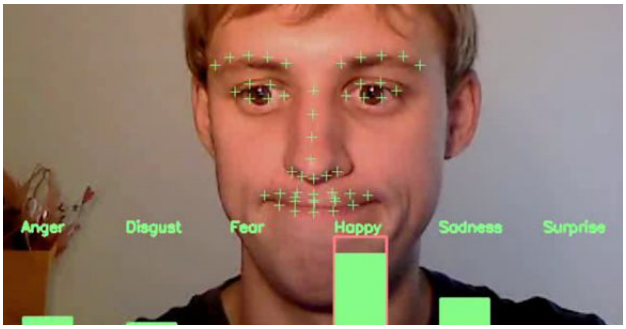

Figure 2.5: Il riconoscimento dell'emozione avviene analizzando l'espressione facciale dell'utente.

Alternative Interfaces : il riconoscimento delle gesture potrebbe permettere in futuro il passaggio dalle interfacce tradizionali, con mouse e tastiera, a nuovi tipi di interfacce più dirette e veloci da utilizzare; esistono già diversi controller virtuali atti al riconoscimento delle gesture, un esempio sono la *Kinect*[11] e il *Leap Motion* (di cui parlemo in maniera più estesa in seguito).

Figure 2.6: Esempio di interazione tramite*Leap Motion*

More immersive playing experience : Le gesture possono essere usate per interagire all'interno di videogiochi o sistemi simulativi, rendendo l'esperienza più realistica e coinvolgente. Negli ultimi anni sono stati presentati diversi progetti che combinano gli schermi per la realtà virtuale (come gli *Oculus Rift*[10]) con sistemi di gesture recognition, con il fine di migliorare l'esperienza dell'utente.

Figure 2.7: L'uso combinato di più device (come *Oculus Rift* + *Omni* potrebbe migliorare l'esperienza di gioco rendendola più simile alla realtà.

Hardware

Il rilevamento dei movimenti e il riconoscimento delle gesture viene eseguito attraverso diversi tipi di device:

Wired glove : consistono in guanti dotati di diversi sensori e magneti, che permettono di acquisire la posizione, il movimento e la rotazione delle mani e delle dita coperte dal device. Sono dotati anche di feedback aptici.

Figure 2.8: L'uso combinato di più device (come *Oculus Rift* + *Omnia*) potrebbe migliorare l'esperienza di gioco rendendola più simile alla realtà.

Infrared cameras : o thermographic cameras, sono dei dispositivi che creano delle immagini a partire dalla radiazione agli infrarossi; questo tipo di telecamere vengono utilizzate sia nella *Kinect* che nel *Leap Motion*, per identificare gli utenti e le gesture che compiono.

Figure 2.9: Le figure più chiare sono quelle più vicine al dispositivo, man mano che ci si allontana dal dispositivo gli oggetti rilevati assumono un colore più scuro.

Depth-aware cameras : sono degli strumenti che permettono di stimare in tempo reale la distanza tra la telecamera e gli oggetti della scena inquadrati; questo viene fatto misurando il tempo che necessità a un impulso luminoso di percorrere il tragitto telecamera-oggetto-telecamera (time of flight). Attraverso queste telecamere è possibile costruire una mappa della profondità della scena visibile dalla camera, dati che verrano poi usati per una ricostruzione 3d della stessa.

Figure 2.10: Le figure più chiare sono quelle più vicine al dispositivo, man mano che ci si allontana dal dispositivo gli oggetti rilevati assumono un colore più scuro.

Stereo cameras : è un tipo di approccio che consiste nell'usare due camere rivolte verso la stessa scena, e ad una distanza ben nota; dall'unione delle immagini registrate dalle due camere è possibile costruire una rappresentazione 3d.

Figure 2.11: Esempio di *stereo camera*.

Single cameras : per il riconoscimento di una gesture si può anche utilizzare una semplice camera in 2d; ovviamente è un device meno efficiente rispetto alle depth/stereo camera; tuttavia esistono diversi software che utilizzano una camera 2d standard 2d, in grado di riconoscere in maniera efficiente e robusta il movimento di mani e dita.

Figure 2.12: Le gesture e i movimenti possono essere rilevati anche con l'utilizzo di una semplice webcam.

Controller-based gesture : sono dei controller che rappresentano un estensione del corpo dell'utente in grado di percepire i movimenti compiuti da quest'ultimo; ai software associati al controller spetta il compito di tradurre questi movimenti in comandi/azioni.

Figure 2.13: Esempio di *Controller-based gesture*.

Nel corso degli ultimi dieci anni sono stati sviluppati e commercializzati diversi dispositivi che permettono la gestione e il riconoscimento delle gesture tramite movimenti della mano o del corpo. Alcuni permettono il rilevamento di gesture eseguite con tutto il corpo (come la *Kinect* o la *Playstation Eye*) altre invece solo di alcune parti del corpo (come il *Leap Motion*); Durante lo sviluppo della tesi, in particolare, abbiamo lavorato sia con la *Kinect 2.0*, sia con il *Leap Motion*:

Kinect One La *Kinect One* è la versione avanzata della *Kinect* uscita nel 2010 per la console *Xbox360*. Come il suo predecessore, la *Kinect 2.0* rappresenta un'interfaccia attraverso cui l'utente può interagire (in maniera naturale) con vari sistemi (*Xbox One* e sistemi *Windows 8*); è in grado di riconoscere e rilevare i movimenti degli utenti, è dotata di un sistema per il riconoscimento vocale, ed è provvista anche di microfono e videocamera con la quale si possono registrare dei filmati.

La *Kinect One*, tramite l'utilizzo del Windows SDK 2.0, ha incrementato notevolmente l'esperienza d'uso rispetto al suo predecessore, facendo raggiungere nuove frontiere all'interazione naturale del corpo con i computer. Sono stati migliorati gli aspetti legati alla precisione, ai tempi di risposta e sono state fornite funzionalità intuitive per decrementare i tempi di sviluppo di applicazioni che utilizzano movimenti, gesti e comandi vocali; andando nel dettaglio:

- riconosce in maniera più accurata e dettagliata i corpi, le varie gesture e i comandi vocali;
- ora è in grado di gestire fino a 6 utenti alla volta;
- e infine la miglior tecnologia, è in grado di gestire 2Gbps di dati, gli permette di rilevare il battito cardiaco di un utente, il suo umore, il suo peso e la sua altezza.

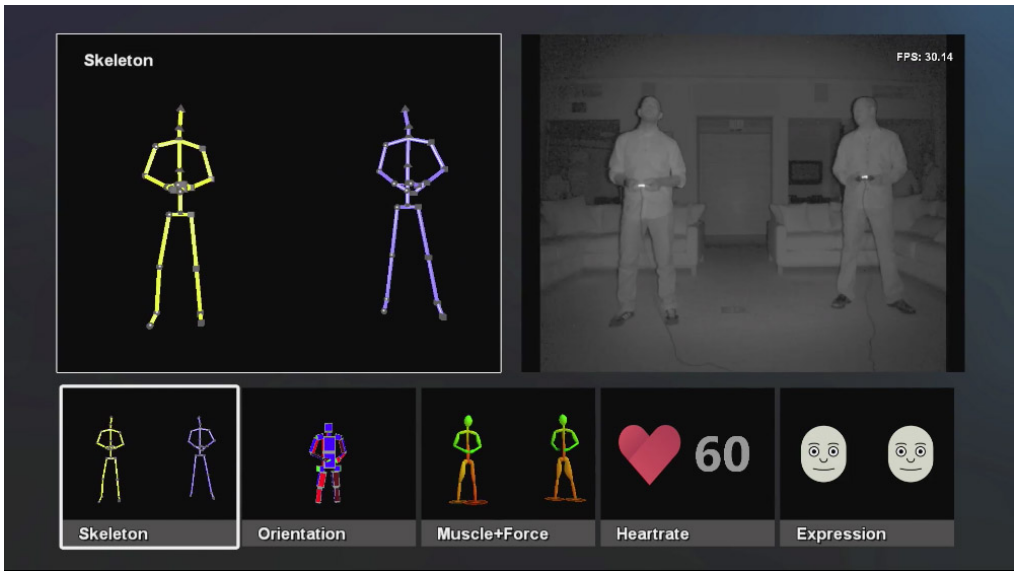

Figure 2.14: Una dimostrazione delle nuove funzionalità della *Kinect One*

La *Kinect* ha un campo visivo di 70° in orizzontale e 60° in verticale, ed è composta da diversi tools:

Color Stream : cattura le immagini video come le classiche webcam, ha una di risoluzione a 1920x1080x16bpp 30fps 16:9.

Depth Stream : permette, diminuendo il rumore di fondo, di identificare il corpo degli utenti (crea la mappa 3D dei corpi tramite l'analisi dei dati forniti dal sensore a infrarossi) e i movimenti di quest'ultimi. La profondità viene misurata in millimetri, e ha una risoluzione di 512x424x16bpp.

Infrared Stream : è la telecamera time of flight che viene usata per ottenere un tracciamento ottimale, anche al buio, della scena in cui viene utilizzata, il che rende la *Kinect* indipendente dalla luce. Ha una risoluzione di 512x424.

Microphone Array : L'array di microfoni permettono il riconoscimento degli input vocali e filtrano i rumori di fondo nella stanza.

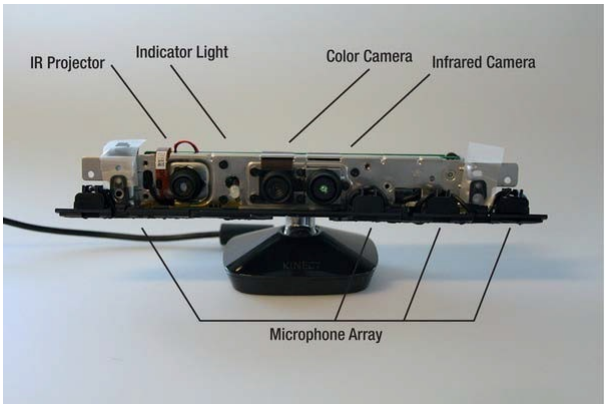

Figure 2.15: Le varie parti che compongono la *Kinect 2.0*

Objects and Gestures Recognition Tutte le informazioni associate agli oggetti rilevati dalla *Kinect* vengono inseriti all'interno di un *frame* e inviati poi al sistema. Le API[12] del dispositivo prevedono una classe Frame per ogni tipo di sorgente:

AudioBeamFrame: è la classe associata ai suoni e ai comandi percepiti dalla *Kinect* durante l'esecuzione. Gli *AudioBeamSubFrame* contengono al loro interno diverse informazioni riguardanti il suono percepito, ad esempio l'angolo da cui è giunto il suono e la sua confidenza, l'angolo del fascio di suono, la durata relativa del suono percepito e la lista di correlazione tra suono e corpi percepiti.

BodyFrame: questa classe viene utilizzata per visualizzare i dati dei corpi rilevati dalla *Kinect*. Dai *Body* è possibile accedere a diverse informazioni:

ClippedEdges: rappresentano gli *edges* che delimitano il campo visivo associato al corpo in questione.

HandState and HandConfidence: disponibile sia per le mano destra che sinistra, indicano rispettivamente lo stato (può essere *Closed*, *Lasso*, *NotTracked*, *Open* o *Unknown*) e la qualità del rilevamento (semplicemente *High* o *Low*).

Joints: rappresentano le diverse giunture rilevate dal device. La *Kinect* riconosce in tutto 25 *Joints*; la classe *Joint* permette a sua volta di accedere alle coordinate del *joint* associato.

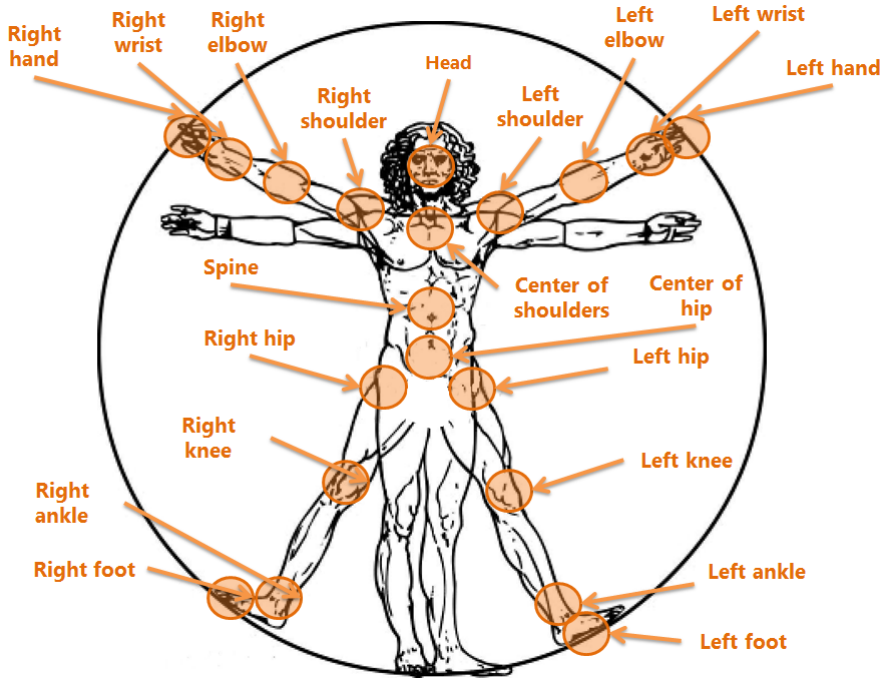

Figure 2.16: La lista di tutte le *joints* rilevabili dalla *Kinect*

JointOrientations: relativamente ad ogni *joint* rilevato restituisce il suo orientamento.

Lean: indica i movimenti eseguiti dal "profilo" dell'utente (spostamenti a destra e a sinistra corrispondono a movimenti lungo l'asse X, viceversa muoversi in avanti o in indietro corrisponde a movimenti lungo l'asse Y).

BodyIndexFrame: contiene al suo interno una mappa (all'infrarosso o di profondità), nella quale indica per ogni pixel se è una parte del corpo di un utente o meno.

ColorFrame: rappresenta le immagini (in 1080p) a colori della scena rilevata dalla *Kinect*. Ha dei metodi che permettono di copiare i dati della scena ripresa all'interno di un array di *byte* (o in un buffer).

DepthFrame: è la classe che contiene le mappe di profondità rilevate dalla *Kinect*, tramite l'apposita telecamera, che permette di copiare i dati che descrivono la scena ripresa all'interno di un array di *byte* o in un buffer.

InfraredFrame: in questa classe troviamo i dati sulla mappa all'infrarosso eseguita dalla *Kinect*; anche in questo caso la classe dispone dei metodi necessari per copiare i dati di ogni pixel all'interno di un buffer o di un array di *byte*.

LongExposureInfraredFrame: descrive la mappa all'infrarosso a lunga esposizione della scena ripresa dal dispositivo. A differenza dell'*Infrared*, si ottiene una qualità molto più alta dell'immagine (con un rumore di gran lunga ridotto), a scapito però di effetti di sfocatura qualora nella scena siano presenti oggetti in movimento.

FaceFrame/HighDefinitionFaceFrame: sono i frame inviati al sistema quando si attiva la face detection del dispositivo. Le due classi contengono i dati relativi ai volti rilevati dalla *Kinect One*, e permettono di accedere alla nuvola di punti relativa al volto e ai *BodyFrame*, *DepthFrame* e *ColorFrame* associati alla *FaceFrame* in gestione. L'*HighDefinitionFaceFrame* invece permette di ottenere un rilevamento dei volti più accurata e precisa, permettendo di poter accedere a informazioni aggiuntive come il colore dei capelli, riconoscere alcune espressioni o stati emozionali, rilevare la presenza degli occhiali. Queste informazioni vengono ottenute sfruttando la classe *FaceModel* e i diversi modelli installati nella libreria, che vengono presi come punto di riferimento durante l'esecuzione.

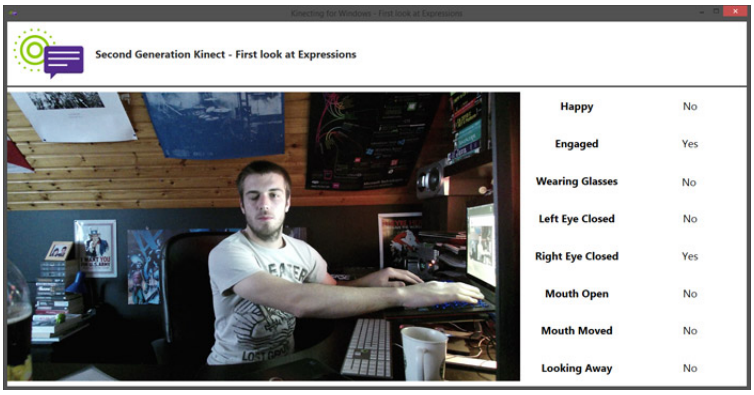

Figure 2.17: Un esempio dei dati rilevabili tramite l'HighDefinitionFaceFrame.

VisualGestureBuilderFrame: è il frame che contiene le informazioni sulle gesture rilevate dalla dispositivo; le gesture possono essere di due tipi: continue o discrete. Entrambe sono rappresentate da una classe:

ContinuousGestureResult: che rappresenta le gesture continue, e ha come attributi: *Progress*, che rappresenta il livello di progresso della gesture, e *Confidance* che invece rappresenta il valore di similarità rispetto alla definizione di una gesture presente nel database.

DiscreteGestureResult: rappresenta le gesture discrete; i suoi attributi sono: *Confidence* che rappresenta il valore di confidenza rispetto ad una gesture del database, e *FirstFrameDetected* che contiene il primo frame da quando è stata rilevata la gesture.

Per poter riconoscere le gesture, la *Kinect One* utilizza dei database in cui sono descritti le gesture che si vogliono riconoscere in quel momento. Le diverse gesture sono create a partire da dei video utilizzando l'applicazione *Visual Gesture Builder*, che si può installare assieme all'SDK.

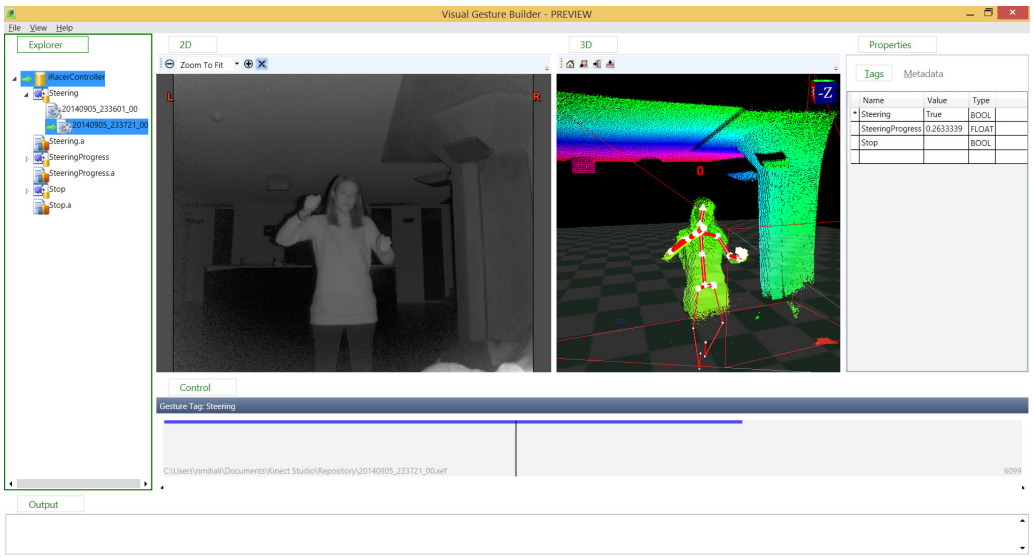

Figure 2.18: Il tool Visual Gesture Builder in funzionamento.

Apps Dal 2014, anno di rilascio in commercio della *Kinect*, sono state sviluppate molte applicazioni, ognuna delle quali si caratterizza per la possibilità che ha l'utente di interagire in maniera naturale col sistema. L'offerta in questo caso è molto ampia, si vanno dai classici videogame (in cui l'utente attraverso le gesture interagisce con gli oggetti del sistema), alle applicazioni per poter controllare dei computer anche con l'ausilio della *Kinect* e delle sue caratteristiche; un altro esempio sono i sistemi per la riabilitazione (*SFMA*, *Kinesology Taping*) e recupero degli infortuni [13]), oppure i programmi per poter condurre esperimenti in campo neurologico e fisiologico [14].

La *Kinect* può essere usata anche per registrare e raccogliere i dati sul motion capture [15] o, come ha recentemente dimostrato la *Nasa*, per controllare delle braccia meccaniche di un robot (tramite l'utilizzo combinato della *Kinect One* per comandare il robot e gli *Oculus Rift* per vedere l'ambiente che circonda il braccio meccanico[16]).

Figure 2.19: Un esempio di interazione naturale tramite *Kinect*

Leap Motion Il *Leap Motion* è un dispositivo, rilasciato nel Luglio del 2013, e come già detto in grado di rilevare e tracciare il movimento delle mani, delle dita dell'utente e di strumenti, come penne o bacchette, in maniera accurata. Obiettivo del *leap motion* è quello di aumentare le possibilità di interazione tra uomo e le macchine, soprattutto nel mondo 3D, nella quale l'utente trova sicuramente più naturale poter interagire direttamente attraverso l'uso delle mani, piuttosto che con mouse e tastiera. Il *leap* si

Figure 2.20: Il *Leap Motion*

presenta come un piccolo box in alluminio, dotato di due telecamere e di tre LEDs

all'infrarosso, che irradiano gli oggetti che rientrano nella scena rilevata dalle due telecamere; il campo di rilevamento del dispositivo è costituito da una piramide inversa, centrata sullo stesso e che si estende, dai 25 ai 600 mm di distanza dal dispositivo.

Figure 2.21: Una visuale di tutte le componenti del *leap motion*

Piano delle coordinate Il *leap* opera sulla parte destra del piano cartesiano, e il punto d'origine coincide con il centro del *leap motion*: l'asse X è orizzontale rispetto a quest'ultimo (quindi cresce da sinistra verso destra), l'asse Z invece assume valori positivi con l'allontanarsi dal dispositivo, e infine l'asse Y si sviluppa in maniera verticale, non può assumere valori negativi e, come nel caso dell'asse Z, cresce con l'aumentare della distanza dal dispositivo.

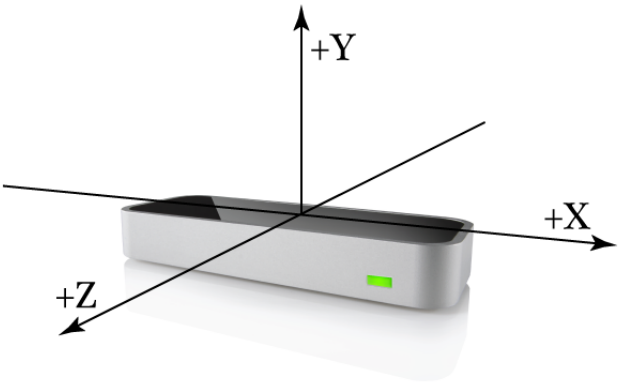

Figure 2.22: Il sistema di coordinate del *Leap*

Objects and Gestures Recognition Tutte le informazioni sugli oggetti, movimenti e gesture rilevate dal *leap motion*, vengono registrate su di un *Frame*. All'interno di ogni frame, ai vari oggetti e gesture tracciati, vengono assegnati degli ID univoci, che vi rimarranno associati per tutto il tempo che resteranno visibili al dispositivo, ciò significa che se l'oggetto viene perso e poi di nuovo tracciato, gli verrà assegnato un nuovo identificativo.

Nel dettaglio un frame contiene al suo interno:

Images - lista delle immagini all'infrarosso acquisite dal dispositivo;

Hands - da questa lista è possibile ottenere le informazioni sulle mani rilevate; l'SDK del *leap motion* permette di poter accedere a molte informazioni relative a una mano rilevata (tramite la classe apposita *Hand*), ovvero le sue coordinate, la normale, la velocità, i movimenti eseguiti, nonché braccio a cui è legata e le dita della mano stessa (se queste sono visibili). Il dispositivo è in grado di rilevare e gestire sia i casi in cui la mano risulta essere chiusa, sia quando sono visibili più di due mani, in quest'ultimo caso però a discapito di velocità e qualità dei dati raccolti. Gli attributi di una mano accessibili tramite il frame inviato dal *leap* sono:

- **palmPosition** - indica la distanza del centro della mano dal *leap motion*, espresso in mm;
- **palmVelocity** - rappresenta la velocità di movimento del palmo della mano, in mm/s;
- **palmNormal** - il valore della norma ricavato dalla posizione del palmo della mano rispetto al dispositivo;
- **direction** - la direzione del palmo della mano, calcolata a partire dalle sue dita, rispetto al dispositivo;
- **sphereCenter** - il centro della sfera ottenuta dalla curvatura della mano;
- **sphereRadius** - il raggio della sfera ottenuta dalla curvatura della mano;
- **Pointables** - restituisce la lista di tutte le dita e gli strumenti associati a quella mano;
- **Fingers** - restituisce solo le dita;
- **Tools** - restituisce solo gli strumenti.

Fingers/Pointables - queste due liste contengono, rispettivamente, tutte le dita e gli strumenti rilevati; Il *leap* distingue gli oggetti orientabili (o pointables) dalle dita; questa distinzione avviene in base alla forma: uno strumento viene riconosciuto come un oggetto più lungo, più sottile e più dritto di un dito. Con la versione 2.0 dell'SDK inoltre il dispositivo è in grado di rilevare anche le singole ossa che compongono ogni dito. Tutte le caratteristiche delle dita e degli strumenti, sono ereditate dalla classe *Pointable*, e sono:

- **length** - la lunghezza, in millimetri, della porzione visibile dell'oggetto in questione.
- **width** - indica la larghezza media, in millimetri, della porzione visibile di un oggetto orientabile
- **direction** - restituisce la direzione a cui punta l'oggetto, rispetto al dispositivo
- **tipPosition** - restituisce la posizione della punta dell'oggetto, in millimetri, dal *leap motion*
- **tipVelocity** - riporta la velocità della punta dell'oggetto in mm/s.

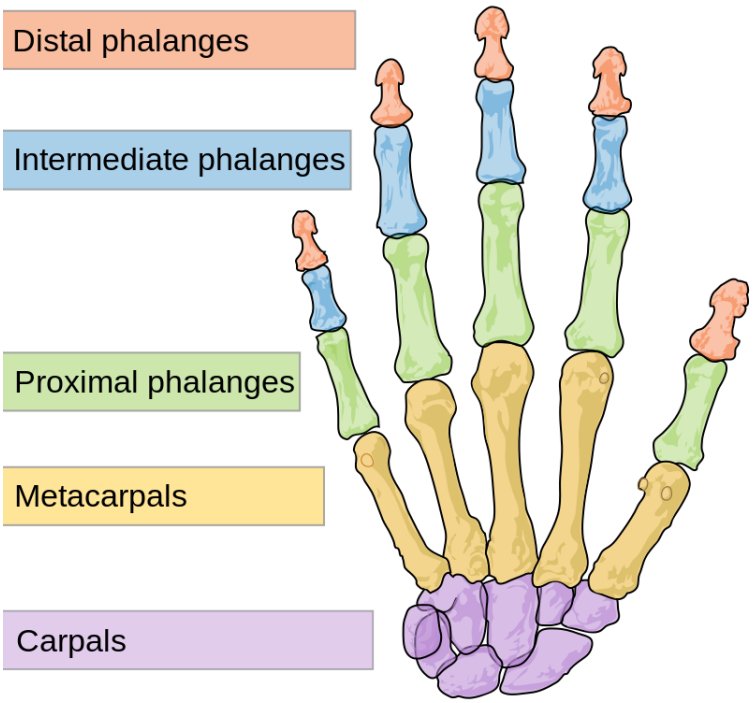

Figure 2.23: Le ossa della mano che il *Leap* è in grado di rilevare

Gesture - Il *leap motion*, come già detto in precedenza, è in grado di riconoscere, tracciare e registrare determinate gesture; Quando il *leap* riconosce una gesture, la inserisce all'interno di un frame. Se la gesture è continua, e si prosegue ad eseguirla, il dispositivo provvede ad aggiornare le informazioni associate nel frame successivo. Le *gesture circle* e *swipe* sono continue, mentre il *key tap* e lo *screen tap* sono discrete, quindi il *leap* indicherà ogni tap come una singola gesture. Le gesture che il dispositivo è in grado di riconoscere di default sono:

Circle - un singolo pointable traccia un cerchio nello spazio; la libreria del *leap motion* permette di poter determinare il vettore del cerchio tracciato, il numero di volte che il cerchio è stato aggiornato (e quindi eseguito), il suo raggio e il dito che lo ha eseguito.

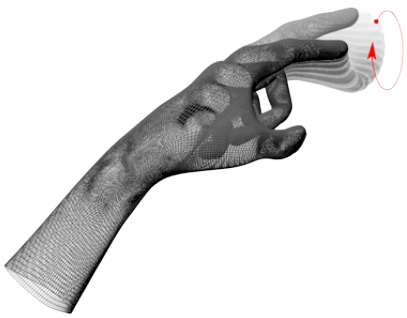

Figure 2.24: La *gesture circle* eseguita con un dito

Swipe - un movimento lineare della mano lungo l'asse X; in questo caso attraverso la libreria possiamo accedere alla sua direzione, la velocità in mm/s, il dito o il tool che lo ha eseguito, la posizione attuale dello swipe e la sua posizione di partenza.

Figure 2.25: La *gesture swipe*

Key Tap - un movimento di tapping del dito, in maniera simile alla pressione su un tasto della tastiera; gli attributi pubblici della classe *key tap* sono la direzione dell'oggetto che esegue la *gesture*, la posizione in cui viene eseguito e il *pointable* che lo esegue.

Figure 2.26: La *gesture key tap* eseguita con un dito

Screen Tap - un movimento di tapping del dito, in maniera simile alla pressione verticale su touch-screen; come già visto con il *key tap*, gli attributi dello *screen tap* sono direzione, posizione e *pointable*.

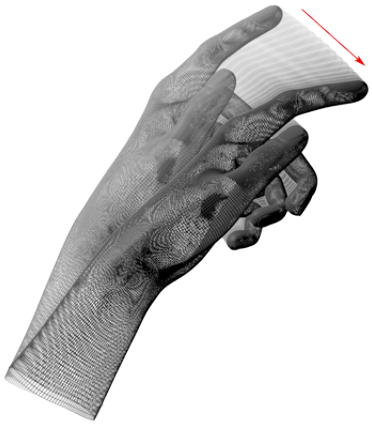

Figure 2.27: Lo *screen tap* eseguita con un dito

Apps Durante questi ultimi due anni sono state sviluppate e rilasciate un gran numero di app che spaziano su diversi ambiti, da quello video-ludico a quello educativo, da quello scientifico a quello del *computer controlls*, da quello robotico/spaziale (la *NASA* ha presentato una versione del robot *ATHLETE*[17], acronimo di *All-Terrain Hex-Legged Extra-Terrestrial Explorer* e progettato per l'esplorazione spaziale, telecomandato a distanza da un *leap motion*), all'esplorazione di realtà virtuali (come nel caso di *Google Earth*).

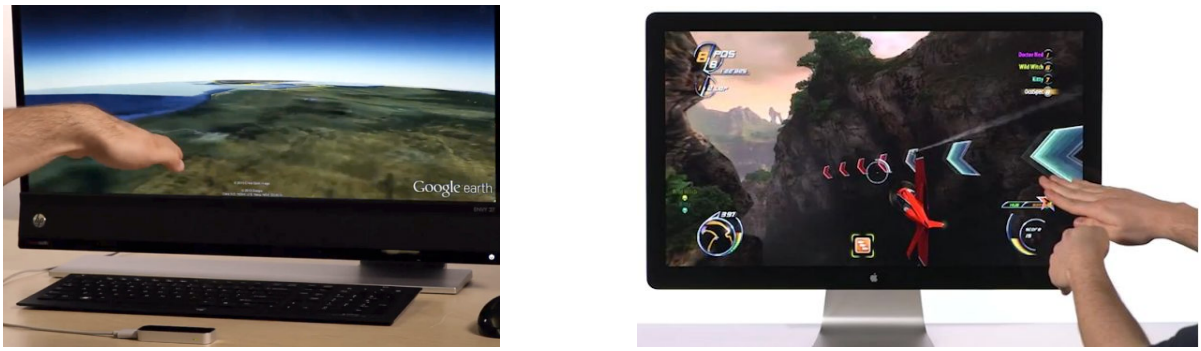

Figure 2.28: Esempi di applicazioni per il *Leap Motion*

Algorithms

Gli algoritmi utilizzati per l'interpretazione dei gesti si differenziano a seconda del tipo di informazioni che arrivano in input. La maggior parte di questi algoritmi si basano su dei punti chiavi rappresentati su coordinate in 3d; sulla base di questi movimenti è possibile rilevare delle gesture, e l'accuratezza dipende ovviamente dalla qualità dei dati processati e dall'algoritmo utilizzato.
In letteratura i due approcci più utilizzati sono: i modelli basati sul 3d e l'*apparance-based*

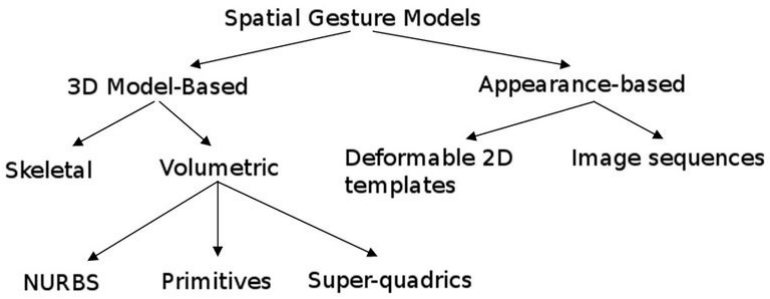

Figure 2.29: Schema riassuntivo[18] dei principali algoritmi di riconoscimento delle gesture e delle informazioni che utilizzano.

3d Model Based In questo tipo di algoritmo si utilizza l'analisi volumetrica dell'utente; questo tipo di approccio permette di ottenere degli ottimi risultati (tant'è che viene utilizzato soprattutto nella computer animation industry e per i programmi di computer vision specializzati); tuttavia risulta essere computazionalmente molto pesante e non sono stati ancora sviluppati dei software che permettano l'analisi di questi dati in tempo reale.

Per la costruzione di un modello 3d dell'utente ci si può basare anche su una semplice rappresentazione scheletrica del corpo, dove il software crea uno scheletro virtuale dell'utente in cui le varie parti del corpo sono rappresentate come segmenti. L'analisi in questo caso è fatta a partire dalla posizione e dall'orientazione dei singoli segmenti, e dalle relazioni tra essi; questo approccio presenta diversi vantaggi, innanzitutto gli algoritmi sono più veloci (in quanto devono processare solo dei parametri) e inoltre è possibile eseguire il pattern matching tra gli elementi di un database. In questa categoria ricade l'architettura (*GestIT*) che abbiamo utilizzato per rilevare le gesture.

Appearance-based Questo modello non usa una rappresentazione spaziale dell'utente, o di alcune sue parti, questo perché estraggono le informazioni direttamente dall'immagine (o dal video). La maggior parte si basano sulla *deformable 2D templates*[4] delle diverse parti del corpo dell'utente tracciate. I *deformable templates* vengono costruiti a partire da un insieme di punti intorno al contorno di un oggetto; dall'interpolazione di questi punti si ottiene, per approssimazione, il contorno dell'oggetto. Questo approccio permette di tracciare i movimenti della mano, anche se le gesture rilevabili sono molto limitate.

Problems

Come già detto, l'interazione naturale con i computer ha subito nell'ultimo decennio un aumento esponenziale (soprattutto grazie ai dispositivi *touch*).
Tuttavia c'è ancora molta strada da percorrere affinché l'interazione diretta possa veramente rappresentare un'alternativa efficiente all'interazione indiretta. Anzitutto serve ancora una tecnologia a basso costo che permetta di poter rilevare in maniera assolutamente accurata i movimenti compiuti dall'utente, riuscendo a gestire i problemi di occlusione, di sovrapposizione delle varie parti del corpo; inoltre non bisogna dimenticare del rumore che provoca la luce, sia solare che artificiale, o altre fonti di calore e di luce durante il tracciamento dei corpi e dei loro movimenti.
Un dispositivo robusto e affidabile dovrà quindi essere in grado di eliminare totalmente il rumore dai dati rilevati, in maniera tale da facilitare il rilevamento dei movimenti.

In secondo luogo è importante studiare e analizzare quali sono le gesture più adatte per l'esecuzione di determinati comandi, e porre in essere uno standard nei movimenti e nei gesti che l'utente deve eseguire durante l'interazione, al fine di poter facilitare nel lungo andare l'esperienza dell'utente e aumentare la sua confidenza con questo tipo di interazione.

Tuttavia l'interazione naturale tramite le gesture presenta anche problemi di carattere meramente fisico; un esempio sono il cosiddetto *Gorilla Arm* o quello dell'occlusione:

Gorilla Arm : è un effetto collaterale dell'uso dell'uso di schermi touch-screen, e venne ravvisato e studiato negli anni '80 solo dopo la commercializzazione dei primi dispositivi con tecnologia touch-screen; questo perché in laboratorio i test effettuati erano brevi e utilizzavano utenti esperti. La conseguenza di tale effetto collaterale fu quella di un forte rallentamento nella diffusione di questi dispositivi (tant'è che bisognerà aspettare altri vent'anni per l'esplosione di questo modello di interazione).
In sostanza il *braccio del gorilla* è l'effetto per cui, dopo un uso prolungato del braccio teso in avanti per compiere piccoli movimenti sullo schermo, l'utente avverte stanchezza e dolore all'arto, che inizia a muovere facendolo penzolare pesantemente come il braccio di un gorilla.

Fat Finger : è un problema tipico dei dispositivi touch dovuto fondamentalmente a due fattori: innanzitutto interfacce in cui gli oggetti (che possono essere link o pulsanti) da selezionare e manipolare sono troppo piccoli e vicino; in secondo luogo il fatto che il dito non è uno strumento molto preciso, può ricoprire anche una grande porzione dello schermo e quindi nascondere una parte dell'interfaccia all'utente.
Il verificarsi di queste due condizioni possono portare l'utente a commettere degli errori, facendogli selezionare degli elementi diversi da quelli desiderati, e rovinando la sua esperienza con tale sistema. Negli ultimi anni nuove metodologie di sviluppo hanno permesso di contrastare questo problema, sviluppando applicazioni e sistemi con interfacce più grandi, minimali e più semplici da utilizzare.

Ad ogni modo, proprio a causa di questi difetti è molto difficile pensare di poter realizzare un sistema complesso interattivo unicamente tramite gesture. In questi ultimi anni sono diversi gli esperti secondo cui l'interazione diretta non soppianterà mai

l'interazione indiretta, ma piuttosto verranno sviluppati sistemi ibridi, che permetteranno diverse modalità di uso contemporaneamente. Si va dunque verso una specie di biodiversità delle interfacce e dei dispositivi, che le renderà più naturali e più varie, ma che richiederà agli utenti una maggior flessibilità per passare da una forma di interazione all'altra.

2.2.2 Speech Recognition

È il processo tramite il quale un sistema trasforma i suoni prodotti dall'utente (o registrati nell'ambiente) in comandi da eseguire; per poter fare questo una parte dei sistemi di riconoscimento vocale richiede prima una fase di *training*, nella quale all'utente viene richiesto solitamente di leggere alcune frasi predefinite o, in alternativa, di scandire ogni singola lettera dell'alfabeto (tutto dipende dal tipo di applicazione in questione). Il fine di questo *enrolment* è quello di migliorare l'accuratezza e la precisione dell'operazione di riconoscimento; i sistemi che utilizzano il *traninig* sono anche chiamati *speaker dependent*.

Figure 2.30: La *Kinect* è un esempio di dispositivo in grado di riconoscere i comandi vocali. Tramite quest'ultimi l'utente può interagire sia col sistema che con i programmi in esecuzione.

Il riconoscimento vocale ha una storia ben più lunga rispetto al rilevamento delle gesture, tuttavia solo di recente ha potuto godere di nuove attenzioni da parte della grande industria commerciale, grazie soprattutto alle innovazioni fatte nel campo del *Data Maining*, nella gestione dei *big data* e nell'*Artificial Intelligence*.

Infatti già nel 1932, presso la *Bell Labs*, vennero avviati degli studi con l'obiettivo di comprendere la scienza che sta alla base della percezione del linguaggio[19]. Nel dopoguerra, esattamente nel 1952, venne sviluppato il primo sistema in grado di riconoscere singole parole; per farlo localizza, all'interno dello spettro di potenza che descrive la parola pronunciata dall'utente, determinati formanti. A causa dei limiti tecnologici, però, il sistema riconosceva un vocabolario molto limitato di dieci parole.

Tra gli anni '60 e gli anni '80, grazie ai nuovi algoritmi (tra tutti il cosiddetto modello *di Markov*) e alle nuove tecnologie che hanno incrementato la capacità dei computer, i ricercatori riuscirono a sviluppare sistemi di riconoscimento più complessi, in grado

di riconoscere fino a mille parole. Tra i software sviluppati in questo periodo, più interessanti sono:

- il sistema che utilizza il modello di *Hidden Markov*, che permette di combinare dati di diverso tipo (come acustica, linguaggio e sintassi) in un unico modello probabilistico;
- e il sistema *Tangora*[20], che abbandona gli approcci indirizzati ad emulare i processi cerebrali dell'uomo a favore di modelli statistici.

Infine, a partire dagli anni 2000, i nuovi sistemi di riconoscimento vocale vengono sviluppati combinando alla statistica anche l'*artificial intellinge*; in particolare il *deep learning* e le *neural networks*[34], che hanno permesso di ridurre del quasi 40% la percentuale di errore di questi sistemi.

Models and Algorithms

Assieme ai modelli acustici e linguistici gli algoritmi e i modelli più utilizzati per lo sviluppo di algoritmi di riconiscimento sono:

Hidden Markov Model : come già accennato in precedenza è un modello[23] statistico dove l'output è rappresentato da una sequenza di simboli; più precisamente è una *Markov chain*[24] in cui però gli stati non sono osservabili direttamente e rappresenta l'esempio più semplice di *rete Bayesiana dinamica*. In particolare:

- la catena ha un certo numero di stati;
- gli stati evolvono secondo una *Markov chain*
- ogni stato genera un evento con una certa distribuzione di probabilità (solitamente sono una combinazione di diverse *diagonal covariance Gaussians*) che dipende solo dallo stato;
- l'evento è osservabile ma lo stato no, di conseguenza la sequenza di tokens generata dal modello fornisce alcune informazioni sulla sequenza degli stati;

Questo modelli sono utilizzati nel riconoscimento vocale perché:

- un segnale vocale può essere visto come un *piecewise stationary signal* o come un *short-time stationary signal*.
- inoltre gli *HMMs* possono essere "addestrati" automaticamente, sono semplici da usare e computazionalmente flessibili.

L'*hidden Markov model* restituisce in output una sequenza di vettori in n-dimensioni, una sequenza ogni n-secondi. Questi vettori rappresentano i coefficienti del *cepstrum*, ovvero il risultato della trasformata di Fourier applicata allo spettro in decibel del segnale inviato dall'utente. Questo modello, oltre che nel riconoscimento vocale, viene usato anche per il riconoscimento della scrittura a mano, nel riconoscimento di textures e nella bioinformatica.

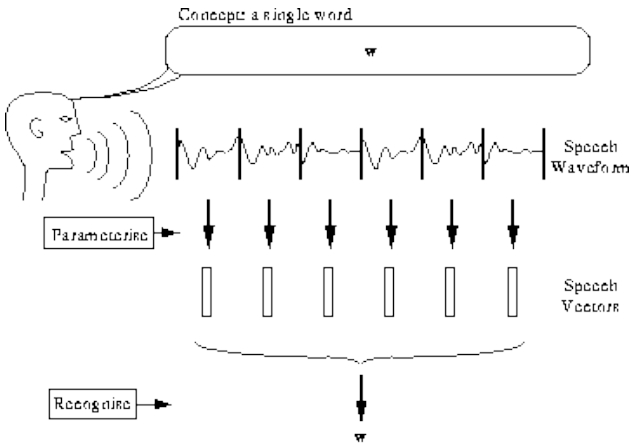

Fig. 1.2 Isolated Word Problem

Figure 2.31: Schema di funzionamento di un *HMM*.

Dynamic time warping : era l'approccio[25] più utilizzato in questo campo di ricerca fino all'avvento dei sistemi basati sul modello nascosto di Markov; è un algoritmo che consente l'allineamento di due sequenze e permette di calcolare la distanza e la similarità tra le stesse in termini di tempo o velocità; generalmente nel calcolo della corrispondenza sono applicate delle limitazioni, ovvero dev'essere garantita la monotonicità nelle corrispondenze e deve esistere un limite massimo di possibili corrispondenze tra elementi contigui della sequenza.

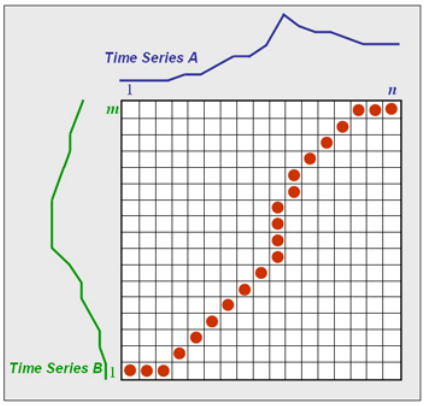

Figure 2.32: Esempio di funzionamento di un algoritmo *DTW*.

Neural networks : sono dei modelli[22] matematici che simulano artificialmente il comportamento del cervello umano; sono costituiti da un insieme di unità o nodi (*neuroni*), e interconnessi tramite dei link a cui vengono associati dei pesi (e che rappresentano le *connessioni sinaptiche*).
Questi modelli matematici possono essere utilizzati sia per ottenere una comprensione delle reti neurali biologiche e sia per risolvere problemi di intelligenza artificiale; nel riconoscimento vocale viene usato per classificare i fonema e isolare le diverse parole. Recentemente sono stati sviluppati gli algoritmi *Recurrent Neural Networks* e *Time Delay Neural Networks* che hanno permesso di identificare

la dipendenza di latenza temporale tra i vari segnali, migliorando di gran lunga le prestazioni nel riconoscimento al prezzo però di un elevato costo computazionale.

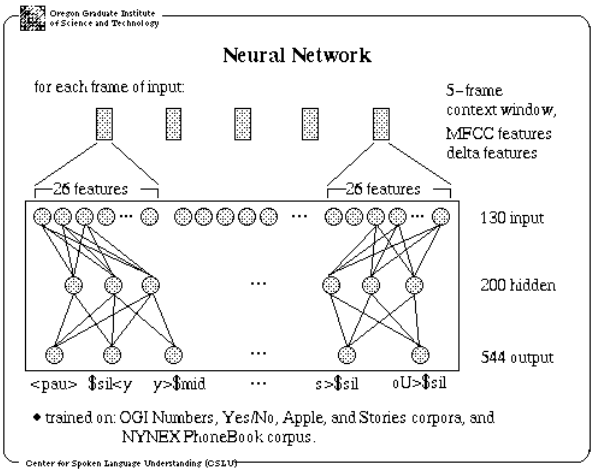

Figure 2.33: Un esempio di rete neurale per il riconoscimento vocale.

Deep Neural Networks : chiamate anche *DNN*, derivano dalle architetture *deep learning* e sono delle reti neurali artificiali "profonde"; ovvero reti dotate di diversi livelli di elaborazione tra l'input e l'output, dove l'input è rappresentato dai dati grezzi del segnale vocale ed opera direttamente sulle *waveforms*. Le *deep neural network* vengono usate per modellare delle relazioni non lineari complesse e generano dei modelli compositi, nelle quali i livelli più interni operano sui dati in uscita dai livelli più alti; grazie a queste caratteristiche il sistema è dotato di un'enorme capacità di apprendimento, il che gli permette di poter modellare e processare segnali vocali anche complessi.
Negli ultimi anni il *deep learning* è diventato molto diffuso nel campo del riconoscimento vocale per descrivere i modelli acustici.

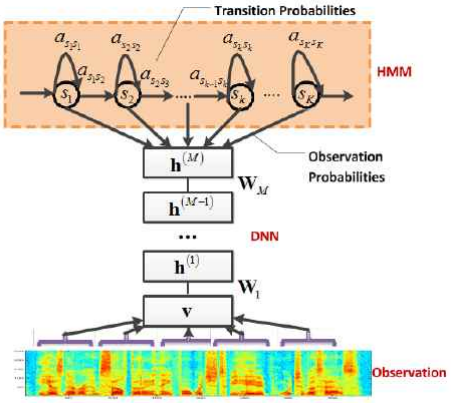

Figure 2.34: Esempio di sistema di riconoscimento vocale che combina due modelli: il *DNN* e l'*HMM*.

Applications

Sono molti gli ambiti in cui viene utilizzato il riconoscimento vocale. I principali sono:

In-Car systems : nelle auto il riconoscimento vocale può essere utilizzato per rendere più sicura la guida[26], permettendo al guidatore di poter eseguire delle semplici azioni (come selezionare una stazione radio, comporre una chiamata, interagire con il navigatore satellitare, ecc.) senza compromettere la sua attenzione durante la guida.

Health care : il riconoscimento vocale ben si adatta alla conversione di un discorso in un testo, e questo è lo scopo principale per cui viene utilizzato in campo medico[27], dove si ha a che fare ogni giorno con un enorme mole di documenti di ogni tipo (che possono andare dalle diagnosi di un paziente, all'analisi di una radiografia o al rapporto di un'operazione); infatti nel settore sanitario il riconoscimento vocale trova applicazione sia nei sistemi di elaborazione della documentazione di tipo front-end che in quello di tipo back-end:

- i sistemi front-end, in questo caso, sono quelli dove l'utente détta il rapporto da presentare, il sistema genera in real-time il relativo testo che potrà poi essere modificato e accettato dall'utente stesso;
- i sistemi back-end, invece, risultano essere totalmente trasparenti all'utente; anche questi generano un testo a partire dalle dettature eseguite dall'utente, ma non può essere modificato successivamente dall'utente.

Il riconoscimento vocale è usato anche in ambito terapeutico, infatti diversi studi (condotti su pazienti affetti da *AVM* (acronimo di *cerebral arteriovenous malformation*) dimostrano che un uso prolungato di questi sistemi aiuti i pazienti a rinforzare la memoria a breve termine. Altre applicazioni permettono invece di poter assistere e migliorare la vita dei pazienti sordi (o comunque con problemi di udito); il sistema di riconoscimento viene utilizzato in questo caso per generare automaticamente dei sottotitoli, permettendo all'utente di poter interagire e discutere con altre persone. I sistemi di riconoscimento vocale vengono usate anche per l'assistenza alle persone affette da disabilità agli arti (o comunque infortunate), o per aiutare i pazienti con gravi disabilità linguistiche a migliorare la loro pronuncia e i loro difetti linguistici.

Military/Civil Aviation : il riconoscimento di comandi vocali è stato, nel corso del tempo, testato ed usato anche in campo militare[28]; più precisamente in campo aeronautico all'interno degli aerei da guerra, dove il sistema di riconoscimento venne in un primo momento testato per il settaggio delle frequenze radio, per inviare direttive al pilota automatico e per il settaggio delle armi e delle coordinate *steer-point*; tuttavia a causa delle interferenze create dal respiro del pilota e dal condizioni in cui opera il mezzo stesso, questo sistema è stato limitato al settaggio delle frequenze audio e per le coordinate.
Stesso discorso si può fare per l'applicazione del riconoscimento vocale negli elicotteri, dove a causa dell'elevata quantità di rumore, può essere usato solo per controllare le comunicare radio e settare i sistemi di navigazione. Per quanto riguarda l'aviazione civile, i sistemi di riconoscimento vocale sono utilizzati per l'addestramento dei controllori del traffico (o *ATC*).

Telephony : nel settore telefonico l'utilizzo del riconoscimento vocale è molto diffuso, sopratutto nel campo degli smarthphone; in questo campo i sistemi di riconoscimento vocale sono utilizzati all'interno di applicazioni intelligenti (che operano similmente come degli assistenti personali) per eseguire determinati comandi (come inviare un messaggio, fare una chiamata, effettuare una ricerca) da parte dell'utente. Un esempio sono i vari *Siri*[29] della *Apple*, *Cortana*[30] della *Microsoft* oppure *Google Now* sviluppata dalla *Google*. Il riconoscimento vocale viene usato anche dalle compagni telefoniche nell'assistenza e nell'esecuzione di semplici operazioni, come ad esempio effettuare la ricarica di un cellulare o attivare delle offerte.

Education : il riconoscimento vocale trova applicazione anche nell'educazione e nell'apprendimento. Un esempio sono le applicazioni[31] che aiutano gli utenti ad imparare una lingua, in cui il riconoscimento vocale viene utilizzato per insegnare la pronuncia corretta. Altre applicazioni invece aiutano gli studenti fisicamente impossibilitati (per un infortunio o per una patologia) di poter comunque studiare, permettendogli per esempio di poter tradurre un loro discorso in un testo scritto, oppure di poter eseguire delle operazioni e delle ricerche con il computer.

Videogame : come accennato in precedenza, la *Kinect* è in grado di riconoscere i comandi vocali impartiti dall'utente, attraverso la quale può interagire col sistema in maniera più naturale, rendendo più ricca e varia l'esperienza di gioco.

Automatic Translation : sono in sviluppo dei software per la traduzione in tempo reale di un testo o di un discorso parlato.

Problems

Le performance di questi sistemi dipendono in gran parte dal numero di parole su cui devono operare, dalla grandezza del dizionario, dal numero degli speaker, dal livello di rumore presente nell'ambiente e dalla strumentazione utilizzata; possiamo perciò elencare i principali problemi attuali, o superati, del riconoscimento vocale come segue:

Accent : un elemento che influenza le prestazioni dei sistemi di riconoscimento vocale è l'accento dell'utente. Nei sistemi personali, dove all'utente è richiesto di eseguire una fase di *training* del sistema pronunciando delle frasi di default (oppure facendo lo spelling dell'alfabeto) il problema non si pone; tuttavia il discorso cambia per quei sistemi installati all'aperto o comunque molto frequentati. In questo caso non è concepibile eseguire una fase di *training* per ogni nuovo utente, perciò il sistema dovrà eseguire nella fase di pre-elaborazione una pulitura del segnale vocale in modo da poter rimuovere le imperfezioni dovute all'accento, al costo ovviamente di una maggior complessità computazionale.

Environment : l'efficacia di un sistema di riconoscimento dipende anche dall'ambiente in cui viene adoperato. Un maggior precisione si ha ovviamente in ambienti chiusi con un rumore di sottofondo nullo, tuttavia la precisione cala quasi drasticamente negli spazi aperti a causa di una maggiore dispersione del segnale stesso (che si può comunque risolvere con l'aiuto di un microfono) e la presenza di un tasso di rumore molto elevato (per esempio grida, intemperie, ecc.).

Dictionary : le nuove tecnologie hanno permesso l'utilizzo di dizionari sempre più grandi, tuttavia questo richiede un maggior costo computazionale per poter ricercare e riconoscere i pattern associati ai vari vocaboli.

Noise : uno dei problemi principali nel campo del riconoscimento è il rumore; in questo caso il rumore si presenta come tutti quei suoni estranei al segnale vocale dell'utente o che comunque lo distorcono. Come indicato in precedenza, la presenza di rumore dipende dall'ambiente in cui si utilizza il sistema; attualmente i sistemi di riconoscimento vocale sono inadatti per operare in spazi molto frequentati (parchi o centri commerciali per esempio) o dove sono presenti rumori di sottofondo molto forti (fabbriche, miniere per fare un esempio).

Physics(acoustics) : gli algoritmi di riconoscimento vocale devono essere in grado anche di comprendere, e di interpretare, le relazioni che intercorrono tra la fisica dei suoni e i meccanismi fisiologici che consento all'uomo di poter emettere e capire dei suoni, tuttavia queste relazioni sono difficili da rappresentare in termini informatici.

Come nel caso del riconoscimento delle gesture, anche nel riconoscimento vocale alcuni teorici affermano che questi problemi potranno essere risolti attraverso la multimodalità, e quindi più sistemi di interazione che convivono e che si completano a vicenda.

2.3 Gesture Modeling

In questa sezione andremo ad analizzare e studiare quali sono gli approcci utilizzati per la modellazione delle gesture. In letteratura troviamo principalmente due tipi di modelli, ovvero quelli classificativi e quelli dichiarativi:

Machine Learning and Classifier : questo tipo di approccio va a definire e a riconoscere la gesture attraverso l'utilizzo di classificatori e di sistemi di apprendimento automatico. In letteratura sono stati proposti tanti tipi di modelli basati su classificatori diversi, dalle reti dinamiche bayesiane[32] alle hidden Markov model[33], dalle reti neurali[34] alle multiclass support vector machine [35].
L'approccio basato sulle catene di Markov e sulle *HMM* rimane uno dei più utilizzati. Nel 1992 Yamato et al. pubblicano uno dei primi articoli[36] sul riconoscimento di gesture. In questo tipo di approccio vengono utilizzati una *HMM* discreta e una sequenza di vector-quantized labels per riconoscere sei tipi di colpi utilizzati nel tennis in base al movimento compiuto dalla mano. Prima di operare attraverso il modello di Markov, sull'immagine vengono eseguiti diverse operazioni di preprocessing, come ad esempio l'applicazione di un filtro passa-basso per ridurre il rumore e della background subtraction per estrarre gli oggetti in movimento. In seguito si procede con la binarizzazione degli oggetti rilevati per generare i cosidetti *blobs*, che rappresenteranno la posa dell'utente tracciato. I *blobs* sono rappresentati tramite dei pixel e descritti attraverso le VQ-labels sulle quali poi verrà applicato l'*HMM*.
Un esempio pratico di classificatori utilizzati per il riconoscimento e la definizione delle gesture sono gli algoritmi utilizzati dalla *Kinect*:

AdaBoost : acronimo di *Adaptive Boosting*, è un sistema di machine learning che si basa sull'idea di creare un delle accurate e precise regole, a partire da un insieme di regole imprecise e deboli, e utilizzando un classificatore di tipo *boost*. Questo sistema di apprendimento è utilizzato dalla *Kinect* per il riconoscimento delle pose.

RFR : L'*RFRProgress*[38] è invece un'altra macchina di apprendimento utilizzata per determinare lo stato di progresso di una gesture. Ciò viene determinato attraverso l'utilizzo di un sistema di etichettatura su dei segnali generati dal sistema durante il tracciamento di una gesture. È quindi un rilevatore *context based* che fornisce informazioni sullo stato della gesture solo dopo che questa è stata eseguita.

Questi modelli sono utilizzati in particolar modo dai dispositivi di tracciamento come la *Kinect* o il *Leap Motion*, e hanno il vantaggio di essere abbastanza precisi. Lo svantaggio però di questo approccio è legato al modo con cui il sistema avvisa quando una gesture è stata riconosciuta; ovvero vengono generati dei singoli eventi che comunicano se la gesture è stata eseguita oppure no. Questo, come abbiamo visto, ha delle ripercussioni anche sugli eventuali sistemi di feedback e feedforward presenti; infatti i dati ottenibili da questi tipi di classificatori sono insufficienti per dare all'utente dei feedback efficienti.

Declaritive : l'approccio dichiarativo invece consiste nel suddividere la gesture in piccole parti (in tante sub-gesture) ognuno dei quali descrive temporalmente le fasi di esecuzione della gesture. Il vantaggio di questo metodo è che durante l'esecuzione si riesce a raccogliere una gran quantità di dati circa il movimento che l'utente sta eseguendo e lo stato della gesture, a discapito però di una minor precisione.
Durante lo sviluppo della tesi abbiamo deciso di utilizzare dei sistemi dichiarativi per la definizione di nuove gesture; e questo perché le informazioni raccolte a tempo di esecuzioni possono essere usate per fornire dei feedback e dei feedforward all'utente.
Nelle prossime sottosezioni andremo ad analizzare più nel dettaglio quali sono i principali modelli proposti per questo tipo di approccio.

2.3.1 GestIT

GestIT è un framework che permette la definizione dichiarativa e composizionale delle gesture, indipendentemente dal dispositivo di riconoscimento utilizzato (per esempio *Kinect*, *Leap Motion*, schermi multitouch, ecc.).
Su *GestIT* una gesture viene quindi rappresentata come un'espressione, e ogni espressione è caratterizzata da un insieme di *ground term*, che descrivono i singoli movimenti che compongono la gesture stessa, e sui cui possono agire a loro volta una serie di operatori di composizione chiamati *composite term*. Questi permettono ai programmatori di mettere in relazione più singoli ground term, consentendo quindi la definizione di gesture più complesse, come un insieme di gesture più piccole (sub-gesture); l'implementazione di questi operatori è basata sulla definizione di relazioni temporali nella modellazione di task ed interfacce[53].
Semplificando una gesture può essere vista come un albero, dove le foglie sono rappresentate dai *ground term* mentre i nodi interni dell'albero (compresa la radice) sono

descritti attraverso i cosiddetti *composite term*.

Oltre alla definizione di nuove gesture, *GestIT* mette a disposizione anche degli strumenti per il loro riconoscimento:

- a partire dai dati rilevati dal dispositivo di tracciamento vengono costruiti, di volta in volta, dei *token* che rappresentano il movimento o il comando vocale tracciato. Quindi, per esempio possono definire lo scheletro dell'utente, descrivere la posizione delle sue mani e delle sue dita o ancora rappresentare un segnale acustico, tutto dipende dal tipo di interazione naturale che si vuole implementare;
- attraverso una lista che contiene la definizione di tutte le gesture che si possono eseguire, il token viene inviato a tutte le espressioni; tramite apposite funzioni, l'espressione verificherà se la gesture è stata eseguita o meno. In questo caso ogni espressione e sub-gesture provvede ad eseguire le eventuali funzioni legate al completamente o all'errore di una espressione, e a comunicare queste informazioni tramite degli appositi eventi.

Nei prossimi capitoli analizzeremo l'implementazione di questo framework in maniera più dettagliata.

2.3.2 Proton

Proton[39] è un framework costruito più specificatamente per l'interazione multi-touch, e cerca di trovare una soluzione a questi problemi:

- come aggiungere e far coesistere delle nuove gesture all'interno di un database di gesture già esistenti;
- come si può estendere il codice già esistente per riconoscere in maniera affidabile il nuovo insieme di gesture ottenuto.

In *Proton* ogni gesture viene definita in maniera dichiarativa attraverso un'espressione regolare, come una sequenze multipla di eventi "touch"; il suo riconoscimento, invece, avviene a partire dagli eventi generati sulla base dell'input dell'utente. Per la creazione delle gesture *Proton* mette a disposizione un editor grafico, il cosiddetto *gesture tablature*, che permette agli sviluppatori di poter creare delle nuove e più complesse gesture in maniera più semplice e diretta, aiutati attraverso una notazione grafica.
A partire dai dati inseriti dal programmatore nell'editor, il sistema trasforma questi valori in un'espressione regolare che definisce la gesture; prima di inserire la gesture nel database, Proton provvede ad analizzare e confrontare l'espressione appena generata con quelle già presenti, al fine di evitare di inserire due gesture con la medesima definizione, che potrebbero generare dei conflitti durante l'esecuzione del programma.

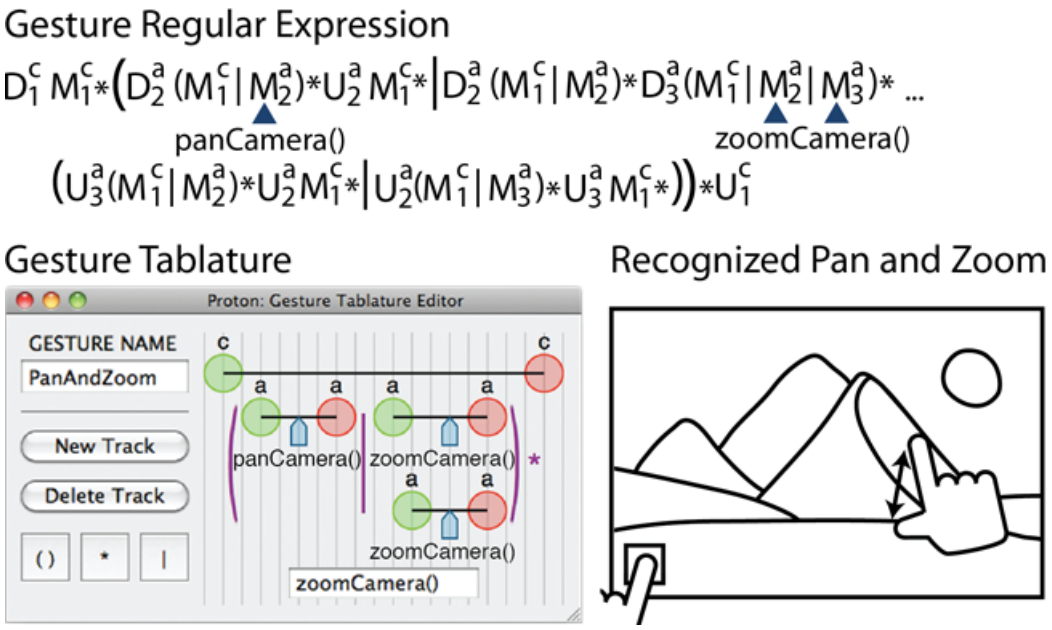

Figure 2.35: L'editor di *Proton*, un esempio delle espressioni generate e che descrivono delle gesture.

I *touche events* sono rappresentati attraverso degli eventi che contengono tre tipi di informazioni:

- il tipo di touch (down, move o up);
- l'identificativo associato (first, second, third, ecc.);
- e il tipo di oggetto selezionato dall'utente.

2.3.3 Gispl

Ora andremo ad analizzare *GlSpL*[40], un linguaggio per la creazione di gestural interface. *GlSpL* è un linguaggio formale che permette a ricercatori e sviluppatori di descrive in maniera diretta e senza ambiguità gli elementi dell'interfaccia che possono essere utilizzati dall'utente per eseguire una gesture.
Tale sistema si base su una sintassi definita tramite *JSON* e supporta diversi tipi di input: multi-touch, penne digitali, l'input proveniente da più mouse contemporaneamente, interfacce tangibili(vedi anche [41][42]), o il movimento di parti del corpo dell'utente. L'obbiettivo di questo linguaggio è quello di semplificare l'implementazione degli algoritmi di riconoscimento, permettendo il riutilizzo di componenti già sviluppate.
GlSpL si basa su una descrizione dichiarativa delle gesture, inteso come un insieme di eventi innescati dall'utente; ogni evento è descritto attraverso delle features, che rappresentano le informazioni più basilari inviate dal sistema (come ad esempio gli oggetti tracciati dal sistema, ecc.).

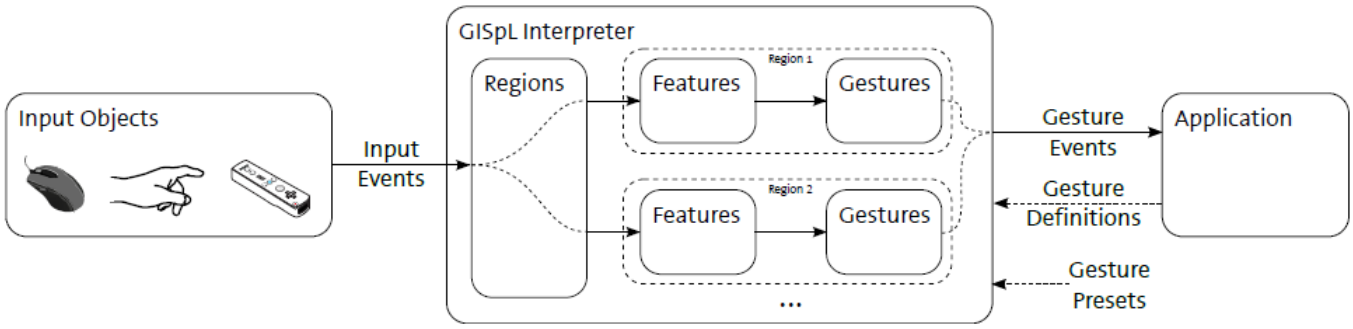

Figure 2.36: Una rappresentazione grafica del funzionamento di *GISpL*.

2.4 Feedback and Feedforward

L'interazione con le macchine e i sistemi informatici è sempre stata caratterizzata dalla difficoltà, per gli utenti inesperti, di potervi interagire in maniera semplice e senza rischio; questo avviene soprattutto con macchinari specializzati, che presentano una scarsa *affordance* o con la quale l'utente non ha confidenza.
I problemi principali riscontrati durante l'interazione sono:

- l'incertezza nel prevedere che effetto avrà nel sistema un determinato comando. Molti degli errori che l'utente commette sono dovuti a questo difetto, e questo ovviamente non aiuta, in quanto porta gli utenti inesperte ad allontanarsi dal sistema in questione a favore di altri più prevedibili;

- la difficoltà di eseguire una qualche azione in quanto l'utente non si ricorda (o non sa) come eseguirla (per esempio, quali opzioni deve selezionare, quali proprietà deve settare, ecc). Nei sistemi informatici che prevedono un'interazione naturale questo si traduce nell'incapacità dell'utente di eseguire in maniera corretta la gesture affinché il dispositivo possa riconoscerlo;

- molti sistemi non prevedono la possibilità di annullare un comando eseguito per errore; questo problema, combinato alla difficoltà e all'incertezza non fanno altro che allontanare gli utenti da tali sistemi.

Per far fronte a questi due problemi, aiutando l'utente inesperto ad interagire con una macchina, sono stati sviluppati i cosiddetti sistemi di *feedback* e di *feedforward*. Durante il lavoro di tesi ci siamo occupati di analizzare, studiare e implementare delle metodologie di feedback e di feedforward per le interfacce di sistemi dedicati all'interazione naturale, sia tramite gesti che tramite comandi vocali.

2.4.1 Feedback

Nell'interazione naturale i meccanismi di feedback vengono utilizzati per mostrare all'utente il comando riconosciuto in seguito ad un'interazione, e per indicare lo stato del sistema e dell'algoritmo di riconoscimento. Permettere all'utente di sapere quale comando il software abbia riconosciuto è molto importante per due motivi:

1. in primo luogo perché permette all'utente di sapere se il movimento che ha eseguito, o il comando vocale che ha impartito, è quello corretto oppure no; se il feedback è in tempo reale l'utente può capire immediatamente cosa ha sbagliato (aggirando così i limiti della memoria a breve termine del cervello umano) e correggere l'errore.

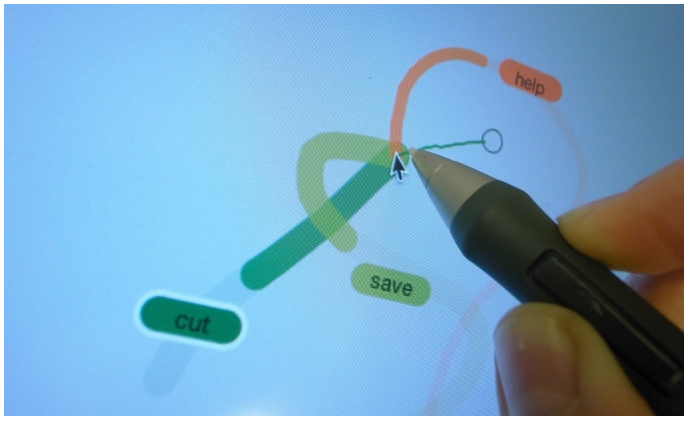

Figure 2.37: I feedback possono aiutare l'utente ad eseguire delle gesture.

2. inoltre, è importante che il sistema comunichi all'utente quale sarà lo stato del programma in seguito all'esecuzione della gesture riconosciuta; in tal modo l'utente saprà in anticipo se l'azione che vuole eseguire è quella desiderata o meno, e di conseguenza, decidere di continuare o di non completare la gesture.

2.4.2 Feedforward

I sistemi di feedforward sono invece utilizzati per consigliare l'utente e guidarlo durante l'interazione; questo può essere fatto in diversi modi: per esempio mostrando per ogni opzione selezionabile quale sarà il suo effetto sul programma, oppure evidenziando in un menù le opzioni che l'utente deve selezionare per eseguire un certo task, o ancora, nel contesto della gesture interface, aiutando l'utente ad eseguire una gesture tramite dei consigli dati passo per passo. In letteratura i sistemi di feedforward vengono suddivisi in base al livello di dettaglio e di aggiornamento:

Level of detail : il livello di dettaglio può andare da un piccolo suggerimento su come eseguire la gesture alla descrizione dell'intera gesture;

Update rate : per quanto riguarda il livello di dettaglio questo può variare dai singoli suggerimenti visualizzati prima d'iniziare l'esecuzione ad un aiuto continuato durante l'esecuzione del programma.

2.4.3 Algoritms

In letteratura esistono diverse metodologie per implementare sistemi di feedback e di feedforward in interfacce naturali e non, a dimostrazione della crescente attenzione che sta interessando questo aspetto dell'interazione uomo-macchina. Le principali architetture che abbiamo studiato (e da cui abbiamo preso spunto) durante la tesi sono:

OctoPocus : è una guida dinamica[43] che fonde dei sistemi di feedfoward e di feedback per aiutare l'utente nell'eseguire ed imparare un certo insieme di gesture; il funzionamento si basa su un'idea molto semplice ma efficace (come hanno dimostrato i vari test): nel momento in cui l'utente inizia ad eseguire un gesto, vengono mostrate all'utente il movimento di tutte le gesture simili (ovvero che hanno un evoluzione simile al movimento registrato fino ad allora). Questo permette all'utente di acquisire familiarità con l'insieme delle gesture, aiutandolo a correggere gli eventuali errori. Il sistema di feedforward si occupa di:

- creare per ogni gesture una rappresentazione grafica dei movimenti che lo caratterizza;
- quando l'utente iniziare l'esecuzione di una gesture vengono stampate la traccia di tutte le varie gesture disponibili, mettendo in evidenza solo la parte iniziale. Dal momento in cui l'utente prosegue nel seguire il percorso del gesto, la parte evidenziata avanza seguendo lo stato d'avanzamento dell'esecuzione della gesture;
- contemporaneamente all'input dell'utente, provvede ad eliminare la traccia delle gesture che differiscono dal movimento eseguito dall'utente.

Invece il meccanismo di feedback si occupa di cambiare lo spessore del movimento che l'utente ha già eseguito, evidenziando al contempo il movimento che manca per completare una certa gesture; ciò permette all'utente di poter correggere il proprio movimento in tempo reale e di sapere quanto manca alla conclusione della gesture.

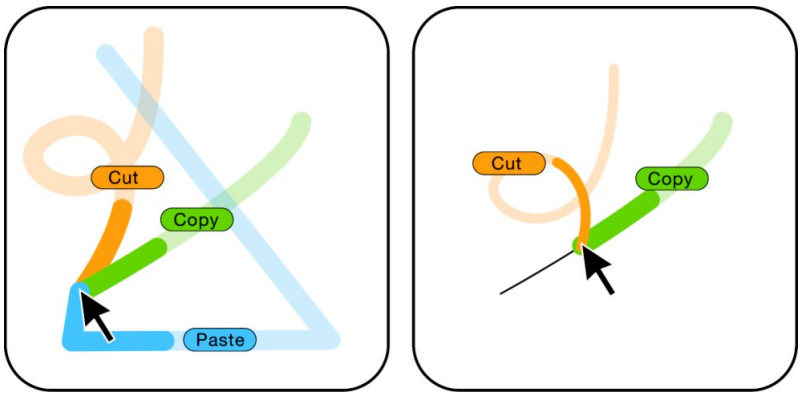

Figure 2.38: Funzionamento della guida dinamica *OctoPocus*.

LightGuide : è un sistema[44] non molto dissimile da *OctoPocus*, si presenta infatti come una guida in tempo reale, per un *interface gesture*, che consiglia l'utente circa l'esecuzione di una gesture. Tramite l'utilizzo di un proiettore, e di una depth camera, i consigli (che possono essere delle frecce o delle icone che possono assumere colori diversi in caso di errore) proiettati direttamente sulla parte di corpo che utente deve muovere; questo permette all'utente di poter migliorare e perfezionare i movimenti richiesti per completare una gesture, in particolar modo gesture complesse come ad esempio degli esercizi fisici o di fisioterapia. La componente feedback della guida provvede a fornire all'utente informazioni di vario tipo, tra cui:

- lo stato di progressione della gesture in esecuzione;
- il prossimo movimento da eseguire per completare la gesture;
- il grado di correttezza del movimento eseguito, e in caso di errore, la "distanza" tra il movimento eseguito e quello corretto.

Figure 2.39: I consigli vengono mostrati direttamente sulla mano, aiutando l'utente sul movimento da eseguire.

An architecture for generating interactive feedback : è un'architettura[45] sviluppata per generare dei feedback interattivi continui che aiutino l'utente durante l'esecuzione di azioni ambigue; tale sistema si basa sul modello di incertezza sviluppato a partire dall'algoritmo di *Monte Carlo*[46] e sul tracciamento di più interfacce (una per ogni possibile sequenza di input che l'utente può inviare al programma). L'architettura si occupa quindi di ridurre il numero delle possibili interfacce (in base all'input dell'utente), di fondere le rimanenti in una singola interfaccia e visualizzarla su schermo; in tal modo l'utente è informato su:

- quale sarà lo stato del programma dopo l'esecuzione di una certa azione;
- in caso di azioni simili il sistema mostra gli stati con la probabilità più alta;
- una volta completato il comando, il sistema permette all'utente di ritornare allo stato precedente o di selezionare un altro stato;

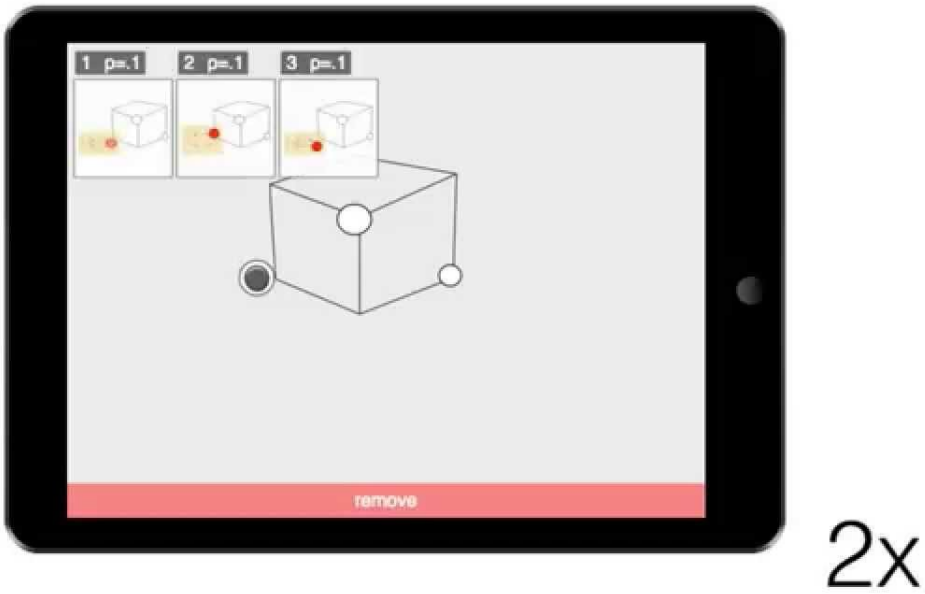

Figure 2.40: In base all'input dell'utente vengono mostrati gli stati più possibili.

Chapter

3
ABSTRACT MODEL

Il nostro lavoro di tesi ha come obbiettivo quello di realizzare dei modelli per la generazione di feedback e feedforward all'interno delle *gestural interfaces*. Per poterlo mettere in essere è necessario utilizzare una definizione dichiarativa e compositiva delle gesture; infatti ciò ci permette di poter raccogliere a tempo di esecuzione tutte le informazioni circa lo stato delle gesture riconosciute dal sistema sulla base dell'input inviato dall'utente. Questi dati verranno poi utilizzati sia per informare l'utente di quali sono le gesture attualmente riconosciute, sia per determinare quale sarà lo stato dell'interfaccia in seguito all'esecuzione di una particolare gesture.

3.1 Gesture Definition

Nel nostro modello, basato sul framework *GestIT*, le gesture sono definite come delle espressioni che mettono in relazione tra loro più movimenti minori, composti tramite un insieme di operator; gli elementi sono i cosiddetti *term*, che si dividono in:

Ground Term : che rappresentano una parte del movimento che l'utente deve eseguire per completare una gesture; un esempio può essere l'azione di chiusura della mano. In generale possono essere di tre tipi:

Start : definisce il gesto iniziale della gesture. Solitamente è rappresentato da un movimento o un comando vocale statico, in modo tale che il sistema possa riconoscere in maniera accurata l'inizio dell'esecuzione di una gesture, ed evitando i problemi legati all'ambiguità dell'input e alla segmentazione dei dati;

Move : rappresenta i movimenti intermedi di una gesture. Questi movimenti sono definiti seguendo un approccio temporale su particolari parti del corpo tracciate dal dispositivo (come il movimento di una mano o di un braccio). Gli elementi fondamentali in questo caso sono rappresentati da informazioni temporali quali le coordinate temporali, la direzione, la distanza percorsa, la velocità ecc..

End : descrive il movimento finale della gesture, e in genere anche questo tipo di *ground term* è rappresentato da un movimento statico; Una gesture viene riconosciuta quando tutte le *sub-gesture* che la descrivono sono stati eseguite correttamente.

Composite Term : sono gli operatori che mettono in relazione i *Ground Term*. In *GestIT* gli operatori utilizzati sono sei, ovvero *Choice*, *Iterative*, *Sequence*, *Parallel*, *Disabling* e *Order Independence*:

Choice ([ ]) - consente il riconoscimento contemporaneo di tutte le gesture o sub-gesture che contiene al suo interno. Va in uno stato di *error* solo se tutti i suoi figli si ritrovano nello stesso stato;

Iterative (*) - permette il riconoscimento del movimento descritto un numero indefinito di volte;

Sequence (») - permette la creazione di una sequenza di movimenti base, che devono essere eseguiti in ordine, da sinistra verso destra;

Parallel (|||) - stabilisce il rilevamento contemporaneo di due o più movimenti;

Disabling ([>) - ferma la rilevazione di un'altra gesture, utilizzabile per bloccare un ciclo iterativo;

Order Independence (| = |) - permette di collegare sub-gesture ed eseguirle in maniera casuale senza un preciso ordine; il movimento è considerato completato quando tutte le sub-gesture sono state eseguite.

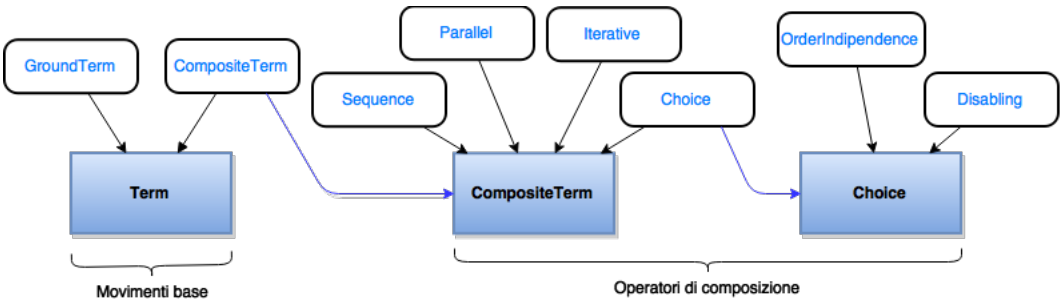

Figure 3.1: Gerarchia dei Term e degli Operatori.

Caratteristica principale di questi operatori è che devono avere almeno un figlio, ognuno dei quali può essere un *ground term* o un altro *composite term*.

Attraverso la combinazione di questi due elementi è possibile definire quindi una gesture. A titolo di esempio prendiamo in considerazione la gesture *Grab* eseguita tramite la mano destra. Dalla nostra definizione questa gesture viene rappresenta come la relazione di tre movimenti più piccoli: **StartCloseHand**$_r$ $\gg$ **(MoveHand**$^*_r$ **[> EndOpenHand**$_r$**)**, dove *StartCloseHand*$_r$ rappresenta il gesto di chiusura della mano, *MoveHand*$^*_r$ un movimento iterativo della mano che si conclude quando l'utente riapre la mano (definito da *EndOpenHand*$_r$). Nel prossimo capitolo analizzeremo più nel dettaglio la loro implementazione e il loro comportamento.

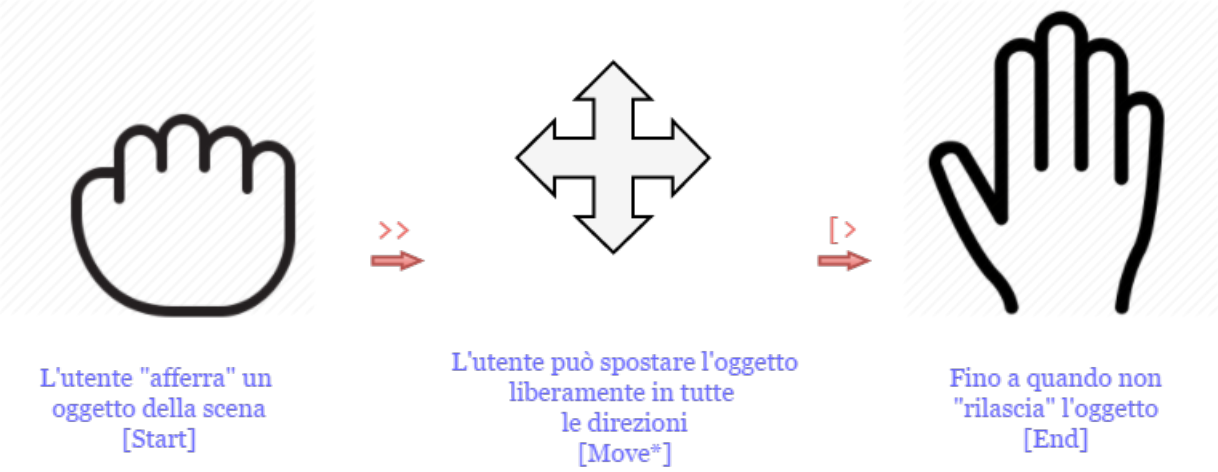

Figure 3.2: Nell'immagine sono riportati i movimenti che deve compiere l'utente per eseguire la gesture *Grab*.

3.1.1 Ambiguity

Questo tipo di definizione introduce però il problema della *selection ambiguity*, cioè situazioni in cui più gesture sono riconosciute contemporaneamente. Questo si verifica quando delle gesture hanno delle componenti (dei prefissi) uguali all'interno delle loro espressioni. Prendiamo come esempio le due gesture *Move* e *Rotate* che possono descrivere rispettivamente lo spostamento orizzontale/verticale lungo una scena e la rotazione di quest'ultima. Nella pratica però le loro due espressioni ricalcheranno esattamente la definizione della gesture *Grab*, infatti una loro definizione può essere la seguente:

- $Move = StartCloseHand_r \gg (MoveHand_r^* \ [> \ EndOpenHand_r)$
- $Rotate = (StartCloseHand_r \ || \ StartCloseHand_l) \gg ((MoveHand_r^* \ || \ MoveHand_l^*) \ [> \ (EndOpenHand_r \ || \ EndOpenHand_l))$.

Possiamo notare che se l'utente utilizza la mano destra per eseguire il movimento o la rotazione, vengono riconosciute all'inizio entrambe: infatti sia *Move* che *Rotate* hanno la stessa definizione di *Start*; per evitare che entrambe le gesture vengano riconosciute occorre descrivere in maniera dettagliata il movimento che dovrà eseguire la mano nel term $MoveHand_r$; per esempio la $MoveHand_r$ di *Rotate* prevederà un movimento più circolare rispetto a quello della *Move*, che sarà invece più improntata su un movimento verticale o orizzontale. *GestIT* permette di gestire questo problema tramite l'operatore *Choice*, che consente di valutare lo stato di più gesture contemporaneamente, senza effettuare una scelta immediata.
Tuttavia occorre gestire anche la possibilità che ai term che descrivono la gesture siano associate delle operazioni. Infatti è possibile che l'interfaccia debba reagire immediatamente al completamento di un certo *Ground Term* che compone una gesture, con la necessità di fornire dei feedback immediati. Riprendendo l'esempio precedente, al riconoscimento del movimento descritto da $StartCloseHand_r$, in entrambi i gesti, i term che li descrivono potrebbe prevedere dei cambiamenti nell'interfaccia; e questo ovviamente rappresenta un problema, perché l'interfaccia dovrebbe "adattarsi" contemporaneamente a due operazioni diverse, che potrebbero risultare essere anche in

conflitto. Inoltre occorre tenere in conto che al passo successivo l'effetto di una delle gesture parzialmente riconosciute dovranno essere poi cancellati dall'interfaccia, in seguito all'evolversi del movimento dell'utente.
L'obbiettivo del nostro modello è quello di gestire questo tipo di input: durante l'interazione si tengono conto di tutte le modifiche previste dalle gesture parzialmente riconosciute e dai term che le compongono. Queste operazioni però non devono agire immediatamente sull'interfaccia, ma piuttosto devono essere prima comunicati all'utente, e solo al completamento di una gesture i suoi effetti verranno poi eseguiti e visualizzati.
Come detto in precedenza, queste problematiche derivano dal fatto che l'input gestito può risultare ambiguo, a differenza dell'interazione con il mouse o la tastiera, dove non esiste questa ambiguità.

3.2 Feedback and Feedforward

Detto ciò i modelli di feedback e di feedforward che vogliamo realizzare devono rispettare alcuni punti:

- innanzitutto devono essere totalmente indipendenti dal tipo di applicazione su cui vengono utilizzati;
- devono essere indipendenti anche dal tipo di dispositivo di tracciamento utilizzato;
- infine queste informazioni devono essere comunicate all'utente durante tutto il periodo dell'interazione con il sistema.

La possibilità di utilizzare delle gesture dichiarative e composizionali ci permette di poter raccogliere in tempo reale tutte le informazioni circa lo stato delle gesture. Questi dati verranno poi utilizzati sia per informare l'utente di quali sono le gesture attualmente riconosciute, sia per determinare quale sarà lo stato dell'interfaccia in seguito all'esecuzione di una particolare gesture.

La realizzazione del modello di feedback da noi proposto si articola su questi fattori:

- La creazione di un albero che rappresenti tutti gli stati che può assumere l'interfaccia durante l'interazione dell'utente con il sistema; in questo albero ogni stato è figlio della radice, rappresenta una gesture e può essere descritto a sua volta come un albero. Nella pratica gli stati sono definiti da un'insieme di variabili, che rappresentano quelle parti dell'interfaccia potenzialmente modificabili al completamento di una gesture o di un *Ground Term*. Ad ogni possibile stato viene associato quindi:
  - il suo handler;
  - e la gesture da cui deriva;
- A runtime, a partire dall'insieme di tutti gli stati ottenibili dall'esecuzione delle diverse gesture, è possibile quindi determinare quali sono quelle modifiche che possono generare dei conflitti e quali no. Nel modello da noi definito utilizziamo un secondo albero in cui vengono memorizzati, sulla base delle gesture parzialmente riconosciute in quel momento, tutte le modifiche che non creano delle

incongruenze. Tale albero viene costruito a partire dal modello intero degli stati definito precedentemente, e ne rappresenta un sottoinsieme. Durante l'interazione il sistema, attraverso questo albero, si occupa di fondere in un unico stato tutte le possibili modifiche che potrebbero essere eseguite.
Un fattore importante che la nostra architettura deve saper gestire sono i problemi legati alla ambiguità della selezione, ovvero la possibilità che più gesture possano essere riconosciute contemporaneamente. Dal punto di vista dei feedback ciò rappresenta un problema, in quanto diventa difficile poter dare le necessarie informazioni all'utente; infatti se le gesture riconosciute lavorano sugli stessi elementi dell'interfaccia, eseguendo azioni diverse, si potrebbero formare dei conflitti o delle incongruenze complicando l'esperienza dell'utente.

Nella pratica questo albero è ottenuto inserendo gli handler associati alle gesture attualmente in esecuzione e, per ognuno di essi, l'elenco delle variabili su cui operano. Questo elenco può essere oggetto di continue modifiche durante l'esecuzione:

- Quando una gesture risulta parzialmente riconosciuta, si inserisce il suo handler nell'albero. Contemporaneamente si modificano l'elenco delle variabili degli handler già presenti nell'albero (cioè delle altre eventuali gesture parzialmente riconosciute) e dell'handler inserito, in maniera tale da evitare conflitti. Si tratta in sostanza di eseguire operazioni di differenza tra l'insieme di variabili dell'handler presente e quello dell'handler da inserire;
- Quando non è più possibile continuare con il riconoscimento di una gesture, l'handler associato viene rimosso dall'albero. Di conseguenza si provvede ad aggiornare l'elenco degli stati dell'interfaccia ancora presenti, inserendovi gli eventuali elementi che avevano in comune solo con l'handler rimosso (ma non anche con gli altri handler).

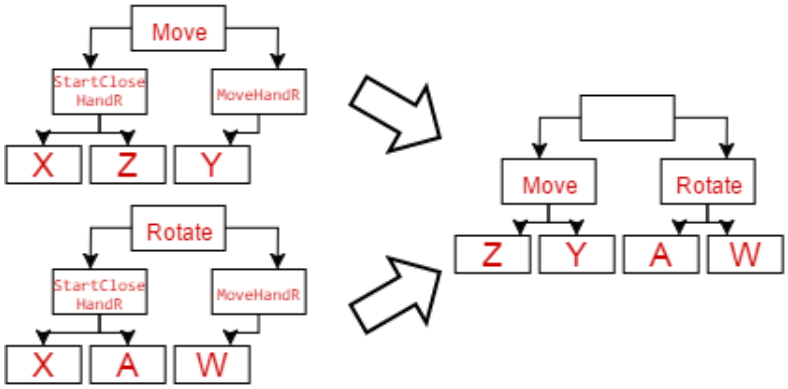

Figure 3.3: A partire dalla definizione degli stati ottenibili dall'esecuzione di una gesture, si passa a un unico albero che fonde gli stati delle gesture riconosciute attualmente, in grado di gestire anche gli eventuali conflitti. Questo è l'albero costruito a partire dalle due gesture d'esempio viste poco prima.

Ogni qualvolta che tale albero viene aggiornato si provvede a comunicare i dati raccolti al sistema. Tali dati potranno poi essere utilizzati dallo sviluppatore per costruire dei feedback e dei feedforward più efficienti ed efficaci, che aiutino

l'utente nella sua interazione. Un esempio può essere quello di visualizzare i vari stati dell'interfaccia che l'utente può ottenere in base alle gesture parzialmente riconosciute in quel momento.

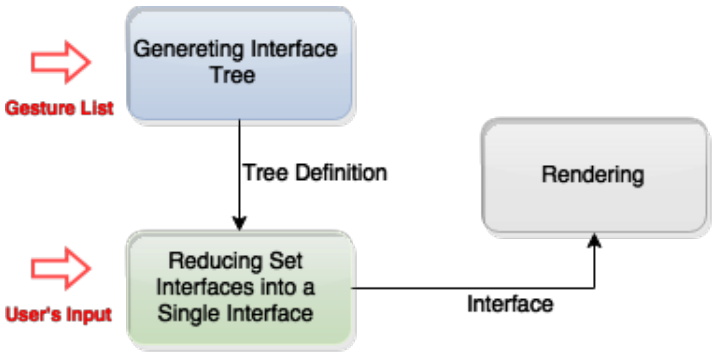

Figure 3.4:

Per sviluppare i modelli da noi definiti, abbiamo realizzato una libreria che metta a disposizione tutti gli strumenti per poter definire delle gesture dichiarative, generare dei feedback e dei feedforward continui e che consenta la gestione in maniera efficace, efficiente e precisa dell'acquisizione di dati dal dispositivo, agevoli l'utente nella creazioni di applicazioni che prevedono l'uso delle funzionalità appena descritte. Vista la necessità di poter riutilizzare il codice in diversi ambiti e con diverse tipologie di dispositivi d'acquisizione, si è scelto di creare una Dynamic Link Library, di cui andremo ad analizzare i dettagli implementativi nel prossimo capitolo.
La libreria è stata scritta in *C#*, ed è stata sviluppata fino alla parte legata ai feedback e feedforward in collaborazione con la collega *Isadora Sanna*.

Chapter

4
LIBRARY

Ora andremo ad analizzare più nel dettaglio la libreria sviluppata e che implementa i modelli discussi nel capitolo precedente. La libreria è caratterizzata da tre componenti fondamentali:

Gesture Recognition : è la componente che implementa il sistema per la definizione compositiva di nuove gesture e per il loro riconoscimento. Il sistema in se è totalmente indipendente dal dispositivo di riconoscimento utilizzato, tuttavia per poter definire delle gesture occorre utilizzare i dati che descrivono i movimenti rilevati dal dispositivo. Attualmente la libreria supporta sia la definizione di movimenti della *Kinect One* che quella del *Leap Motion*.

Generating feedback and feedforward : è invece la componente che implementa il sistema di generazione dei feedback e dei feedforward continui a partire da un database di gesture.

Other Utilities : è l'elemento della libreria che offre delle funzionalità di ausilio nella programmazione con la *Kinect* e il *Leap Motion*, come la gestione e la stampa dei dati acquisiti dai due dispositivi;

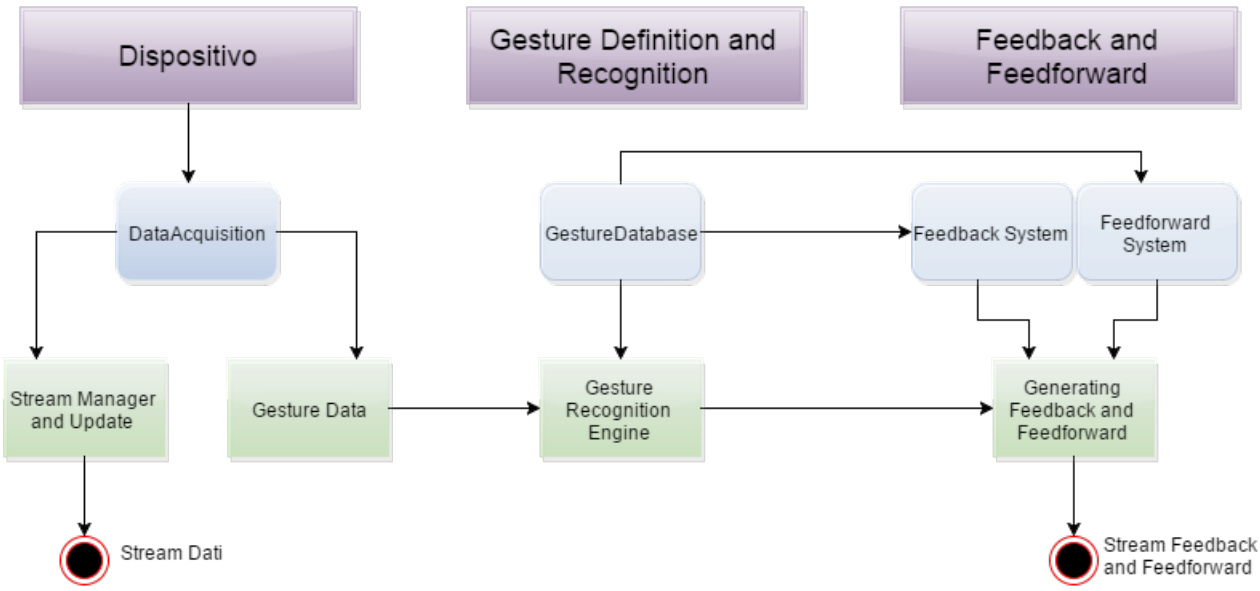

Figure 4.1: Una panoramica ad alto livello della libreria.

4.1 Gesture Recognition

La prima componente che andiamo ad esaminare è quella relativa alla creazione e al riconoscimento di nuove gesture. Come già accennato in precedenza, per la realizzazione di questa componente abbiamo implementato il framework *Gestit*, adattandolo al linguaggio *C#* e alle altre componenti della libreria, e aggiungendovi nuovi elementi. Inoltre accanto all'implementazione del framework *GestIT* abbiamo messo a punto anche degli strumenti per il riconoscimento di gesture basato sul tool *VisualGestureBuilder*, strumento distribuito assieme all'SDK della *Kinect One*. Mentre il sistema basato su *Gestit* è completamente indipendente dal dispositivo utilizzato, il secondo è invece rivolto solo alla *Kinect One*, e abbiamo deciso di implementare questo sistema per poter meglio analizzare e studiare la differenza di resa tra le due metodologie.

4.1.1 Gestit

Come abbiamo visto in precedenza 2.3.1, *GestIT* rappresenta un modello dichiarativo per la creazione e il riconoscimento di nuove gesture. In tale sistema una gesture è un'espressione costruita sulla base di due *term*: i *Ground Term* e i *Composite Term*.

Term

Tutte le componenti che descrivono una gesture (*ground* e *composite term*) sono rappresentate nella libreria attraverso una classe apposita, e ciascuna classe deriva da un'unica superclasse chiamata *Term*. La classe *Term* definisce gli attributi e i metodi principali di ogni componente, in particolare:

state : rappresenta lo stato del term, e può assumere tre valori rappresentati dall'enum *ExpressionState*:

Complete (= 1) sta ad indicare che il movimento che descrive il term è stato eseguito dall'utente, quando ciò succede il sistema provvede a generare un evento.

Error (= -1) quando il movimento eseguito dall'utente non coincide con la definizione prevista dal term, e anche stavolta, quando ciò succede, il sistema provvede a generare un evento.

Default (= 0) quando il term è in attesa di poter esaminare il token che descrive il movimento eseguito dall'utente o quando è in esecuzione.

excluded e once : sono due booleani che, come vedremo più avanti, vengono utilizzati durante la fase di riconoscimento;

num_total e num_discrete : il primo rappresenta il numero di volte che una gesture (o una parte di essa) è stata eseguita dall'avvio del programma, il secondo invece rappresenta il numero di volte che una gesture (o parte di essa) è stata eseguita in maniera continuata;

lookahead : è la funzione su cui si basa il riconoscimento di una gesture. In sostanza provvede ad esaminare se il term è stato eseguito (e quindi se si trova in uno stato di complete) oppure no, e nel caso comunicare il cambiamento di stato tramite un evento. Nei *ground term* questa funzione verifica se la definizione del movimento contenuta al suo interno corrisponde con quello effettivamente eseguito dall'utente. Nei *composite term* invece, tale funzione può avere definizioni diverse a seconda delle peculiarità dell'operatore preso in considerazione, tuttavia lo scopo è quello di verificare quale sarà lo stato dei suoi figli analizzando gli ultimi dati in arrivo.

fire : anche questa funzione viene utilizzata nel riconoscimento di una gesture; la funzione *fire* viene utilizzata dai *composite term* per verificare se il movimento o la gesture che descrivono è stata eseguita nell'ultimo *token* inviato dal sistema. Se durante l'analisi di un'espressione lo stato di uno dei suoi termini va in Error, le informazioni acquisite sui movimenti per quella gesture fino a quel momento vengono resettati e sarà necessario eseguirla da capo.

Di seguito andremo ad analizzare nel dettaglio come vengono definite e riconosciute delle gesture.

Defining

Una nuova gesture viene definita, come accennato in precedenza, tramite dei *ground term* messi in relazione attraverso dei *composite term*. I *ground term* sono dotati, a differenza dei *composite term*, di due funzioni (*accept* e *_accept*) che specificano come dev'essere eseguito il movimento che descrivono. Queste due funzioni devono essere definite dal programmatore nel momento in cui crea la gesture.
Per dare una dimostrazione di come funziona il sistema di creazione delle gesture, vedremo un semplice esempio in cui andremo a definire il movimento di Pan lungo l'asse X; la parte di corpo presa in considerazione sarà la mano destra.

Il gesto Pan può essere suddiviso in tre parti:

- Start: chiusura della mano destra;
- Move: uno spostamento continuo della mano destra in linea orizzontale sull'asse delle X;
- End: apertura della mano destra;

Per mettere in relazione questi tre *ground term* vengono utilizzati tre operatori:

- Iterative: Il suo unico figlio sarà rappresentato dal *ground term* che rappresenta il Move. Questo operatore permette all'utente di spostare la mano lungo l'asse X per quanto vuole fino a quando non riaprirà la mano.
- Disabling: questo operatore ha come figli l'*Iterative* definito in precedenza e il *ground term* di tipo End. Attraverso questo operatore è possibile completare la gesture aprendo la mano destra durante il movimento orizzontale; In sostanza permette di sbloccare il movimento descritto dall'*Iterative*.
- Sequence: rappresenta invece la radice dell'albero, e i suoi figli sono il *ground term* Start e il *composite term Disabling*. Questo operatore prevede un esecuzione sequenziale dei suoi operatori. Quindi prima dovrà essere eseguito il movimento descritto da Start, quindi dovrà poi essere completato il movimento rappresentato dal *Disabling*.

E l'espressione che la rappresenta può essere scritta in maniera formale nel seguente modo: **Start** $\gg$ **(Move*** **[> End)**.

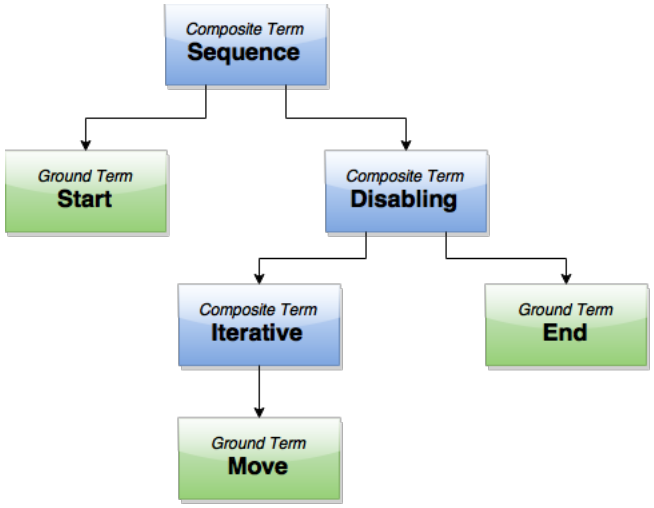

Figure 4.2: L'albero che descrive la gesture Pan.

Recognition Nella fase di riconoscimento vengono utilizzati, come accennato in precedenza, i dati rilevati dal dispositivo di tracciamento e la classe *Sensor*; tale classe ha il compito di creare i token e di gestire la lista che contiene tutte le gesture rilevabili; questa lista viene solitamente rappresentata da un *composite term* che ha come figli le espressione che definisco le gesture.
Dai dati ricevuti dal dispositivo, il sistema costruisce i *token*, oggetti che contengono le informazioni basilari sui corpi e gli elementi rilevati, ad esempio lo scheletro di un utente nel caso della *Kinect*.
Anche i *token* possono essere classificati in tre tipologie diverse: Start, Move o End, a seconda che l'oggetto che descrive sia stato appena identificato, che fosse già presente nell'elenco degli oggetti tracciati o che non venga più rilevato dal dispositivo; per facilitare questa operazione di verifica ogni oggetto è dotato di un ID. La classe *StateSequence*, invece, ha il compito di gestire i token ricevuti e un dizionario. In questo dizionario, per ogni oggetto riconosciuto (e ancora tracciato), è associato un buffer circolare in cui vengono memorizzati gli ultimi *n* token creati (e che riguardavano tale oggetto). L'associazione all'oggetto, e a suoi ultimi token, vengono eliminati dal dizionario nel momento in cui viene ricevuto un token di tipo End.

Una volta costruito e classificato il token, inizia la fase di riconoscimento vero e proprio e questo avviene sfruttando il meccanismo della visita in-order di un albero. Per farlo il sensore invia il *token* all'operatore che raccoglie tutte le gesture, chiamando la funzione *fire* di quest'ultimo. Il comportamento di tale funzione non varia molto a seconda del *composite term* preso in considerazione (se non l'ordine con cui si verifica lo stato dei figli con il nuovo *token*), e il suo funzionamento è molto semplice come si può notare dal seguente pseudo-codice della funzione *fire*:

Algorithm 1: *Fire* function.

```
   Data: token
 1 int index ;                // l'indice utilizzato dal term per accedere ai suoi figli
 2 ;
 3 if Si controlla se il nuovo token rispetta la condizione descritta dal term then
 4 |   children[index].fire(token);
 5 else
 6 |   Si pone in error il term e i suoi figli;
 7 |   return;
 8 if index > children.Count then
 9 |   Si pone in error il term e i suoi figli;
10 |   return;
11 else
12 if children[index].state == Complete then
13 |   index++;
14 |   if index == children[index].state then
15 |   |   La gesture è stata eseguita correttamente;
16 |   else
17 else
18 Lancia un evento per comunicare l'avvenuta gestione di un token;
19 return;
```

Per comprendere meglio il funzionamento della fase di riconoscimento, facciamo un piccolo esempio ripartendo dalla gesture 'PanX' che abbiamo definito poc'anzi. Una volta creato il *token*, questo viene sparato al *Composite Term* più esterno che fa da radice all'albero, in questo caso il *Sequence*. Quest'ultimo richiama a sua volta la funzione *lookahead* sul suo primo figlio, ovvero il *ground term* che rappresenta il movimento di Start. Se i dati contenuti nel *token* sono coerenti con la sua definizione del movimento, allora la funzione *lookahead* restituirà true e il *ground term* viene posto in uno stato di Complete. Di conseguenza Il successivo *token* in arrivo verrà confrontato con il suo secondo figlio, ovvero il *Disabling*; non essendo un termine finale, una chiamata alla sua funzione *lookahead* comporta un'altra chiamata alla stessa funzione, stavolta, del primo figlio *Iterative* che a sua volta richiama la *lookahead* del *ground term* di tipo Move. Cosi facendo, nel momento in cui tutti i figli di *Sequence* verranno impostati a Complete, il gesto potrà essere segnalato come eseguito. In caso contrario, *Sequence* sarà nello stato Error e verrà resettata.
Le due variabili *excluded* e *once* vengono utilizzate durante le chiamate alla funzione *lookahead*.

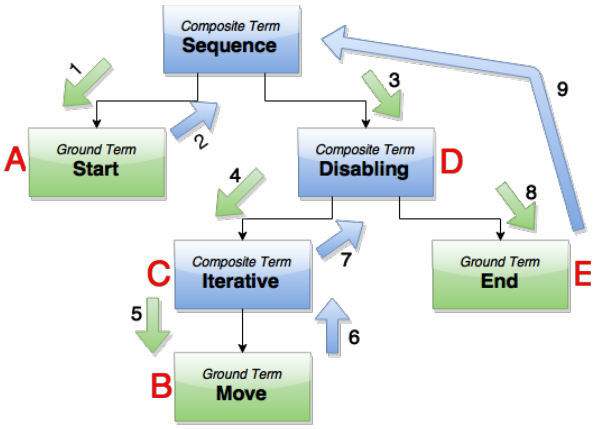

Figure 4.3: Ordine in cui vengono valutati e gestiti i Term che compongono l'espressione.

4.1.2 VisualGestureBuilder

L'altro strumento di riconoscimento delle gesture presente nella libreria è dedicato interamente alla *Kinect One*, ed è basato sulla componente *VisualGestureBuilder* presente nell'SDK del dispositivo. Sia la creazione che la definizione delle gesture avviene in utilizzando dei classificatori (ad esempio l'*HMM*, le reti bayesiane ecc.).
A partire dai *body* tracciati dalla *Kinect*, si provvede a creare un oggetto *GestureDetector* per ogni corpo rilevato. La classe *GestureDetector* si occupa, a partire da un database di gesture in formato *vgb*, di controllare all'arrivo di un frame se si è verificato o meno una gesture, e nel caso comunicare (tramite evento) quale gesture è stata eseguita e da quale utente. La *Kinect* distingue due tipi di gesture, quelle discrete e quelle continue. Mentre per le prime la classe *VisualGestureBuilder* permette di conoscere, per ogni gesture definita, il livello di confidenza, per quelle continue mette a disposizione anche informazioni riguardanti il livello di progressione delle stesse. Il sistema provvede a generare degli eventi per comunicare il riconoscimento di una gesture:

- L'evento *DiscreteGestureExecute* comunica quale gesture dinamica è stata eseguita; i suoi parametri sono la gesture riconosciuta, rappresentata dalla classe *Gesture*, e l'attributo *DiscreteGestureResult* generato dall'SDK della *Kinect*.
- L'evento *ContinuousGestureExecute* comunica invece il riconoscimento di una gesture continua. In questo caso i parametri sono la gesture rilevata e l'attributo ContinuousGestureResult.

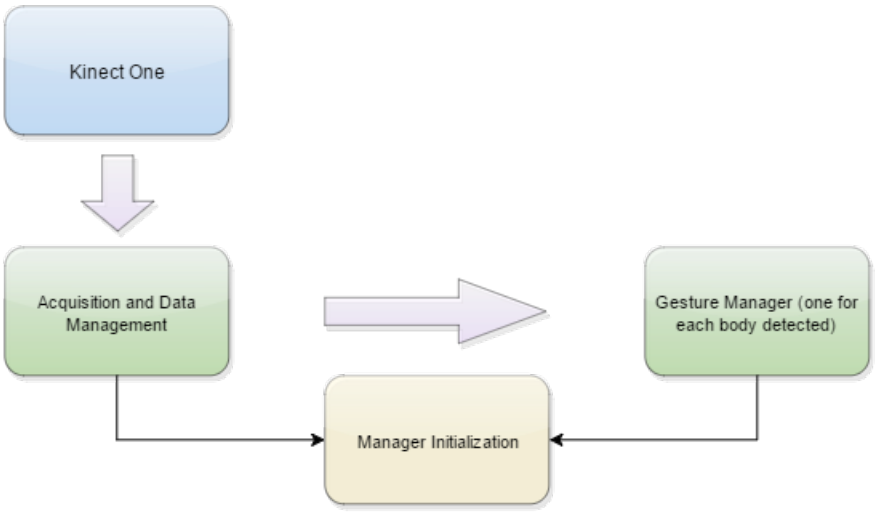

Figure 4.4: Schema di comunicazione dei dati tra il dispositivo e il sistema di rilevamento delle gesture.

4.2 Generating Feedback and Feedforward

Andiamo ora ad analizzare la componente che implementa i due modelli di generazione dei feedback e dei feedforward.
Per lo sviluppo del modello di generazione dei feedback ci siamo basati sull'architettura per la generazione di feedback interattivi[45] presentata a *CHI'15*, adattandolo al nostro contesto; il nostro modello implementa quindi l'idea di base di tale architettura in un contesto differente, ovvero in un sistema con cui è possibile interfacciarsi tramite delle gesture definite attraverso gli strumenti messi a disposizione dalla nostra libreria.

Lo sviluppo del modello di generazione dei feedforward, svolto dalla collega Isadora Sanna nell'ambito della sua tesi, si basa invece sul sistema *OctoPocus*[43]; anche in questo caso è stato necessario adattare l'idea di base di *OctoPocus* ad un'interazione più naturale rispetto all'utilizzo di mouse o di digital pen.

4.2.1 Feedback

Abbiamo visto in astratto quali sono gli obiettivi che vogliamo raggiungere con il nostro modello di feedback. Dal punto di vista pratico gli alberi e i feedback vengono realizzati in questo modo:

- l'albero che riporta tutte le gesture in esecuzione è realizzato attraverso un dizionario (o mappa), dove la chiavi sono rappresentate dagli handler delle gesture riconosciute e i valori, invece, rappresentano le variabili personalizzabili del programma su cui andranno poi ad agire.

- come già accennato in precedenza, le informazioni raccolte a partire dall'input dell'utente e dagli alberi mantenuti dal sistema vengono utilizzati per creare i feedback. Questi feedback solitamente vengono visualizzati su schermo, permettendo all'utente di conoscere sia quali saranno gli effetti sul programma in base alla gesture che sta eseguendo, sia vedere come il suo movimento è stato riconosciuto fino a quel momento dal sistema.

Tree

All'interno del sistema, l'albero che descrive tutte le interfacce viene costruito a partire dall'insieme di gesture rilevabili dal programma. La classe *FeedbackTree* della libreria prende in un input un *composite term*, che descrive queste gesture, e provvede a inizializzare la radice di tale albero.
La radice è rappresentata dalla classe *FeedbackRoot*, e ha un figlio per ogni gesture passato in input. Le gesture sono quindi un nodo dell'albero, ognuno definito dalla classe *FeedbackGesture*; a sua volta i nodi che rappresentano le singole gesture possono avere due tipi di figli:

- i *FeedbackLeafGround*, associati ai singoli *ground term* che compongono la gesture;

- i *FeedbackLeafComposite*, associati invece ai singoli *composite term* che intervengono nella definizione della gesture.

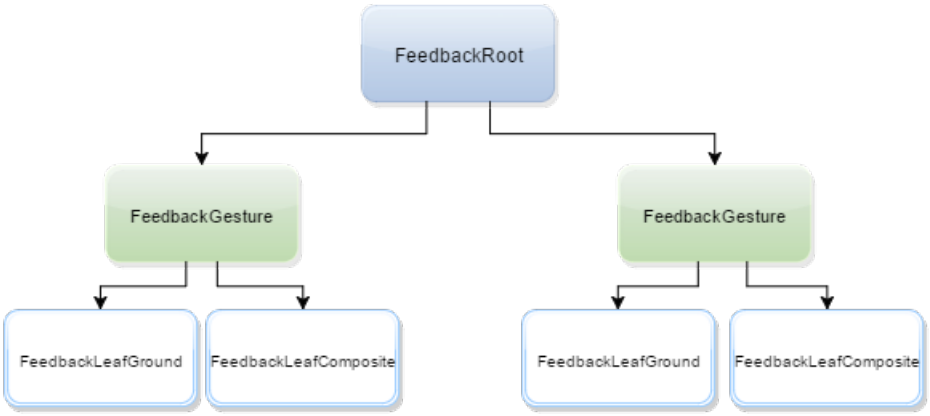

Figure 4.5: Tale immagine rappresenta lo schema logico con cui una lista di gesture vengono descritte tramite l'albero generato dal sistema dei feedback.

Per comprendere meglio come viene creato l'albero a partire da una lista di gesture vediamo un esempio pratico. Il sistema riceve in input un *composite term* che contiene al suo interno due gesture, il PanX e il PanY; l'espressione di quest'ultimo è uguale a quella che definisce il PanX, con la sola differenza che il Move controlla se il movimento avviene lungo l'asse Y dall'alto verso il basso. Il costruttore della radice dell'albero prende in input il term che contiene le gesture, e a partire dalla lista di figli dell'operatore provvede a creare un nodo per ogni gesture definita; in questo caso avremo quindi due *FeedbackGesture*. A loro volta questi due nodi vanno ad attraversare la loro rispettiva espressione per creare tutti i loro rispettivi nodi figli; entrambi avranno quindi cinque figli, di cui tre *FeedbackLeafGround* (relativi ai term di Start, Move ed End) e di due *FeedbackLeafComposite* (associati ai due term *Interative* e *Disabling*). I *FeedbackLeafGround* e i *FeedbackLeafComposite* rappresentano perciò le

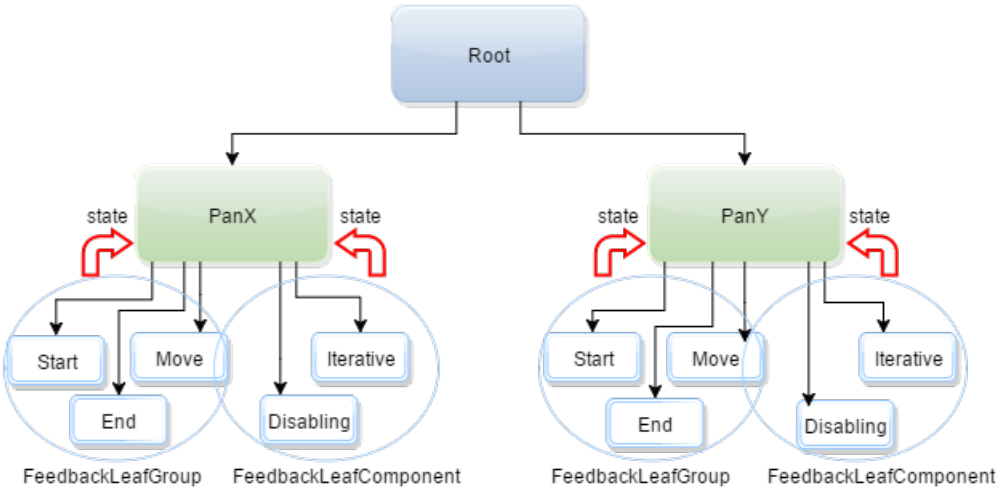

Figure 4.6: L'albero che descrive le due gesture.

foglie dell'albero di cui il *FeedbackRoot* è la radice, e gli attributi di queste due classi sono:

term : che rappresenta l'espressione da cui deriva il nodo;

state : è un flag che indica lo stato del term associato al nodo. Gli stati possibili sono quattro, ai tre stati che abbiamo già visto per il *term* (Complete, Default ed Error) si aggiunge anche lo stato Continue, che sta ad indicare un movimento non ancora completato (e potenzialmente eseguibile).

likelihood : è un float che rappresenta la probabilità di esecuzione associata al term. Il sistema assegna un valore di probabilità solo ai *ground term*, mentre la probabilità degli operatori viene calcolata a partire da un modello statistico da noi definito.

handler : rappresenta in sostanza la funzione che viene eseguita nel caso in cui il term raggiunga lo stato di Complete. Gli handler sono rappresentati nella libreria dall'omonima classe, e gli attributi di questa classe sono la funzione associata all'esecuzione della term e la lista di *custom attributes* del programma su cui agirà la funzione.

Likelihood

Per poter calcolare la probabilità di ogni handler è stato necessario aggiungere nella classe *Term* un float che rappresenta la probabilità che quella singola espressione venga eseguita.

- gli eventi del modello sono rappresentati dai *ground term*;
- ogni evento è indipendente; ciò significa, per esempio, che la probabilità di muovere la mano verso sinistra non è influenzata da nessun altro evento;
- gli eventi possono essere compatibili o incompatibili, una mano può essere chiusa o aperta, un braccio può muoversi verso destra o sinistra per esempio;

In questo modello abbiamo deciso che la probabilità dei singoli *ground term* dev'essere assegnata di default dal programmatore (usando magari i dati di test o prove precedenti); per quanto riguarda i *composite term*, invece, il sistema mette a disposizione una serie di metodi per calcolarne la probabilità. Questi metodi sono implementati dalla classe *ComputeLikelihood*.
In questo modello la probabilità viene calcolata a partire dal tipo di *composite* in questione, dalla sua definizione e dai figli che lo compongono; di conseguenza, nella maggior parte dei casi, la probabilità di ogni *composite term* è definita dalla probabilità composta degli eventi che lo compongono con piccole differenze a seconda dell'operatore in oggetto:

Sequence, Disabling and Parallel : poiché ogni evento è indipendente e in questi operatori i figli di devono essere eseguiti in sequenza o contemporaneamente, allora la loro probabilità è data semplicemente dal prodotto della probabilità di tutti i suoi figli.

Order Independece : in questo caso, la sua probabilità è data dal numero di figli che lo compongono moltiplicato per il prodotto della probabilità composta di quest'ultimi.

Iterative : la probabilità di questo operatore è ottenuta moltiplicando la probabilità del suo figlio per una certa costante; questa costante rappresenta la lunghezza media di un movimento di tipo iterative.

Choice : come visto in precedenza al punto 3.1, l'operatore *Choice* permette l'esecuzione di uno o più movimenti; in questo caso il valore della sua probabilità viene associato al valore di probabilità più alto dei *term* che ha come figli.

Per questioni di tempo non è stato possibile approfondire lo studio sulla gestione della probabilità. Un possibile modello che abbiamo intenzione di sviluppare, e di cui parleremo in dettaglio più avanti, è quello basato sulle catene di Markov. Questo è un modello molto affidabile, in quanto ci permetterebbe di calcolare la probabilità di passare da un evento a tutti gli altri eventi (tenendo quindi conto della loro compatibilità o incompatibilità).

Generating Feedback

La radice si occupa, durante l'esecuzione, di tenere traccia di quali sono le gesture che si trovano nello stato di Continue e quali nello stato di Error o Complete. I nodi di tipo *FeedbackGesture* aggiornano automaticamente, a tempo di esecuzione, il loro stato in base al cambiamento dello stato di una delle sue sotto-componenti. Un nodo si trova nello stato Continue dal momento in cui viene soddisfatto il suo *ground term* di Start fino a quando una delle sub-gesture non va in uno stato di Error o di Complete.

La classe *FeedbackRoot* ha anche un altro compito, quello di costruire a tempo di esecuzione, una *mappa* che riporti per ogni gesture nello stato Continue il suo handler e la lista degli *custom attributes* su cui opera. Questo permette al sistema di poter indicare all'utente le modifiche allo stato del programma in tempo reale; per evitare eventuali conflitti durante la fase di visualizzazione dei feedback all'utente, il modello definito prevede che per ogni handler presente siano riportati solo quei *custom attributes* non utilizzati da altri handler presenti nella *map*.
Il metodo che si occupa dell'aggiornamento della struttura può essere rappresentato meglio attraverso un algoritmo in pseudo-codice:

Algorithm 2: *FeedBackGesture* Update

Data: FeedbackGestureEvent, che comunica quando cambia lo stato di un FeedbackGesture

if *feedbackGesture[index].state == Continue && il suo handler non è presente nella dictionary* **then**
    Inserisce l'handler all'interno della struttura, e per ogni handler della map rimuove gli attributi personalizzabili in comune;
else
if *feedbackGesture[index].state == Error && il suo handler è presente nella dictionary* **then**
    Rimuove l'handler dalla struttura e aggiorna la lista dei custom attributes dei rimanenti handler. (vengono rimessi gli attributi che gli handler avevano in comune solo con l'handler rimosso);
else
if *feedbackGesture[index].state == Complete* **then**
    Viene svuotata completamente la map, in attese di un nuovo movimento da parte dell'utente;
else

A tempo d'esecuzione la libreria si occupa di inviare queste informazioni al programma, che verranno poi visualizzate, per esempio, all'utente attraverso delle piccole finestre in alto sulla sinistra dello schermo. Spetta al programmatore decidere come visualizzare le informazioni raccolte dalla libreria. Ad ogni modo, utilizzando questo tipo interfaccia, se ci sono più di tre *FeedbackGesture* nello stato di Continue, il sistema provvederà a visualizzare i cambiamenti delle sole gesture con la probabilità più alta di essere eseguita.

4.2.2 Feedforward

Per quanto riguarda la realizzazione del sistema di feedforward sono state apportate delle modifiche minimali alle classi gia esistenti nella libreria e sono state aggiunte due nuove classi che agevoleranno il sistema di apprendimento delle gesture.

Gesture Track

L'idea di base su cui poggia il modello da noi definito è quello di creare un sistema che dia la possibilità, data la definizione di un gesto, di poterne disegnare a video la traccia che rappresenti i movimenti corretti da eseguire; Questo sistema risulta indipendente dal tipo di piattaforma o applicazione in cui si desidera di implementare la libreria.

Il sistema più semplice per suddividere una traccia è risultato essere quello della suddivisione in segmenti, e più saranno i segmenti che compongono la traccia più questa sarà precisa, in particolare nei tratti curvi. La creazione e la gestione di questi segmenti sono rappresentati nella libreria dalla classe *Segment*. Ogni segmento è caratterizzato dalla lunghezza e dall'angolo, espresso in gradi, che il segmento forma con l'asse delle ascisse. Tra gli attributi non sono state inserite le informazioni riguardanti il punto iniziale e finale del segmento, in quanto possono cambiare a seconda del sistema di riferimento in cui verranno disegnati. Per sistema di riferimento si intende il punto da cui l'ambiente utilizzato calcola il punto di origine degli assi. Ad ogni modo, una volta noto il sistema di riferimento, il punto di partenza e di fine del segmento verranno calcolati utilizzando le informazioni già impossesso, ovvero lunghezza e angolo.

La componente dei feedforward si basa anche sulla classe *GestureSegment*; questa classe è composta da un dizionario avente come chiave il nome del gesto, e come valore la lista di segmenti che compongono la traccia del gesto; questo permette al sistema di poter memorizzare più gesti e individuarli in modo univoco tramite il loro nome.
Per ogni *Ground Term* appartenente ad una gesture è possibile associare più di un segmento dato che, ad esempio, un *Ground Term* definito da un gesto curvo non può essere descritto con solo un segmento ma ne serviranno diversi con differenti angoli.

```
1 public GestureSegments gestureSegments = new GestureSegments();
2 public Segment tempSegment;
3
4 // Start
5 GroundTerm term1 = new GroundTerm();
6 term1.type = "Start";
7 //Definisco un segmento senza lunghezza che definisce il gesto discreto
    iniziale
8 tempSegment = new Segment(0, 0);
```

```
9  gestureSegments.addSegment("PanX", tempSegment);
10 term1.accepts = close;
11
12 // Move
13 GroundTerm term2 = new GroundTerm();
14 term2.type = "Move";
15 //Definisco che il movimento orizzontale di PanX e' suddiviso in 5 segmenti
       lunghi 0.4 l'uno e con angolo pari a zero.
16 for (int i = 0; i < 5; i++)
17 {
18     tempSegment = new Segment(0.4f, 0);
19     gestureSegments.addSegment("PanX", tempSegment);
20 }
21 term2.accepts = moveX;
22
23 // End
24 GroundTerm term3 = new GroundTerm();
25 term3.type = "End";
26 //Definisco un segmento senza lunghezza che definisce il gesto discreto
      finale
27 tempSegment = new Segment(0, 0);
28 gestureSegments.addSegment("PanX", tempSegment);
29 term3.accepts = open;
```

Listing 4.1: Aggiunta delle informazioni riguardanti i segmenti associati ai Ground Term

Facendo riferimento al comando for di riga 16, il *Ground Term* di PanX viene definito da cinque segmenti. In questo caso maggiore sarà il numero di segmenti più è preciso la rappresentazione grafica della traccia del movimento e i feedforward che verrano inviati all'utente durante l'esecuzione del gesto. In aggiunta di ciò il sistema prevede anche una colorazione interattiva dei segmenti come avviene nell'architettura *OctoPocus* fornendo anche dei meccanismi di feedback all'utente.

Unity

Durante lo sviluppo della libreria, si è deciso di implementare anche un'estensione per *Unity 5.0*. Il primo problema riscontrato è stato legato al vincolo creato dell'utilizzo di una versione obsoleta di Mono da parte di Unity 5. Questa versione consente l'utilizzo di versioni del *Framework .NET* pari o inferiori alla 3.5. A causa di questa carenza non è stato possibile innanzitutto poter integrare totalmente la libreria, e inoltre è stato necessario adattare alcune classi e rimuovere diverse funzionalità.

L'integrazione ha richiesto l'aggiunta di tre nuovi script:

Initializer : si occupa di inizializzare il dispositivo di tracciamento. Attualmente questa estensione supporta solo la *Kinect One*, tuttavia si prevede in futuro di supportare anche il *Leap Motion*. Al suo interno vengono anche definite le gesture accettate dall'applicativo. L'*Initializer* si occupa anche di inizializzare le altre due classi aggiungitive.

TrackPointer : gestisce le informazioni rielaborate dalla libreria sugli oggetti tracciati dal dispositivo. Nella demo realizzata l'interazione avviene tramite delle gesture eseguite con le mani; per questo motivo l'icona, che rappresenta il movimento

della mano dell'utente all'interno dell'interfaccia, raffigura una mano stilizzata che segue i movimenti dell'utente. L'icona cambia a seconda che la mano sia aperta o chiusa, in modo che l'utente possa avere un feedback in tempo reale relativamente al rilevamento dello stato della sua mano.

GestureDrawer : questa classe viene utilizzata per la stampa a video del tracciato che rappresenta le varie gesture. Prendendo come riferimento il dizionario definito nella classe *GestureSegment*, per ogni chiave presente crea un nuovo *GameObject* conenente al suo interno un *LineRenderer*. Il *LineRenderer* è un oggetto composto da un numero personalizzabile di punti i quali, dopo essere stati definiti, vengono uniti con una retta. A partire dalla lista di segmenti che compongono ogni gesture, di cui conosciamo lunghezza e angolo che formano con l'asse delle ascisse, è possibile calcolare il punto di origine e di fine dei segmenti facendo uso delle coordinate polari. Questi punti verranno utilizzati dal *LineRenderer* e uniti a tempo d'esecuzione dando l'effetto di un tracciato. Partendo da un origine fissa (0, 0, 0) le coordinate dei punti dei segmenti vengono calcolate come segue:

```
1  public Vector3 getNewPoint(Vector3 lastPoint, float angle, float
    length)
2  {
3      var x = length * Mathf.Cos(angle * Mathf.Deg2Rad);
4      var y = length * Mathf.Sin(angle * Mathf.Deg2Rad);
5      var newPosition = lastPoint;
6      newPosition.x += x;
7      newPosition.y += y;
8      return newPosition;
9  }
10
```

Listing 4.2: Calcolo del punto successivo della traccia tramite coordinate polari

4.2.3 Speech problems

I modelli di feedback e di feedforward che abbiamo implementato non sono però adatti per dare all'utente delle informazioni aggiuntive durante l'esecuzione di comandi vocali. Innanzitutto perché il sistema di feedback da noi implementato non è adatto a fornire delle informazioni durante l'esecuzione di azioni rapide. Infatti i comandi vocali sono degli input molto veloci, solitamente composta da due, al massimo da quattro parole; e questo rende molto problematico trovare un modo per poter visualizzare dei feedback e dei feedforward in tali sistemi.

Effettivamente in letteratura non esistono molti articoli che trattano di tale problematica. Uno dei più importanti è un articolo[48] di Ainsworth W.A. e Pratt S.R. che propone un modello per aiutare l'utente a correggere i suoi errori in un sistema di *speech recognition*. Questo modello consiste nel richiedere all'utente di ripetere quei comandi il cui riconoscimento è risultato essere ambiguo, fornendo al contempo delle informazioni circa gli errori riscontrati. Un altro articolo[**?**] importante è quello presentato da Norris D., McQueen J.M. e Cutler A. alla conferenza *Behavioral and Brain Sciences* nel 2000. Questo articolo in sostanza afferma che i top-down feedback non aiutano realmente l'utente nell'interazione vocale con un sistema, e per dimostrarlo va ad analizzare il coinvolgimento lessicale nel processo fonemico decisionale.

Eliminare del tutto i feedback o i feedforward non, è secondo noi, la strada maestra; infatti non tutti gli utenti hanno lo stesso grado di esperienza, e il non fornire loro nessun tipo di aiuto potrebbe scoraggiarli dall'utilizzare tali sistemi. Inoltre esistono comunque dei modelli di feedback e di feedforward che si potrebbero applicare nei sistemi di riconoscimento vocale, e che possono aiutare l'utente durante l'interazione.
Un esempio di feedback potrebbe essere quello di occupare una parte dell'interfaccia, a tempo di esecuzione, per visualizzare un elenco dei comandi vocali che il sistema riconosce e quali saranno i loro effetti sullo stato del programma.
Un esempio invece di feedforward potrebbe essere, molto banalmente, quello utilizzato nei karaoke almeno per i comandi formati da più parole. Dopo aver riconosciuto la prima parola, il sistema potrebbe indicare quali sono tutti i comandi che iniziano con quella parola, e man mano che l'utente completa l'input, evidenziare le parole riconosciute.
Bisogna considerare inoltre che il riconoscimento vocale trova impiego anche nel campo della riabilitazione ed educazione. Se l'utente non riceve dei feedback, delle informazioni sulle sue prestazioni, come può migliorare le sue prestazioni? Probabilmente in nessun modo.

4.3 Other Utilities

La terza componente della libreria è quella che si occupa di comunicare con i dispositivi di tracciamento e di elaborare le informazioni ricevute da questi. La libreria supporta i dispositivi *Kinect One* e il *Leap Motion*, fornendo tutti gli strumenti per l'acquisizione delle informazioni.

4.3.1 Kinect

La classe *InitKinect* si occupa di stabilire una connessione valida con il dispositivo. La classe *AcquisitionManager* invece ha il compito invece di smistare e gestire tutti i frame ricevuti, di inizializzare le strutture dati che conterranno le informazioni ricevute. L'interazione con la *Kinect* avviene sfruttando i metodi presenti nel *dll Microsoft.Kinect*. Le strutture dati sono rappresentate da classi apposite, che consentono alla libreria e a chi la utilizzerà di poter accedere in maniera rapida e semplice alla informazioni acquisite.

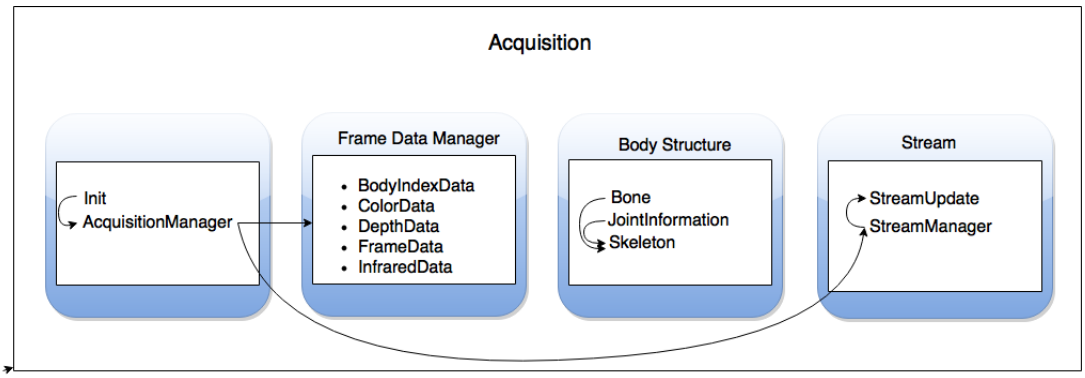

Figure 4.7: Schema che descrive la componente dedicata alla *Kinect One*.

In generale queste classi sono caratterizzate da degli array di tipo *byte* in cui vengono trasferiti i dati contenuti nei frame ricevuti dalla *Kinect 2.0*. E ciò vale per i frame di tipo *Color*, *Depth*, *Infrared* e *LongExposure*. Per quanto riguarda i corpi rilevati dal dispositivo la libreria prevede una classe più complessa, *Skeleton*, che non si limita a raccogliere i dati utilizzabili ma li rielabora in più modi. La libreria della *Kinect One* mette a disposizione la classe *Body* per descrivere i corpi tracciati e rilevati dal dispositivo. Tuttavia abbiamo deciso di implementare comunque una nuova classe per descrivere questi corpi, rimuovendo alcune informazioni ridondanti ed aggiungendone delle nuove.

Skeleton

La classe *Skeleton* riporta sia l'identificativo associato dal dispositivo sia un id assegnatogli dal sistema, oltre allo stato delle mani e dei joint. Quest'ultimi sono descritti attraverso la classe *JointInformation*. La classe *JointInformation* si appoggia alle classi *JointType* e *TrackingState*, fornite dalla libreria *Microsoft.Kinect*, consentendo di usufruire in maniera più semplice e intuitiva le informazioni sulle varie giunture quali:

- identificativo del corpo a cui appartengono;
- il tipo (mano destra, polso destro, spalla sinistra, ecc.);
- le coordinate;
- l'orientamento (rotazione);
- lo stato (Tracked, NotTracked o Inferred);

In ogni *JointInformation* troviamo anche il metodo *Clone*, che deriva dall'interfaccia *ICloneable* e viene utilizzata per creare delle copie di questi scheletri. Questa funzione è indispensabile per poter mantenere uno storico dei movimenti eseguiti dall'utente. Anche la classe *Skeleton* è dotata del metodo *Clone*, (funzione che la classe *Body* non prevede invece).

Le informazioni sulle giunture vengono inoltre utilizzate anche per fare una ricostruzione delle principali ossa che compongono il corpo. Ogni osso viene individuato come il segmento che unisce due joint, ed è rappresentato nella libreria dalla classe *Bone*. Per ognuno di esso vengono calcolati la posizione iniziale e finale, la lunghezza, l'orientamento e l'angolo che formano con le altre ossa. Tenendo conto delle giunture riconosciute dalla *Kinect*, il sistema genera e gestisce per ogni utente 24 ossa.

```
1  private static void boneBuilder(List<Bone> bones)
2  {
3          // Torso
4          bones.Add(new Bone(JointType.Head, JointType.Neck));
5          bones.Add(new Bone(JointType.Neck, JointType.SpineShoulder));
6          bones.Add(new Bone(JointType.SpineShoulder, JointType.SpineMid));
7          bones.Add(new Bone(JointType.SpineMid, JointType.SpineBase));
8          bones.Add(new Bone(JointType.SpineShoulder, JointType.ShoulderRight
       ));
9          bones.Add(new Bone(JointType.SpineShoulder, JointType.ShoulderLeft)
       );
10         bones.Add(new Bone(JointType.SpineBase, JointType.HipRight));
11         bones.Add(new Bone(JointType.SpineBase, JointType.HipLeft));
12
13         // Right Arm
14         bones.Add(new Bone(JointType.ShoulderRight, JointType.ElbowRight));
15         bones.Add(new Bone(JointType.ElbowRight, JointType.WristRight));
16         bones.Add(new Bone(JointType.WristRight, JointType.HandRight));
17         bones.Add(new Bone(JointType.HandRight, JointType.HandTipRight));
18         bones.Add(new Bone(JointType.WristRight, JointType.ThumbRight));
19
20         // Left Arm
21         bones.Add(new Bone(JointType.ShoulderLeft, JointType.ElbowLeft));
22         bones.Add(new Bone(JointType.ElbowLeft, JointType.WristLeft));
23         bones.Add(new Bone(JointType.WristLeft, JointType.HandLeft));
24         bones.Add(new Bone(JointType.HandLeft, JointType.HandTipLeft));
25         bones.Add(new Bone(JointType.WristLeft, JointType.ThumbLeft));
26
27         // Right Leg
28         bones.Add(new Bone(JointType.HipRight, JointType.KneeRight));
29         bones.Add(new Bone(JointType.KneeRight, JointType.AnkleRight));
30         bones.Add(new Bone(JointType.AnkleRight, JointType.FootRight));
```

```
31
32          // Left Leg
33          bones.Add(new Bone(JointType.HipLeft, JointType.KneeLeft));
34          bones.Add(new Bone(JointType.KneeLeft, JointType.AnkleLeft));
35      bones.Add(new Bone(JointType.AnkleLeft, JointType.FootLeft));
36
```

Listing 4.3: Creazione delle ossa a partire dalle joint riconosciute dal sistema

Ogni *Skeleton* contiene informazioni anche sul viso dell'utente a cui sono associati. I dati raccolti sono quelli derivanti dalle classi *FaceFrame* e *HighDefinitionFaceFrame*; ciò consente al programmatore di poter utilizzare sia dati meno pesanti ma con meno informazioni (*FaceFrame*), sia a ricostruzioni del volto molto più dettagliate ma più pesanti da gestire.
Attraverso gli *HighDefinitionFaceFrame* la classe *Skeleton* può riportare non solo il colore dei capelli dell'utente a cui è associato, ma anche riconoscere delle espressioni facciali, la presenza o meno di occhiali e un modello in 3d che descrive i punti principali di un volto (ovvero occhi, naso e bocca). Tutti questi dati possono essere utilizzati per sviluppare efficienti e precisi sistemi di *facial recognition*.

Stream

Tramite degli eventi la classe *AcquisitionManager* comunica al sistema che si occupa della stampa della ricezione di un nuovo frame. Questa componente è rappresentata dalla classe *StreamManager* e mediante questa classe l'utente può indicare quale tipo di dato stampare. Il sistema prevede la stampa dei frame di tipo *BodyIndex*, *Color*, *Depth* o *Infrared*, e la stampa dei corpi rilevati dal dispositivo.
StreamManager si occupa quindi:

- di inizializzare i *WriteableBitmap*, uno per ogni tipo di dato, e che conterranno l'immagine costruita a partire da un frame ricevuto dal dispositivo.
- di aggiornare le *WriteableBitmap* sulla base dei dati ricevuti e gestiti da *AcquisitionManager*;
- di comunicare, tramite degli eventi, l'aggiornamento di una *WriteableBitmap*;

La costruzione di un'immagine a partire dai dati ricevuti dalla *Kinect* viene eseguita dalla classe *StreamUpdate*. Questa classe è caratterizzata dal metodo *convertBitmap*, una funzione che prende in input una *WriteableBitmap* e la descrizione di un frame o dei corpi rilevati dal dispositivo.

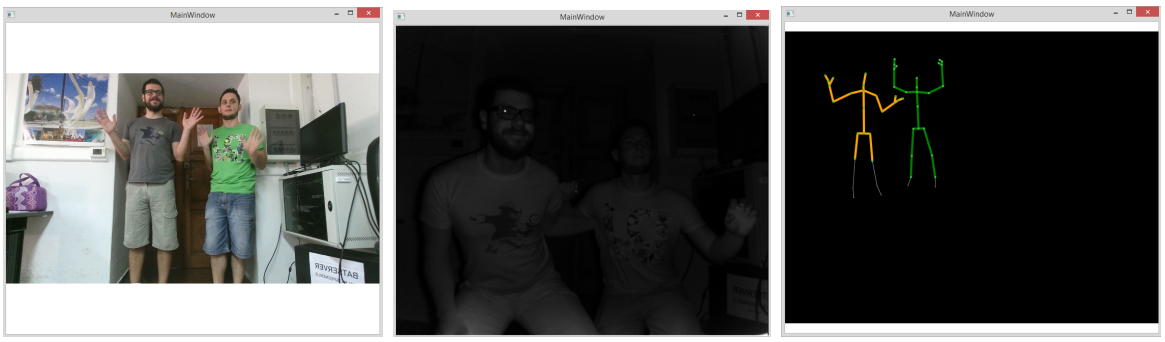

Figure 4.8: Da sinistra: Color stream, Infrared stream e lo stream degli scheletri tracciati.

GestIT

La componente che gestisce la *Kinect* prevede altre tre classi:

SkeletonToken : rappresenta un token valido per il sistema *GestIT*. Ogni token inviato al sistema rappresenta uno *Skeleton* valido. Al suo interno troviamo anche un buffer degli ultimi scheletri recuperati dal sistema.

KinectSensor : è la classe che gestisce l'interazione tra la componente della *Kinect* e quella di *GestIT*. Si occupa di creare i token degli scheletri rilevati dal dispositivo e inviati dalla classe *AcquisitionManager*.

KinectStateSequence : si occupa di gestire, a tempo d'esecuzione, i dizionari che contengono gli ultimi aggiornamenti ricevuti per ogni scheletro valido rilevato; questa operazione viene effettuata a partire dal tipo di token ricevuto.

4.3.2 Leap

L'altro dispositivo che la libreria supporta attualmente è il *Leap Motion*. La componente della libreria che si occupa di gestire i dati inviati e la comunicazione con il dispositivo è la classe *AcquisitionLeapManager*; all'interno di questa classe troviamo un gestore per ogni evento che il *Leap* può lanciare durante l'esecuzione.
Tali eventi sono diversi, e vengono utilizzati dal dispositivo per comunicare situazioni varie, come l'arrivo di un frame oppure un cambiamento di stato; tra questi i più importanti sono:

OnConnect : l'evento viene lanciato dal dispositivo per comunicare l'apertura della connessione tra il dispositivo; al verificarsi di questo evento l'*AcquisitionLeapManager* provvede ad inizializzare le strutture dati che conterranno i dati ricevuti.

OnFrame : il *Leap Motion* genera questo evento per comunicare l'invio di un nuovo frame; la classe *LeapData* viene utilizzata dalla libreria per memorizzare i dati più importanti contenuti nel frame ovvero:

- la lista delle mani tracciate dal dispositivo (rappresentati dalla classe *Hand*);
- la lista dei *Finger* rilevati;
- la lista degli *Pointable*;
- la lista delle immagini all'infrarosso registrate dal dispositivo;
- la lista delle gesture di default (Circle, Key Tap, Screen Tap e Swipe) riconosciute dal sistema (e che potranno essere poi usate per la definizione di gesture più complesse tramite la componente *GestIT*);
- l'*InteractionBox*, che rappresenta il punto di vista del *Leap Motion* ed è in sostanza una regione rettangolare;
- un array che contiene la rappresentazione del frame nel formato *byte*;
- un float che rappresenta il tempo di arrivo del frame;

OnExit : evento che viene lanciato quando il programma che sta usando il leap viene chiuso; al giungere di questo evento, la classe *AcquisitionLeapManager* provvede a salvare l'ultimo frame disponibile e a chiudere la connessione con il dispositivo.

La componente che si occupa di questo dispositivo è provvista anche di altre classi:

StreamManager and StreamUpdate : che si occupano rispettivamente di gestire e aggiornare la *WriteableBitmap* costruita a partire dalle immagini all'infrarosso acquisite dal dispositivo; anche in questo caso l'aggiornamento della *WriteableBitmap* viene comunicata da questa componente attraverso degli eventi.

HandToken : è la classe che rappresenta un token nella componente che implementa *GestIT*. Il sistema crea un *HandToken* per ogni mano rilevata. Al suo interno troviamo anche una lista delle vecchie informazioni riguardanti la stessa mano.

LeapSensor : rappresenta l'interfaccia tra la classe che gestisce le informazioni ricevute dal dispositivo (l'*AcquisitionLeapManager*) e il sistema che implementa la definizione e il riconoscimento delle gesture; provvede a creare gli *HandToken* a partire dalle informazioni contenute nell'ultimo *LeapData* costruito dal sistema.

LeapStateSequence : gestisce gli *HandToken* ricevuti dal *LeapSensor* e il buffer che memorizza gli ultimi token ricevuti. In base al tipo di token ricevuto, aggiorna le informazioni contenute nel dizionario.

4.3.3 Audio

La libreria anche per quanto concerne il riconoscimento dei comandi vocali, utilizza come dispositivo di rilevamento dei segnali acustici la *Kinect One*. In questo caso il dispositivo comunica le informazioni circa i segnali rilevati tramite gli *AudioBeamFrame*, e la gestione dei dati avviene tramite la classe *AcquisitionAudioManager*.
Tale classe raccoglie e gestisce sia i semplici segnali audio rilevati che i comandi vocali impartiti dagli utenti. I segnali audio sono racchiusi all'interno dei frame inviati dall'utente, e sono definiti dalla classe *AudioBeamSubFrame*; questi segnali vengono descritti nella libreria tramite la classe *AudioData*, che fornisce tutta una serie di informazioni raccolte a partire dai subframe:

beamAngle : è un float che rappresenta l'angolo, rispetto alla *Kinect*, da cui è partito il suono;

BeamAngleConfidence : rappresenta invece la confidence del segnale;

AudioBodyCorrelations : è una lista che indica per ogni utente rilevato se il suono è stato emesso da lui o meno;

frameData : rappresenta l'array di byte che descrive il subframe;

duration : la durata del segnale, è rappresentato dalla classe *TimeSpan*;

Il riconoscimento dei comandi vocali viene implementato sfruttando le funzioni messe a disposizione dalla libreria *Microsoft.Speech*. Questa libreria consente:

- la creazione di grammatiche;
- l'inizializzazione e la gestione di sistemi di riconoscimento vocale;
- l'interpretazione dei dati restituiti dal sistema di riconoscimento;

- operare con i comandi vocali riconosciuti;

Il riconoscimento vocale è implementato utilizzando i seguenti elementi:

- il sistema di riconoscimento vocale messo a disposizione dalla libreria *Microsoft.Speech*;
- gli *AudioData* costruiti a partire dai segnali rilevati dalla *Kinect*;
- il dizionario delle parole da riconoscere (rappresentato come un file *.xml*) e passato in input dal programmatore;
- la grammatica costruita dal dizionario passato in input;

Attraverso questo sistema, la classe *AcquisitionAudioManager* è in grado di rilevare i comandi vocali inviati dall'utente, e di comunicarlo poi al programmatore e alle restanti componenti della libreria tramite degli eventi. Inoltre, utilizzando la componente che implementa *Gestit*, è possibile definire delle sequenze di parole, o meglio delle espressioni, che possono rappresentare dei comandi più complessi; questo permetterebbe di migliorare l'esperienza dell'utente, consentendogli un interazione più semplice ed efficace con il computer.

Le classi *AudioSensor* e *AudioStateSequence* provvedono alla creazione dei token e alla gestione di quest'ultimi; rappresentano l'interfaccia tra la componente che si occupa di rilevare i comandi vocali e la componente di *GestIT*.

Chapter

5
TEST

Una volta concluso lo sviluppo e la realizzazione della libreria, per dimostrare l'utilizzo e l'efficacia delle varie componenti della libreria, abbiamo implementato delle demo.

- Una prima demo dimostra la creazione dei meccanismi di feedback e di feedforward basati sul sistema *OctoPocus*;
- Una secondo demo che dimostra invece il modello di feedback continui, generati per informare in tempo reale l'utente circa lo stato del programma rispetto ai movimenti che sta eseguendo;

Feedforward La prima demo è stata realizzata utilizzando l'ambiente di sviluppo *Unity 5.0*. Per la realizzazione di questa demo abbiamo sfruttato l'estensione della nostra libreria dedicata a questo ambiente di sviluppo. Il programma realizzato serve a dare un esempio delle potenzialità dei metodi di feedback e di feedforward generati dalla libreria, nel contesto di interfacce e sistemi d'interazione tramite gesture.
In questa demo sono state implementate quattro gesture, realizzabili tutte quante tramite il movimento della mano destra:

PanX: come visto già nei capitoli precedenti, consiste in un movimento continuo della mano lungo l'asse delle X. La gesture inizia con la chiusura della mano destra dell'utente e si conclude con la sua apertura.

PanY: è il movimento speculare al *PanX*. In questo caso la mano destra seguirà un movimento continuo lungo l'asse delle Y.

Arch: questa gesture rappresenta un movimento angolare di 45°. Come le precedenti il movimento di start e di end sono descritti, rispettivamente, dalla chiusura e dall'apertura della mano destra. In questo caso però l'utente deve eseguire un movimento più complesso sia lungo l'asse delle X che lungo l'asse delle Y.

Elle : infine l'ultima gesture definita per questo tipo di applicazione è quella che rappresenta una "L". In questo caso la gesture è descritta attraverso due *Disabling*, ognuno delle quali descrive un lato della "L".

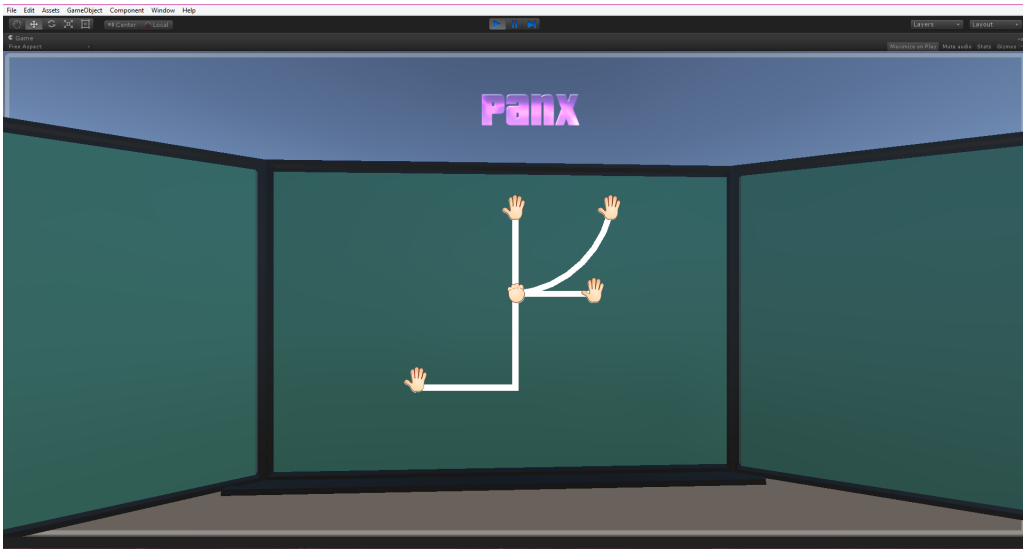

Figure 5.1: Visualizzazione delle tracce delle gesture definite per la demo su Unity.

Feedback La seconda demo è realizzata tramite *Windows Presentation Foundation* (*WPF*) attraverso l'ambiente di sviluppo Visual Studio 2015. Quest'applicazione è stata sviluppata per mettere in evidenza il modello di feedback implementato nella libreria; in questo caso la scena risulta essere sempre molto semplice, e rappresenta una tavolozza dove l'utente può disegnare delle forme (tramite delle gesture) e interagire con esse. Mentre i feedback vengono visualizzati attraverso delle finestre in alto dello schermo, che descrivono gli stati del programma durante l'esecuzione di un movimento, sulla base dei dati ricevuti dal dispositivo.
Le gesture implementate in questo caso sono sette, di cui quattro realizzabili tramite la mano destra e tre attraverso la mano sinistra. La mano destra viene utilizzata per disegnare le forme sulla tavolozza; le forme previste sono quattro a cui sono associate altrettante gesture:

Circle: la prima forma implementata è rappresentata dal cerchio. La gesture che la descrive è rappresentata da tre *Ground Term*, ovvero chiusura della mano destra, movimento circolare e apertura della mano. Per verificare se il movimento effettuato equivale, all'incirca, ad un cerchio si controlla innanzitutto se la fine del movimento coincide con l'inizio, e quindi se la lunghezza dei due semicerchi è più o meno simile.

Rectangle: l'altra forma implementata è quella del rettangolo. In questo caso la sua descrizione è composta da ben quattro *Disabling*, ognuno dei quali rappresenta un lato del rettangolo. Durante l'esecuzione della gesture le funzioni di *accept* controllano costantemente che i lati verticali siano quelli più corti rispetto ai lati orizzontali.

Square: la gesture che rappresenta il quadrato è molto simile a quella che descrive il rettangolo. Anch'essa è costituita da quattro *Disabling*, uno per ogni lato; in questo caso però si verifica durante la sua esecuzione se tutti i lati sono lunghi, all'incirca, lo stesso tanto.

Triangle: infine l'ultima forma prevista è quella del triangolo. In questo caso entrano in gioco solo tre *Disabling* che descrivono il movimento che deve eseguire l'utente. Mentre il secondo *Disabling* controlla se il movimento della mano avviene lungo l'asse delle X. Gli altri due operatori, invece, verificano che il movimento avvenga in maniera costante sia lungo l'asse delle X che lungo quello delle Y e che si formi con la base un angolo di 45° circa.

Le gesture implementate tramite la mano sinistra invece permettono all'utente di:

Move/Cancel: è il comando con la quale l'utente può spostare o cancellare un oggetto. La gesture che lo implementa consiste in un movimento abbastanza semplice, dove l'utente seleziona con la mano sinistra l'oggetto che vuole spostare (o cancellare), quindi spostando la mano si sposta anche la forma selezionata. Se la mano viene aperta oltre la finestra che rappresenta l'applicazione, allora l'oggetto è cancellato, viceversa la posizione in cui la mano è stata aperta rappresenta le nuove coordinate dell'oggetto.

Swipe: attraverso lo swipe l'utente può pulire la tavolozza, eliminando con unico gesto tutti le forme in esso presenti. La gesture che la descrive è rappresentata da un movimento orizzontale molto veloce della mano sinistra aperta, che si conclude dopo aver raggiunto una certa lunghezza.

Ora andremo ad esaminare un esempio d'interazione nell'applicazione sviluppata. Nell'esempio partiremo da una tavolozza con già due forme disegnate e in cui si vuole aggiungere un nuovo cerchio:

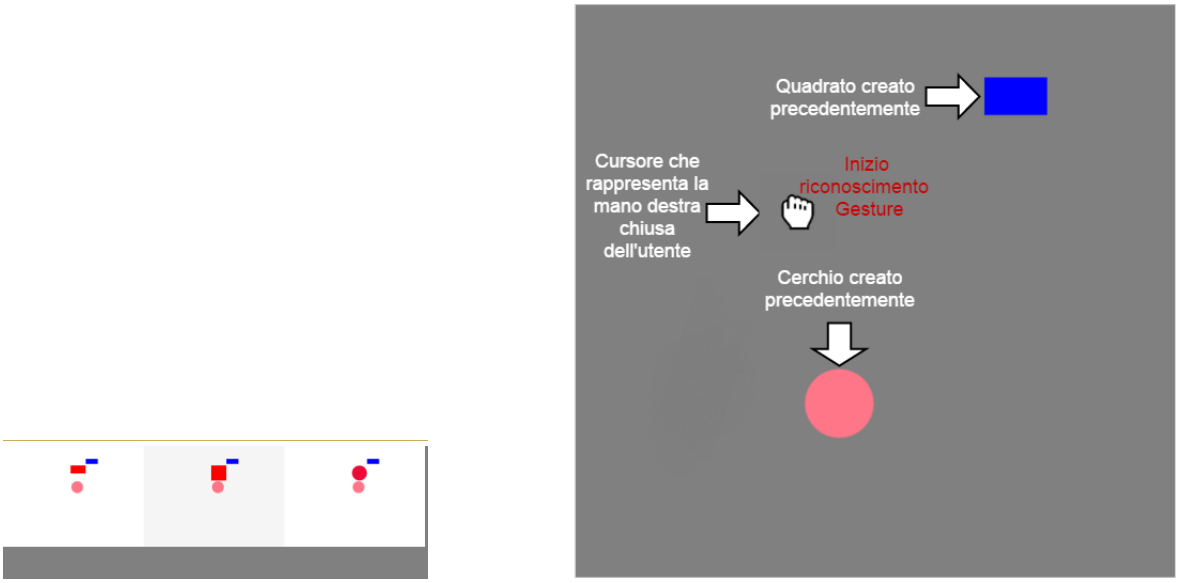

Figure 5.2: Nel momento in cui l'utente chiude la mano destra, vengono riconosciute inizialmente le gesture con cui è possibile disegnare una forma. E queste semplicemente perché in tutte e quattro il term di Start è rappresentato dalla mano destra chiusa. Nella figura a sinistra possiamo notare le finestre utilizzate per mostrare i feedback, mentre in quella a destra la tavolozza in cui sta disegnando l'utente.

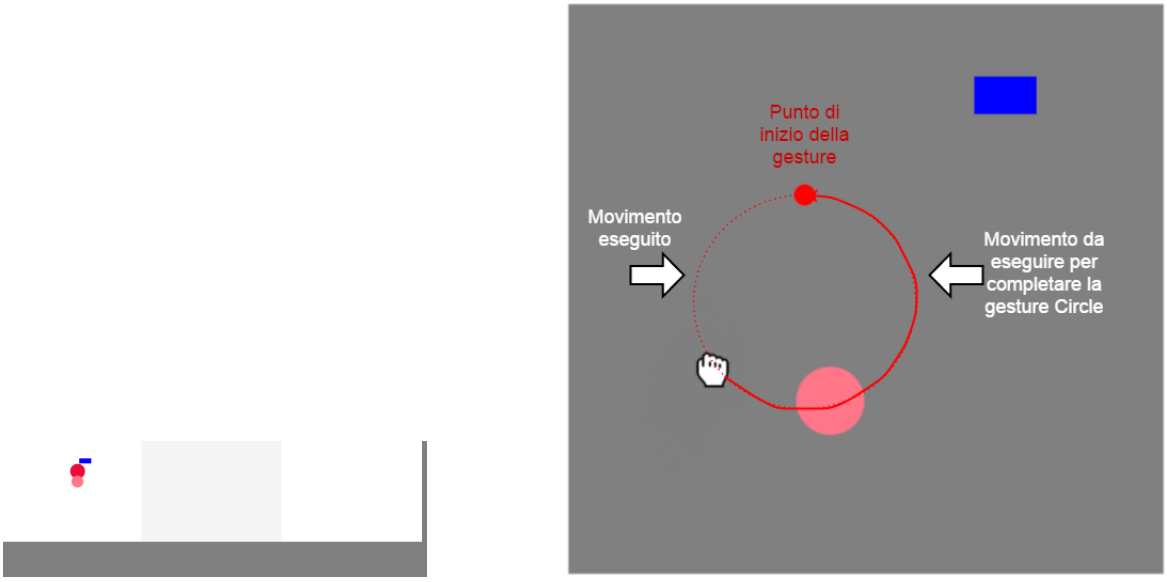

Figure 5.3: L'utente, dopo aver chiuso la mano destra, esegue un movimento circolare; di conseguenza solo la gesture *Circle* continua ad essere riconosciuta, mentre per le altre il riconoscimento termina in quanto il movimento eseguito non corrisponde alla loro definizione. E questo è sottolineato dal fatto che ora vengono mostrati solo i feedback legati al cerchio.

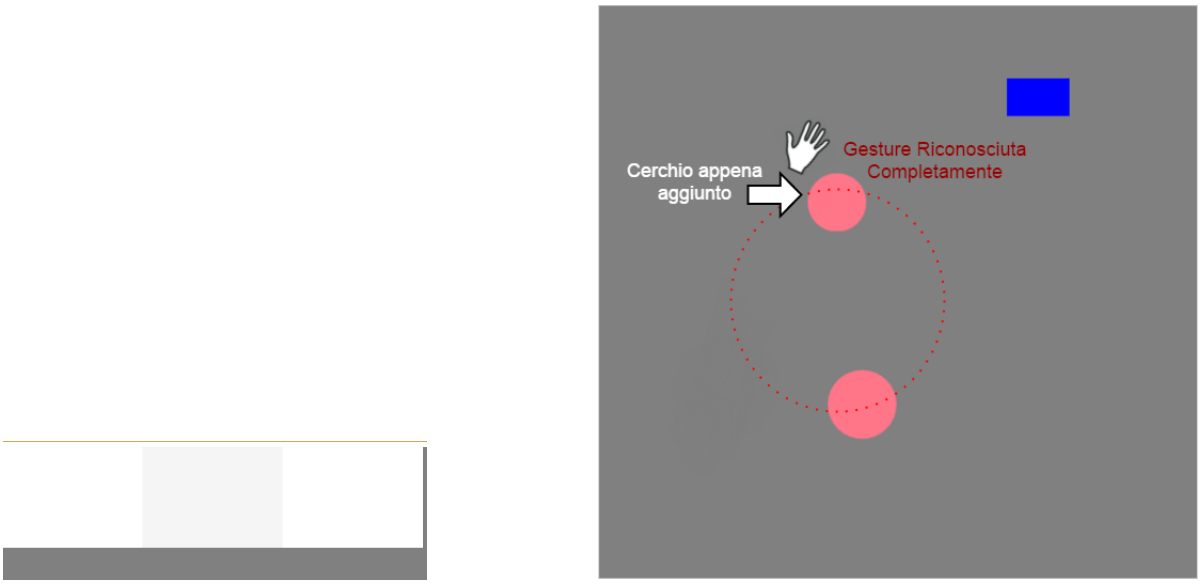

Figure 5.4: Una volta che l'utente ha "disegnato" un cerchio a mezz'aria e in seguito all'apertura della mano viene aggiunta nella scena il nuovo cerchio. Una volta conclusa la gesture la finestra dei feedback viene pulita, in attesa che l'utente inizi l'esecuzione di una nuova gesture.

Chapter

6
CONCLUSION AND FUTURE WORK

Al completamento dei lavori, gli obbiettivi che ci siamo posti all'inizio della tesi sono stati sicuramente raggiungiti. Siamo riusciti a modellare e sviluppare delle architetture per la generazione di feedback e di feedfoward continui per interfacce basate sul riconoscimento delle gesture. Inoltre, attraverso il sistema di riconoscimento e definizione delle gesture basato su *GestIT* siamo riusciti a dimostrare gli effetti positivi ottenuti nell'interazione attraverso il modello messo a punto.

Gesture L'implementazione del framework *GestIT* permette di poter utilizzare la libreria anche come sistema per la creazione e la definizione di gesture. Questa architettura risulta essere molto efficiente ed efficace sia nella fase di creazione della gesture che nella sua fase di riconoscimento; Anche se la sua affidabilità è strettamente legata al dispositivo di interazione utilizzato, sulla qualità delle informazioni inviate.
In futuro miriamo a creare un sistema ancora più rapido e semplice per la definizione e la realizzazione di una gesture, ancora a più alto livello, che permetta non soli ai programmatori ma anche agli utenti inesperti la definizione di una gesture; l'obiettivo è quello di facilitare la formulazione delle gesture, rendendo più semplice la definizione delle funzioni di *accept*.
Altre idee che vorremo attuare in futuro sono quelle di utilizzare la *Computer Graphics* e l'*Artificiale Intelligence* per migliorare il sistema di riconoscimento; preso in input il movimento dell'utente, il sistema esegue il riconoscimento utilizzando come dati direttamente la forma ottenuta, lo spazio occupato e la velocità con la quale il movimento è eseguito dall'utente.

Un approccio che vorremo utilizzare in futuro per migliorare il riconoscimento è il metodo *Monte Carlo*; questo metodo fa parte della famiglia dei metodi statistici non parametrici, ed è utile per superare i problemi computazionali legati ai test esatti. Tale metodo è stato già utilizzato con successo per gestire lo stato e i feedback in sistemi[46] dove l'input dell'utente può risultare incerto.
Riteniamo che il metodo di *Monte Carlo* possa essere applicato anche nel nostro contesto con ottimi risultati. Infatti preso in input un certo movimento, questo modello

permetterebbe di determinare a quale gesture è più probabile che assomigli l'input ricevuto, e sarebbe di grande aiuto anche nella fase di generazione dei feedback. Per esempio, per disegnare un quadrato, il riconoscimento basato su questo metodo lavorerebbe anche sulla forma tralasciando magari il lato da cui si iniziare disegnare o la rotazione applicata.

Feedback Il sistema di feedback modellato ben si adatta ad operare in sistemi in cui l'interazione avviene tramite gesture; questo potrebbe permettere in futuro la realizzazione di sistemi in grado di fornire dei feedback immediati e utili agli utenti, semplificando la loro esperienza con questo tipo di sistemi e incoraggiandoli a ripetere tali esperienze. Tuttavia per motivi di tempo non è stato possibile migliorare l'architettura realizzando un modello e un approccio probabilistico più affidabile ed efficiente. È già previsto in un futuro prossimo la realizzazione di un modello basato sulle *catene di Markov*, in cui gli eventi saranno rappresentati dagli stati e ogni stato sarà collegato tramite un link pesato che ne rappresenta la probabilità; l'utilizzo di questo modello consente di poter calcolare in maniera più efficiente quale sarà il movimento con la probabilità più alta di essere eseguito.

Un altro elemento da implementare per migliorare il modello di generazione dei feedback è quello di poter permettere, a tempo di esecuzione, la scelta tra le diverse gesture riconosciute nel movimento eseguito dall'utente; ciò permetterebbe all'utente di poter cambiare il commando da eseguire, nel caso in cui abbia sbagliato qualcosa nell'esecuzione della gesture.

Feedforward Dal canto suo la componente che si occupa di generare i feedforward ha dimostrato che il modello *OctoPocus* non solo è ideale per imparare ad eseguire delle gesture, ma si adatta anche nei sistemi in 3d come dimostrato con l'estensione per *Unity 5* che abbiamo implementato.
Dai test è risultato che attraverso questo modello l'apprendimento dell'utente risulta essere facilitato e molto più efficiente, anche grazie alle informazioni che vengono visualizzate al compimento corretto di ogni gesto. Purtroppo a causa della carenza di tempo non è stato possibile completare il sistema di feedforward ispirato al sistema OctoPocus. In futuro si prevede di applicare una colorazione differente ai segmenti che compongono la traccia della gesture a seconda della confidenza associata all'esecuzione di un determinato gesto.

Devices La libreria, come visto in precedenza, supporta sia la *Kinect* che il *Leap Motion*. Uno degli obiettivi principali seguiti nella realizzazione della libreria era quello di rendere più semplice e immediato l'utilizzo dei suddetti dispositivi. Infatti grazie alla classe *AcquisitionManager* e alle altre componenti, del sistema che supporta la *Kinect* il lavoro del programmatore si è ridotto di gran lunga. L'architettura, la struttura della libreria e l'utilizzo degli eventi ha reso l'elaborazione dei dati molto più veloce consentendo un incrementando delle prestazioni sul calcolo dei dati e la visualizzazione di stream fluidi. Stesso discorso si può applicare per il *Leap Motion*.
L'obiettivo per il futuro sarà quello di implementare il supporto per altri dispositivi di tracciamento, migliorando l'efficienza del sistema e aiutando l'utente durante l'interazione con questi altri dispositivi.

BIBLIOGRAPHY

[1] http://www.jnd.org/dn.mss/natural_user_interfa.html

[2] http://www.virtuix.com/

[3] https://www.leapmotion.com

[4] http://www.ima.umn.edu/imaging/W5.22-26.06/activities/Buhmann-Joachim/templates.pdf

[5] Rung-Huei Liang, Ming Ouhyoung
A real-time continuous gesture recognition system for sign language. [1998] FG'98, pp. 558

[6] http://www.gesturetekhealth.com/products-rehab.php

[7] Ushaw G., Ziogas E., J. Eyre, Morgan G.
An Efficient Application of Gesture Recognition from a 2D Camera for Rehabilitation of Patients with Impaired Dexterity. [2011] UIST'11, pp. 235-244.

[8] Vaitukaitis V., Bulling A.
Eye Gesture Recognition on Portable Devices. [2012] UbiComp '12, pp. 711-714

[9] http://www.nviso.ch/

[10] https://www.oculus.com/en-us/

[11] http://www.xbox.com/it-IT/xbox-one/accessories/kinect-for-xbox-one

[12] https://msdn.microsoft.com/en-us/library/dn758675.aspx

[13] http://kinecthealth.co.uk

[14] http://www.neurobs.com/menu_presentation/menu_features/kinect

[15] https://vimeo.com/121436114

[16] http://blogs.msdn.com/b/kinectforwindows/archive/2014/01/13/robotic-control-gets-a-boost.aspx

[17] Owano N., NASA uses Leap Motion to move ATHLETE rover, 2013, http://phys.org/news/2013-04-nasa-motion-athlete-rover-video.html

[18] https://en.wikipedia.org/wiki/Gesture_recognition#/media/File:BigDiagram2.jpg

[19] http://www.stokowski.org/Harvey%20Fletcher%20Bell%20Labs%20Recordings.htm

[20] http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/speechreco/

[21] Yen Y., Fanty M., Cole R. A.
Speech recognition using neural networks with forward-backward probability generated targets. [1997] ICASSP'97, Vol. 4, pp. 3241 - 3244

[22] http://neuralnetworksanddeeplearning.com/

[23] http://www.nature.com/nbt/journal/v22/n10/full/nbt1004-1315.html

[24] http://www.dartmouth.edu/ chance/teaching_aids/books_articles/probability_book/Chapter11.p

[25] http://www.springer.com/cda/content/document/cda_downloaddocument/9783540740476-c1.pdf?SGWID=0-0-45-452103-p173751818

[26] http://www.nuance.com/forbusiness/mobilesolutions/dragondrive/index.htm

[27] http://www.nuance.com/for-healthcare/by-solutions/speech-recognition/index.htm

[28] http://www.stephanepigeon.com/Docs/TRIST037ALL.pdf

[29] http://www.apple.com/ios/siri/

[30] http://www.windowscentral.com/cortana

[31] Bain K., Basson S.H., Wald M.
Speech recognition in university classrooms: liberated learning project. [2002] Assets '02, pp. 192-196

[32] Cabanas de Paz R., Flores M.Julia., J. Martìnez-Gòmez
Dynamic Bayesian Networks for Gesture Recognition. [2011] XII Work. Ag. F.

[33] Mitra S., Acharya T.
Gesture recognition: A survey. [2007] IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS - PART C - Volume 37, Number 3, pp. 311-324

[34] Xu D.
A Neural Network Approach for Hand Gesture Recognition in Virtual Reality Driving Training System of SPG. [2006] ICPR '06 Proceedings of the 18th International Conference on Pattern Recognition - Volume 03 pp. 519-522

[35] Hafizur Rahman Md., Jinia Afrin
Hand Gesture Recognition using Multiclass Support Vector Machine. [2013] International Journal of Computer Applications - Volume 74, No.1

[36] Yamato J., Ohya J., and Ishii K.
Recognizing human action in time sequential images using hidden Markov model. [1992] Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn., Champaign, IL, pp. 379-385

[37] http://rob.schapire.net/papers/explaining-adaboost.pdf

[38] https://msdn.microsoft.com/en-us/library/dn785524.aspx

[39] Kin K., Hartmann B., DeRose T., Agrawala M.
Proton: multitouch gestures as regular expressions. [2012] CHI'12, pp. 2885-2894

[40] Echtler F., Butz A.
GISpL: gestures made easy. [2012] TEI '12, pp. 233-240

[41] http://tangible.media.mit.edu/

[42] https://www.interaction-design.org/encyclopedia/tangible_interaction.html

[43] Bau O., Mackay W.E.
Octopocus a Dynamic Guide for Learning Gesture-Based Command Sets. [2008], UIST'08 pp. 37-46

[44] Sodhi R., Benko H., Wilson A.D.
LightGuide: Projected Visualizations for Hand Movement Guidance. [2012], CHI'12, pp. 179-188

[45] Schwarz J., Mankoff J., Hudson S.E.
An Architecture for Generating Interactive Feedback in Probabilistic User Interfaces. [2015], CHI'15, pp. 2545-2554

[46] Schwarz J., Mankoff J., Hudson, S.E.,
Monte Carlo methods for managing interactive state, action and feedback under uncertainty. [2011] UIST'11, pp. 235-244.

[47] Spano L.D., Cisternino A., Paternò F., Fenu G.
GestIT: a declarative and compositional framework for multiplatform gesture definition. [2012], EICS'13, pp. 187-196

[48] Ainsworth W.A, Pratt S.R.
Feedback strategies for error correction in speech recognition systems. [1992], International Journal of Man-Machine Studies, pp. 833-842

[49] Norris D., McQueen J.M. e Cutler A.
Merging information in speech recognition: Feedback is never necessary. [2000], Behavioral and Brain Sciences 2000, pp. 299-370
Schwarz J., Mankoff J., Hudson S.E.
Monte Carlo methods for managing interactive state, action and feedback under uncertainty [2011] UIST 2011, pp. 235-244.

[50] Kim G.J.
Human-Computer Interaction Fundamentals and Practice. [2015]: CRC Group, NY

[51] Dix A., Finlay J., Abowd G.D., Beale R.
Interazione uomo-macchina. [2004]: McGraw-Hill, NY

[52] Rabiner L., Juang B.H.
Fundumentals of Speech Recognition. [1993]: Prentice-Hall International, NJ

[53] Paternò F.,
Model-based design and evaluation of interactive applications. [2000]: Applied Computing, Springer