Politecnico di Milano

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Specialistica in Ingegneria Informatica

Gesture Recognition tramite Kinect utilizzando il classificatore Template-Based KMC

Un caso di studio

Candidato:
Mikhail Vorontsov
Matricola 755744

Relatore:
Prof.ssa Giuseppina Gini

Anno Accademico 2011–2012

Indice

Elenco delle figure

Elenco delle tabelle

Abstract

Con il termine *Gesture Recognition* si identifica quel processo che permette di riconoscere e classificare determinate azioni utilizzando le informazioni provenienti da specifici sensori. Si tratta di un ambito dell'informatica che ha da sempre attirato grande attenzione viste le sue enormi potenzialità. I campi di applicazione del *Gesture Recognition* sono infatti molteplici, dallo sviluppo di interfacce uomo-macchina, robotica, applicazioni di video sorveglianza ai più recenti videogiochi. Grazie al progresso tecnologico questo campo è sempre in continua evoluzione e ha permesso di raggiungere notevoli risultati in diversi ambiti, tra i quali anche quello sportivo. All'interno di questo contesto, più precisamente nell'ambito degli sport da combattimento, si inserisce il progetto *KMC (Kinect Multiplayer Classificator)*: l'implementazione di un classificatore per il *Gesture Recognition* basato su *Kinect* che permette di gestire e riconoscere le azioni compiute da due soggetti diversi. L'azione del *pugno al viso* costituisce il caso di studio trattato.

Capitolo 1

Introduzione

Con il termine *Gesture Recognition* si definisce quel processo che permette di riconoscere e classificare correttamente determinate azioni utilizzando le informazioni provenienti da specifici sensori. Si tratta di un ambito dell'informatica che ha da sempre attirato grande attenzione viste le sue enormi potenzialità. I campi di applicazione del *Gesture Recognition* sono infatti molteplici, dallo sviluppo di interfacce uomo-macchina, robotica, applicazioni di video sorveglianza ai più recenti videogiochi.

Grazie al progresso tecnologico questo campo è sempre in continua evoluzione e ha permesso di raggiungere già notevoli risultati in diversi ambiti. Nel campo dei videogiochi il *Wii Remote* e il più recente *Kinect* di *Microsoft* hanno infatti permesso di raggiungere esperienze di gioco senza precedenti coinvolgendo il giocatore al massimo livello.

Il *Gesture Recognition* trova applicazioni anche in ambito sportivo, soprattutto in sport quali tennis e baseball.

Proprio all'interno di questo contesto si inserisce il progetto *KMC (Kinect Multiplayer Classificator)*, un classificatore per il *Gesture Recognition* che permette di gestire e riconoscere tramite *Kinect* le azioni compiute da due soggetti diversi. L'ambito di applicazione è quello sportivo, in particolare quello degli sport da combattimento. In tali discipline lo scopo è quello di eseguire e mettere a segno contro il proprio avversario più tecniche possibili

come ad esempio pugni, calci ecc. Per ogni tecnica portata a segno viene assegnato un diverso punteggio che varia a seconda della zona del corpo colpita oppure la parte del corpo utilizzata per l'esecuzione. Il vincitore dell'incontro sarà quindi colui che riesce a portare a segno più tecniche possibili ottenendo il punteggio maggiore.

Scopo finale di tale progetto è sviluppare un'applicazione che permetta di riconoscere in tempo reale le diverse azioni (pugno, calcio, ecc) durante una sequenza di combattimento attribuendo il corretto punteggio per ogni determinata azione, diventando così un sistema di conteggio assoluto ed oggettivo. La tesi tratta la fase iniziale e il riconoscimento della più semplice azione del pugno al viso.

Nel secondo capitolo verrà presentato lo *Stato dell'Arte* riguardo al campo del *Gesture Recognition*, illustrando i principali tipi di approccio utilizzati durante ogni fase del processo. Verranno inoltre descritti gli attuali strumenti elettronici utilizzati negli sport da combattimento.

Il terzo capitolo introduce il caso di studio oggetto della tesi, presentando una breve descrizione del *Taekwon-Do ITF* con particolare riferimento al sistema di conteggio dei punti attuale.

Il quarto capitolo offre una panoramica per quanto riguarda la dotazione hardware e software utilizzata. Verranno quindi descritti il *Kinect*, l'ambiente di sviluppo e le diverse librerie adottate.

Nel quinto capitolo saranno invece descritti tutti i passi riguardanti lo sviluppo dell'applicativo e l'implementazione del classificatore. Saranno quindi approfonditi il processo di normalizzazione, l'algoritmo di riconoscimento, l'acquisizione di *template* e la parametrizzazione dell'applicativo.

Il sesto capitolo tratta la fase di acquisizione dei dati, ricoprendo aspetti quali l'ambiente, i modelli adottati e il tipo di approccio utilizzato durante la fase di registrazione dei *template*.

Nel settimo capitolo verranno discussi i risultati ottenuti durante le di-

verse fasi, analizzando le eventuali problematiche riscontrate e proponendo un iniziale overview dei punti di forza e debolezza del classificatore dopo le prime fasi di testing.

Nell'ottavo capitolo infine si tratteranno i possibili sviluppi futuri del progetto, descrivendo aspetti quali l'integrazione di nuovi sensori, la gestione del parallelismo e i possibili nuovi campi di applicazione del classificatore.

Capitolo 2

Stato dell'Arte

In questo capitolo verrà presentata una breve descrizione riguardo allo stato dell'arte nel campo dell' *Action Recognition* (Riconoscimento di Azioni), analizzando aspetti quali il tracking dei soggetti interessati, i modelli spaziali e temporali utilizzati, l'estrazione di features, le tipologie di hardware ed il riconoscimento di azioni. Infine verranno descritti gli attuali strumenti elettronici utilizzati negli sport da combattimento.

2.1 Action Recognition

L'*Action Recognition* rappresenta un campo di ricerca molto importante nell'ambito del *Computer Vision*, includendo numerose applicazioni quali sviluppo di interfacce uomo-macchina, software di video sorveglianza, robotica e molte altre. Storicamente, l'*Action Recognition* era suddiviso in sezioni quali *Gesture Recognition*, *riconoscimento di espressioni facciali* e *riconoscimento del movimento* per le applicazioni di video sorveglianza.

L'*Action Recognition* può essere definito come il processo mediante il quale si etichettano le azioni, cioè si assegna ad una determinata azione una particolare classificazione, utilizzando le osservazioni provenienti dai sensori. In Figura 2.1 sono rappresentati i principali componenti di un sistema di *Action Recognition* e la loro rispettiva relazione.

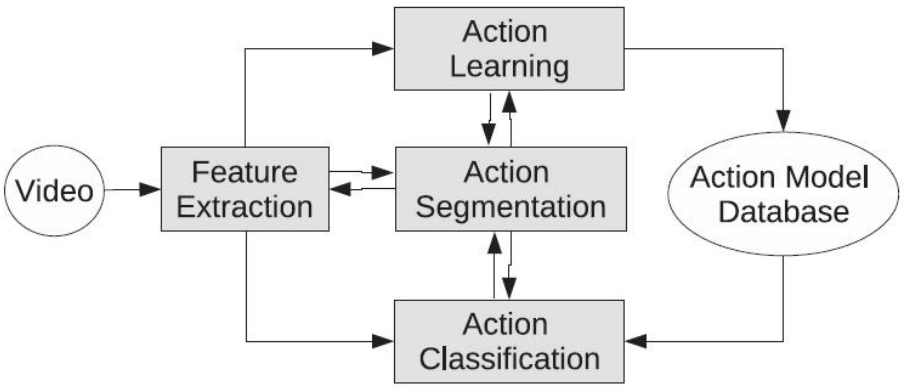

Figura 2.1: Struttura generale di un sistema di Action Recognition.

Il processo di *Feature Extraction* rappresenta il task più importante del processo di *Action Recognition* e consiste nell'estrarre gesti ed azioni dai video elaborati. A questo scopo possono essere utilizzati diversi modelli, da quelli più complessi dell'intero corpo umano a quelli più semplificati e stilizzati. In qualsiasi caso, bisogna far fronte a problemi quali la distinzione dello sfondo, la gestione delle luci e delle ombre. Inoltre è importante anche implementare soluzioni che permettano di far fronte alle differenze di abbigliamento o di struttura fisica nel caso i modelli osservati siano esseri umani.

I processi di *Action Learning* e *Action Classification* costituiscono gli step intermedi per poter permettere la costruzione di modelli statistici dalle *features* estratte, utilizzando queste informazioni per la successiva classificazione di nuove *features*. Il problema principale sono le praticamente infinite possibilità con cui un'azione si presenta: infatti cambia a seconda del soggetto, della dimensione, della velocità, del tempo di esecuzione etc.

Le azioni più comuni come calciare, tirare pugni, salutare con la mano rappresentano alcuni esempi di azioni che vengono eseguite in tanti possibili modi diversi. Costruire un modello che sia in grado di identificare le azioni

nel modo più completo possibile costituisce sicuramente la sfida più grande in questo campo.

Il processo di *Action Segmentation* è necessario per poter estrapolare dalle sequenze video solamente quelle informazioni necessarie per poter riconoscere le azioni, quindi in questa fase è importante ad esempio distinguere correttamente lo sfondo e differenziarlo dai soggetti in primo piano sui quali viene effettuata l'acquisizione.

Le tecniche di rappresentazione, segmentazione e riconoscimento di azioni umane vengono classificate in base a numerosi criteri diversi, ad esempio le parti coinvolte, la tipologia di *features* estratte, i modelli utilizzati per la classificazione. Vi sono numerosi tipi di approccio diversi che vengono suddivisi in base al dominio utilizzato: spaziale oppure temporale.

In un dominio di tipo spaziale, l'*Action Recognition* può essere basato sulle *features* globali delle immagini, allineate in accordo con la scena o con la posizione della videocamera, oppure *features* parametriche allineate in base alla geometria del campo umano, oppure ancora modelli statistici che descrivono la distribuzione e organizzazione spaziale delle *features*.

In un dominio di tipo temporale invece l'*Action Recognition* può essere applicato utilizzando per il confronto una sequenza di *features* da inizio a fine, oppure modelli *grammaticali* che descrivono in quale ordine avvengono i diversi momenti di un'azione, oppure modelli statistici che rappresentano la distribuzione delle possibili *features* nel corso del tempo.

2.1.1 Spatial Action Representations

In questa sezione verrà descritto brevemente il processo che permette di riconoscere azioni utilizzando informazioni di tipo visuale provenienti dai sensori servendosi di una rappresentazione di tipo spaziale. Come descritto precedentemente, il primo passo fondamentale nel processo di *Action Recognition* è l'estrazione delle *features* principali che identificano la posizione

ed il movimento del corpo umano.

Modello del corpo

Verranno ora descritti diversi metodi utilizzati per rappresentare la struttura spaziale delle azioni utilizzando come informazione la posizione del corpo umano. In ogni frame di un video, la posizione del soggetto viene ricavata da una grande varietà di diverse *features*, e il processo di *Action Recognition* avviene in base alla stima di tali posizioni. Si tratta di un approccio intuitivo e biologicamente-plausibile, sviluppato mediante un lavoro psicofisico sulla interpretazione visuale di movimenti biologici [1].

Johansson mostra come sia possibile identificare le azioni umane semplicemente dal movimento di *MLD*[1] attaccate al corpo, come mostrato in Figura 2.2. Il suo esperimento prende origine da un problema irrisolto: cioè se il cervello umano riconosce le azioni direttamente dai pattern 2D oppure esegue una ricostruzione nello spazio 3D. Nel contesto del *Machine Vision* tale problematica viene affrontata utilizzando due tipi di approccio diversi [2]:

Riconoscimento tramite ricostruzione modello 3D Tale processo viene suddiviso in due step principali: uno in cui si ricostruisce il modello 3D del corpo umano, ed uno in cui si lavora sulle traiettorie descritte dai diversi giunti del corpo. I due problemi principali riguardano il grande numero di gradi di libertà con i quali si muovono i giunti del corpo umano e tutte le loro forme possibili. Per far fronte a tale problematica deve essere selezionato un opportuno modello parametrico del corpo umano che sia possibile calibrare per permettere il riconoscimento delle azioni e la loro generalizzazione. Una grande varietà di modelli sono stati presentati nel corso della storia: alcuni di questi sono rappresentati in Figura 2.3.

[1] Moving Light Displays

Marr e Nishibara [3] proposero un modello basato su una gerarchia di cilindri 3D, come rappresentato in Figura 2.3a. Modelli invece più flessibili vengono utilizzati in [4]. L'approccio descritto in [5] utilizza ad esempio come punto di partenza un insieme di forme 2D e le trasforma in 3D (Figura 2.3c). Anche tecniche di *Motion Capture* che richiedono *marker* speciali attaccati al corpo sono state utilizzate per il riconoscimento di azioni, come descritto in [6], un esempio è rappresentato in 2.3b.

Riconoscimento diretto da modello 2D Tale approccio lavora direttamente sul modello 2D del corpo umano, utilizzando come informazioni le diverse parti del corpo senza nessuna conversione in 3D. Tipiche rappresentazioni 2D sono le cosiddette *Stick Figures* [7] (Figura 2.3f). Un altro tipo di approccio 2D utilizza invece una rappresentazione rozza del corpo adottando come informazioni ad esempio *blob* e *patches*, come la traiettoria di una mano [35] (Figura 2.3d), oppure la rappresentazione dell'intero corpo [8].

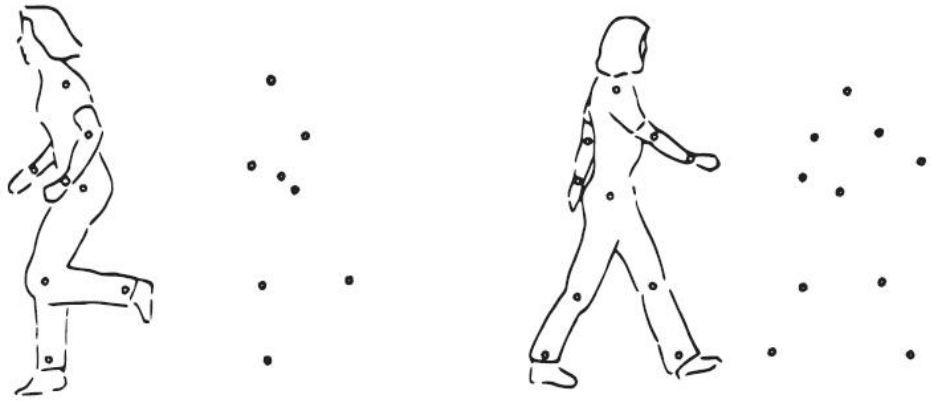

Figura 2.2: Moving Light Displays

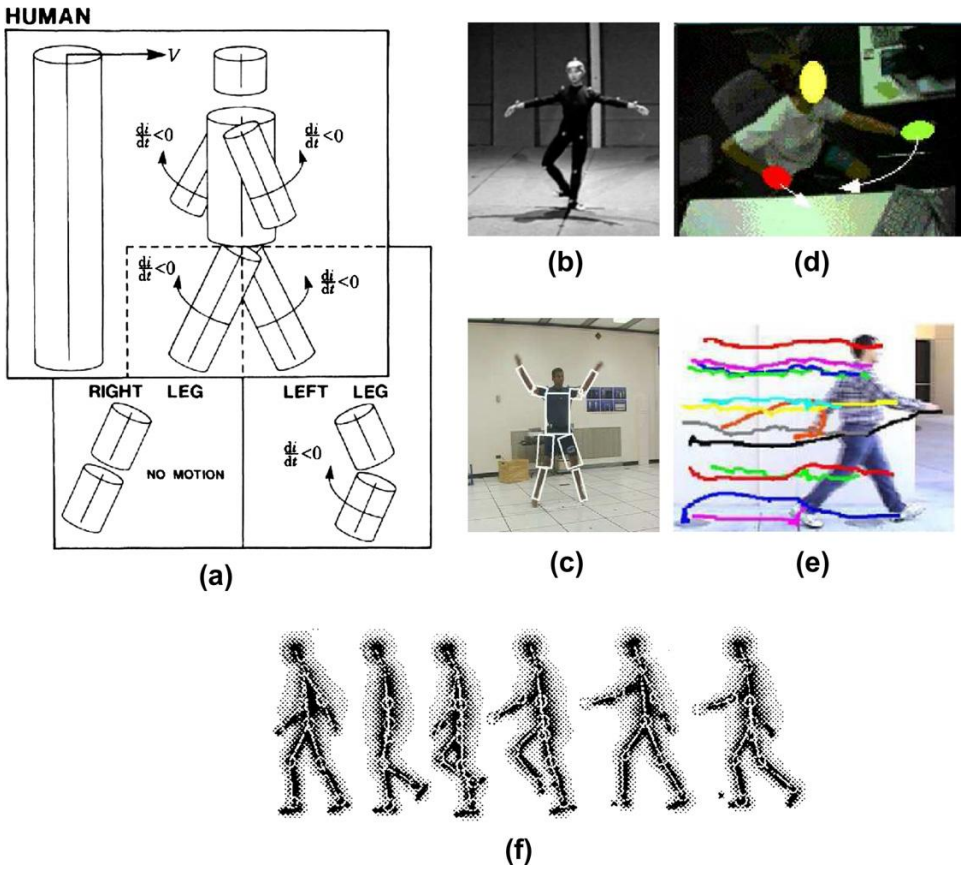

Figura 2.3: Modelli di rappresentazione: (a) Modello gerarchico 3D basato su cilindri. (b) Ballerino con appositi marker attaccati al corpo. (c) Modello basato su forme rettangolari. (d) Modello Blob. (e) Marker di traiettorie 2D. (f) Stick figures.

Image Models

In questa sezione verrà descritto il processo di rappresentazione delle azioni basato su immagini. Tale tipo di approccio non richiede necessariamente la differenziazione delle diverse parti del corpo. Lo step principale consiste nel trovare la *ROI*[2], centrata attorno al soggetto. Nella maggior parte dei casi, le *features* vengono estratte ed elaborate tramite una griglia geometrica che ricopre interamente il modello. In Figura 2.4 sono rappresentati alcuni esempi di tali modelli.

Questi tipi di modello possono risultare molto più semplici di quelli parametrici del corpo, di conseguenza il loro processo di elaborazione avviene in modo più veloce ed efficiente. Un esempio tipico è stato presentato da Darell e Pentland [9], dove le immagini di gesti eseguiti con la mano sono strettamente correlate senza l'estrazione di *features*. Questo tipo di approccio tuttavia prevede come assunzione uno sfondo statico di colore nero.

Una prima classe di modelli basati su immagini utilizza le sagome ed i contorni del soggetto che compie l'azione. Esempio di questo tipo di approccio si trova in Yamato [33]: i pixel delle immagini vengono raggruppati in *super-pixel* in cui le *features* sono il rapporto tra i pixel bianchi e quelli neri. Una rappresentazione simile è utilizzata in [11](Figura 2.4a e 2.4b). In [42] le sagome vengono invece integrate durante delle sequenze temporali in immagini chiamate *MHI*[3] e *MEI*[4], com in Figura 2.6a. Un'altra via è invece quella di integrare una sequenza temporale in un'unica immagine, per lavorare successivamente sul volume ricoperto dalla sagoma durante tale sequenza [13] (Figura 2.6c).

Utilizzare la sagoma del soggetto come *feature* da una parte presenta un grosso vantaggio perché è assolutamente indipendente dal colore, dalle texture e dal contrasto, ma dall'altra parte è sconveniente quando si ha a

[2]Region Of Interest
[3]Motion History Image
[4]Motion Energy Images

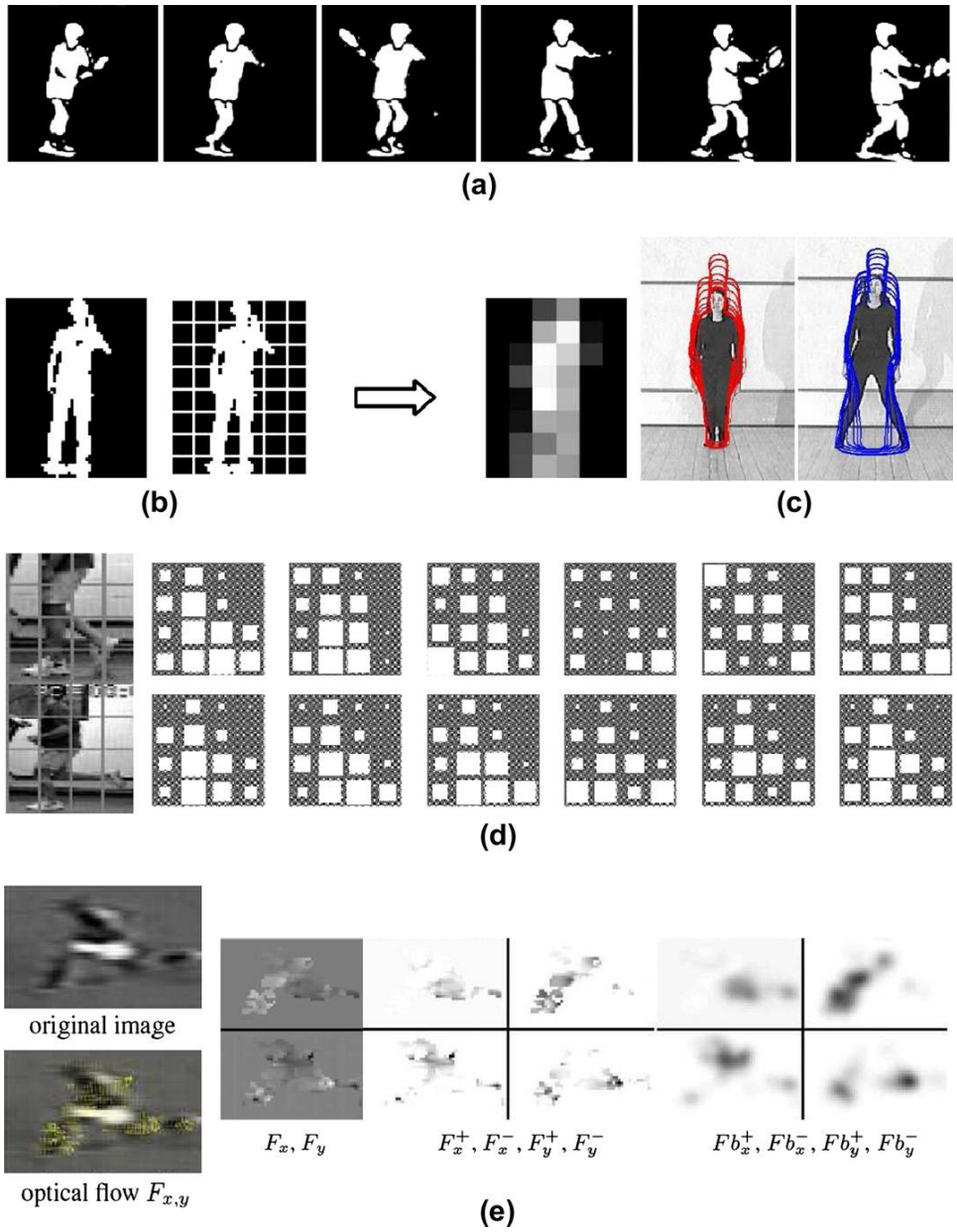

Figura 2.4: Rappresentazione globale: (a) Sagoma di un giocatore di tennis. (b) Pixel di una sagoma suddivisi in griglia. (c) Contorni di un'azione. (d) Flusso visivo magnetico rappresentato in griglia. (e) Flusso visivo diviso in componenti direzionali di movimento.

che fare con sfondi difficilmente identificabili ed è quindi richiesto un robusto processo di segmentazione e separazione dello sfondo.

Un'altra importante classe di modelli basati su immagini utilizza invece il flusso visivo estratto tra frames consecutive. Un primo esempio di questo tipo di approccio si trova in [14], dove viene eseguita una computazione temporale delle textures, per riconoscere eventi come ad esempio il movimento degli alberi provocato dal vento. Sempre Polana in collaborazione con Nelson [15] propongono invece *features* per il riconoscimento di azioni basate sui flussi magnetici accumulati e visualizzati in una griglia, come rappresentato in Figura 2.4d. Un altro tipo approccio è stato proposto da *Cutler* e *Turk* [16], che suddivisero il flusso in un insieme di diversi *Motion Blobs*. Successivamente la posizione di questi, il loro movimento e la loro dimensione vennero utilizzati come *features*.

Un'altra soluzione, come descritto in [18], è quella di suddividere le immagini in 4 componenti direzionali: due orizzontali e due verticali. Questi componenti vengono poi elaborati separatamente (Figura 2.4e).

Un'ulteriore classe di modelli di questa classe è quella basata sui gradienti. Come descritto in [19], vengono elaborati i diversi gradienti nelle componenti XYZ ed ogni frame viene rappresentato poi attraverso un istogramma. Anche il descrittore *HOG*[5], che venne applicato con successo al riconoscimento di persone ed oggetti, venne utilizzato per l'*Action Recognition*, come descritto in [20]. Invece di elaborare un singolo gradiente per ogni frame, mediante il descrittore *HOG*, l'immagine viene divisa in blocchi regolari e viene poi elaborato un istogramma per ognuno di questi blocchi.

Così come i flussi visivi, i gradienti condividono caratteristiche quali la indipendenza dall'estrazione dello sfondo, ma sono fortemente sensibili alle proprietà dei materiali, delle texture e delle luci. Sono invece discriminativi per quanto riguarda le parti in movimento e quelle statiche, e questo presenta

[5]Histogram of Oriented Gradients

sia vantaggi che svantaggi. Ad esempio le parti non in movimento possono rappresentare importanti informazioni per le azioni, ma possono anche essere facilmente confusi con oggetti statici sullo sfondo. Lavori recenti hanno dimostrato risultati migliori mediante la combinazione di flussi e gradienti [21], oppure sagome e flussi [22].

L'ultima classe di modelli analizzata è quella basata su un approccio neuroscientifico chiamato *HMAX* [23]. Ad esempio l'approccio descritto in [24] combina insieme i filtri di *Gabor* ed i flussi visivi in un unico grande schema, in modo da simulare le funzioni di risposta agli stimoli base della corteccia visiva. Utilizzando filtri di *Gabor* ancora di più alto livello si possono certamente ricavare informazioni più dettagliate, ma sicuramente a discapito del tempo computazionale.

Le rappresentazioni basate su immagini sono state utilizzate in numerosi approcci di tipo diverso, tuttavia molti di questi partono sempre dal presupposto che verranno migliorate in futuro grazie al progresso tecnologico. In particolare molti metodi assumono che per trovare la *ROI* attorno ad una persona, con possibilmente differenziato lo sfondo, viene prima prevista una precedente fase di pre-elaborazione della scena. Di conseguenza questi approcci dipendono fortemente dal progresso negli specifici campi come ad esempio il rilevamento e il tracking di una persona.

Altri approcci invece lavorano su soggetti completamente visibili e non si pongono il problema di come adattare delle osservazioni parziali dei soggetti. Da notare che applicazioni di video sorveglianza si adattano ottimamente a queste assunzioni in quanto solitamente i soggetti registrati si trovano ad una certa distanza.

Statistiche nello spazio

In questa sezione si approfondiranno le rappresentazioni locali di azioni che utilizzano un approccio basato sulla divisione del video/immagini di input

in piccole regioni, indipendenti dalle parti del corpo o dalle coordinate delle immagini.

Le *features* locali possono essere elaborate suddividendo le immagini in densi o sparsi insiemi di regioni diverse. I punti di interesse *spazio-temporale* [25] furono introdotti per generalizzare i punti di interesse ed i descrittori locali, già precedentemente utilizzati per il riconoscimento di oggetti. Tali tipi di approccio si basano principalmente sull'identificazione dei punti di interesse nell'immagine, assegnando ogni regione ad un insieme di cosiddetti *features-vocaboli*. Il processo di classificazione delle immagini si occupa poi di ridurre queste regioni in cosiddetti *BOF*[6]. Acluni esempi di questo tipo di descrittori sono descritti in [25] e in [26] (Figura 2.5).

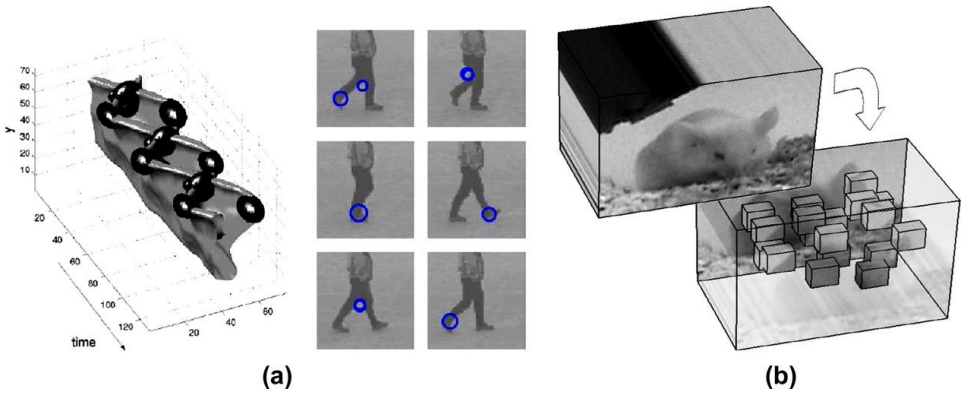

Figura 2.5: Rappresentazione locale: (a) Punti di interesse spazio-temporale [25]. (b) Features spazio-temporali [26].

Uno dei vantaggi principali di questo tipo di approccio è che il rilevamento dell'agente non richiede necessariamente la computazione delle *features* spazio-temporali. I punti di interesse rilevati devono presentare però qualche consistenza tra osservazioni simili. Tuttavia le *features* rilevate sono solitamente disordinate e di diverse dimensioni, di conseguenza risulta difficile applicare un modello geometrico ed una struttura temporale.

[6]Bag Of Features

Per aggiungere ulteriori informazioni riguardo alla struttura alcuni approcci, come ad esempio quello descritto in [27], utilizzano a tal scopo dei modelli grafici con variabili nascoste per la posizione delle regioni. Nel metodo illustrato in [28] vengono introdotte le cosiddette *Compound Features*, che possono essere viste come una sorta di *super-features* che considerano la relativa posizione delle *features* nelle vicinanze. Un'altra possibilità per aggiungere informazioni di tipo strutturale è quella di suddividere l'immagine in diversi *istogrammi BOF*, cosi come descritto in [29].

Nonostante la maggior parte delle rappresentazioni locali di *features* sono basate sull'estensione *SIFT*[7], anche altri tipi di approccio sono stati proposti. Come descritto in [30], le zone locali vengono elaborate mediante una segmentazione basata sul colore del volume spazio-temporale. Le relazioni spaziali tra i segmenti risultanti che invece vengono perse in un primo momento verranno poi elaborate tramite strutture grafiche, come illustrato in [31]. Tali strutture saranno poi utilizzate per confrontare le azioni.

In conclusione, i metodi statistici basati su *features locali* hanno da sempre suscitato grande attenzione in quanto promettono grandi vantaggi soprattutto nel riconoscimento di oggetti statici, ed anche perché possono essere applicati in modo abbastanza semplificato a scene complesse, come ad esempio video provenienti da internet. Tuttavia, la complicata natura delle azioni e varietà dei comportamenti umani renderà sicuramente necessaria l'integrazione di questi metodi con modelli spaziali e/o temporali. La combinazione di questi differenti modelli verrà descritta nella sezione successiva.

2.1.2 Temporal Action Representations

In questa sezione verranno descritti alcuni dei metodi possibili per ricavare la struttura temporale delle azioni estrapolando le informazioni dalle *features*

[7] Scale Invariant Feature Transform

estratte. Questi tipi di approccio vengono classificati in base alla tipologia di componente temporale utilizzata nelle osservazioni.

Si distinguono dunque tre principali categorie: *grammars*, *templates* e *statistiche temporali*.

Action Grammars

L'approccio ivi descritto rappresenta un'azione come una sequenza di momenti, ognuno identificato da una propria dinamica. Una soluzione comune in questo caso è quella di raggruppare le *features* in configurazioni simili, come ad esempio gli stati, ed analizzare quindi le transizioni temporali tra questi.

Il modello maggiormente utilizzato è sicuramente quello basato sul *HMM*[8] [32]. Questo metodo è diventato molto popolare soprattutto per il suo grande successo nel riconoscimento e nell'elaborazione del linguaggio naturale.

Il primo lavoro che ha utilizzato il modello *HMM* risale a *Yamato* [33], dove un *HMM* discreto viene usato per rappresentare delle sequenze di vettori che identificano la sagoma di un giocatore di tennis, come mostrato in Figura 2.4a. *Wilson* e *Bobick* [34] utilizzarono invece il modello *HMM* per riconoscere i gesti di una mano.

Vi poi sono numerosi altri approcci che utilizzano tale modello: ad esempio *Brand* [35] descrive come applicare tale metodologia in un unico spazio e mappato in un altro. *Wang* [36] propone invece di misurare la distanza tra *HMMs* per classificare in *cluster* i gesti in modo non supervisionato.

Analizzando questo tipo di approccio, risulta chiaro che una limitazione dei modelli basati su *HMM* è il fatto che rappresentano *modelli generativi* di azioni e si basano su modelli statistici semplificati per elaborare la probabilità delle posizioni assunte dai giunti degli stati e delle *features* osservate, mentre un modello discriminativo più generale può predire la probabilità condizionata degli stati date le *features* osservate. Come risultato, molti

[8] Hidden Markov Model

autori hanno lavorato sull'utilizzo di modelli discriminativi nel processo di riconoscimento delle azioni.

Sminchisescu [38] nel suo lavoro descrive l'utilizzo di *CRF*[9] al posto dei *HMM*. I *CRF* sono modelli di Markov discriminativi che possono usare *features* non indipendenti ed osservazioni nel corso del tempo (contrariamente alle assunzioni dei *HMM*). Inoltre, i parametri dei *CRF* possono essere allenati per massimizzare la potenza discriminativa del classificatore.

Altre tipologie di modelli dinamici furono utilizzati per il riconoscimento di azioni: modelli auto-regressivi come descritti in [39], oppure anche reti neurali traslate nel tempo illustrate in [40].

In conclusione, un importante punto di forza delle *Action Grammars* è sicuramente il loro alto grado di modularità. Tale peculiarità li rende scalabili ed adattabili a grandi variazioni dovute al cambiamento di stile e velocità delle azioni. I parametri delle grammatiche probabilistiche possono essere ricavati in modo piuttosto efficiente utilizzando un piccolo numero di esempi etichettati in modo supervisionato, oppure un grande numero di esempi non-etichettati in modo non-supervisionato. In generale comunque la struttura di tali grammatiche viene scelta manualmente, come ad esempio in [41].

Come risultato, la valutazione delle *Action Grammars* con un grande numero di classi di azioni rimane un problema in sospeso.

Action Templates

Invece di estrarre *features* dinamicamente ed in modo separato, alcune tipologie di modelli tentano di apprendere direttamente mediante l'utilizzo di interi blocchi temporali di *features*, definiti *templates*. Tipicamente questo tipo di approccio rappresenta le dinamiche di un'azione direttamente attraverso sequenze di esempi, sia raggruppando diverse *features* in un unico singolo vettore di *features*, sia estraendo le *features* da un volume

[9] Conditional Random Fields

spazio-temporale n-dimensionale durante un prestabilito arco di tempo, come mostrato in Figura 2.6. La maggior parte dei metodi basati su *template* utilizzano come base di partenza dei modelli basati su immagini, come ad esempio descritto in [13], che forniscono un *template* costruendo una singola rappresentazione volumetrica partendo da diverse singole immagini. Oltre a questi vi sono anche modelli parametrici come descritti in [7] oppure basati su rappresentazioni locali come descritto invece in [30].

I *template* vengono tipicamente elaborati attraverso una sequenza di frame, e non devono essere confusi con le *features spazio-temporali*, le quali invece utilizzano piccole finestre sul tempo (solitamente 2-4 frames). *Bobick* e *Davis* [42] implementarono nel loro lavoro delle *MHI*[10] mappando frame sequenziali di una sagoma in un'unica singola immagine, cosi come mostrato in Figura 2.6a. Diverse estensioni di *MHI* furono proposte (Figura 2.6b). Le *MHI* furono anche estese in *MHV*[11] [43] (Figura 2.6b), usando a tal scopo invece delle sagome 2D dei gusci volumetrici formati durante le sequenze dal movimento delle diverse parti del corpo.

I *template* solitamente costituiscono rappresentazioni di vettori di dimensione fissa, il che li rende facilmente implementabili in combinazione con la maggior parte delle tecniche di classificazione statica. A differenza degli approcci descritti precedentemente tuttavia non permettono di rappresentare le variazioni di velocità o stile durante l'esecuzione di un'azione.

Per poter affrontare il problema delle azioni con durata temporale variabile, uno step di normalizzazione addizionale è richiesto per assicurare che il vettore risultante abbia le stesse dimensioni, oppure è possibile utilizzare un metodo più avanzato come il *DTW*[12], descritto in [9].

Al posto di utilizzare diversi *template* con un classificatore convenzionale, *Rodriguez* [44] propose di costruirne uno unico da un insieme di *template*

[10] Motion History Image
[11] Motion History Volumes
[12] Dynamic Time Warping

usando un filtro di tipo *MACH*.

Altri importanti esempi di *Action Tempates* utilizzano *Fourier* o una rappresentazione ad onda nel dominio temporale, come descritto in [14]. Anche le traiettorie dei giunti del corpo o le *features* estratte delle immagini possono costituire *template*. Ad esempio, *Lv* e *Nevatia* [45] introducono *template* basati sulle traiettorie delle diverse parti del corpo durante una sequenza temporale.

Tale tipo di approccio, basato sulle traiettorie descritte dai giunti del corpo durante una sequenza temporale, costituisce il modello utilizzato per il caso di studio trattato in questa tesi.

In conclusione, le rappresentazioni basate su *template* sono state proposte in numerosi metodi. Generalmente risultano efficienti e discriminativi nella rappresentazione delle azioni, e molto attrattivi in quanto permettono di integrare potenti classificatori statici come ad esempio SVM[13]. Dall'altra parte peccano per la mancanza di metodi efficienti di segmentazione temporale e non gestiscono il problema delle osservazioni parziali o incomplete.

Statistiche temporali

In contrasto con gli approcci descritti precedentemente, alcuni tentano di costruire dei modelli statistici delle azioni, senza necessariamente avere un modello esplicito della loro dinamica. Tipici esempi sono quei metodi che permettono di ricavare dei modelli di azioni da un singolo significativo *key-frame* come avviene nel campo della fotografia [37], oppure trovare un modello utilizzando le informazioni proveniente dagli istogrammi costruiti dalle *features* estratte durante la sequenza.

Carlsson e *Sullivan* [46] nel loro lavoro introdussero l'uso dei *keyframe*, ad esempio per distinguere il dritto ed il rovescio del tennis.

[13] Support Vector Machine

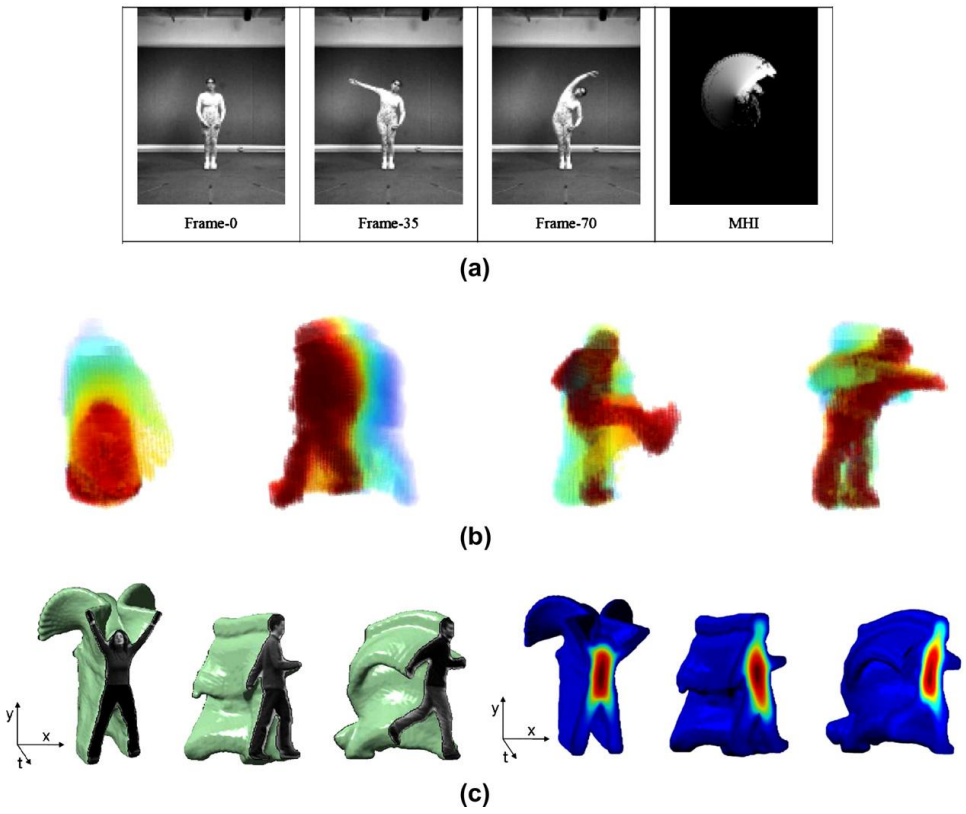

Figura 2.6: Action Template: (a) Motion Histoty Images. (b) Motion History Volumes. (c) Space-Time Shapes.

Tecniche basate su *Temporal Bag Of Features* furono utilizzate per rappresentare delle sequenze temporali semplicemente basandosi sulla frequenza delle *features* durante il lasso di tempo, cosi come descritto in [47].

Un'estensione dei *Temporal BOF* basato sul *Temporal Binning* viene proposto invece nel lavoro descritto in [48], per provvedere esplicitamente a trovare l'ordinamento nel breve tempo delle *visual words*. Anche correlogrammi spazio-temporali furono proposti per aggiungere dipendenze spaziali e temporali ai metodi basati su *BOF*, così come descritto in [17].

Naturalmente, un approccio di questo genere non può essere discriminativo rispetto ad ogni tipo di azione, basta considerare ad esempio due azioni che coinvolgono pose simili ma in ordine diverso. Tuttavia, vi è un interesse sempre crescente in questo metodo di rappresentazione, in particolare nell'approccio *BOF*. Tale attenzione è dovuta principalmente al recente trend nell'adottare *features locali*, che permettono di ottenere risultanti migliori nonostante la semplificata classificazione. Anche nel caso di piccole azioni atomiche i modelli in questione presentano notevoli proprietà, come l'efficienza di computazione insensibile e la forte discriminazione alla variazione della scala di tempo, aspetti che li rendono molto interessanti anche per quanto riguarda il riconoscimento di azioni su larga scala.

2.1.3 Action Segmentation

Una domanda a cui è importante trovare risposta è se sia veramente possibile distinguere ed eseguire separatamente i processi di *Action Segmentation* e *Action Recognition*. Da un punto di vista computazionale, risulta certamente più utile dividere inizialmente il flusso video prima di applicare il processo di riconoscimento, visto che assegnare un'etichetta ad una singola sequenza è sicuramente più efficiente che etichettare tutte le sotto-sequenze del flusso in questione. Ma questo rende difficile trovare un generico vocabolario per classificare le azioni, ed un generico metodo per suddividere il video nelle

corrispettive sequenze. In pratica questo risulta essere un problema tanto difficile quanto quello del riconoscimento delle azioni vero e proprio.

Verranno ora descritti diversi metodi di segmentazione temporale, suddivisi in tre classi principali: *Boundary Detection*, *Sliding Windows* e *Grammar Concatenation*.

Boundary Detection

Una strategia comune per riconoscere le azioni è quella di utilizzare un generico metodo di segmentazione basato sul rilevamento dei contorni descritti dal moto, e successivamente classificare i segmenti risultanti in modo separato.

Marr e *Vaina* [49] nel loro lavoro affrontarono il problema della segmentazione 3D del movimento del corpo umano, proponendo come soluzione l'utilizzo di particolari stati, come ed esempio i minimi locali, del moto 3D descritto dagli arti come una naturale transizione tra movimenti base. *Rubin* e *Richards* invece definirono nel loro lavoro [51] due tipi elementari: *starts* e *stops*, analoghi agli stati descritti da *Marr* e *Vaina* in [49], e *dynamic boundaries* risultanti invece dalla discontinuità.

In linea teorica, i metodi di *boundary detection* risultano attrattivi in quanto permettono una segmentazione generica del video, indipendentemente dalle classi di azioni studiate. In pratica, tale processo deve essere applicato con le opportune precauzioni in quanto è soggetto innanzitutto ad errori nel rilevamento del campo del moto, ed in secondo luogo perché non sono stabili col cambiamento dei punti di vista. Inoltre spesso possono essere confusi a causa della presenza di movimenti multipli e simultanei.

Sliding Windows

Un'altra strategia per riconoscere le azioni consiste nel suddividere le sequenze video in più sequenze sovrapposte utilizzando delle *Sliding Windows* (finestre a scorrimento). La classificazione avviene in modo sequenziale su

tutte le sequenze candidate, ed i picchi dei punteggi delle classificazioni risultanti vengono interpretati come la posizione delle azioni. A differenza del *Boundary Detection*, in questo caso la segmentazione è strettamente legata al riconoscimento della scena. Risulta quindi chiaro che tali metodi non possono essere applicati in una scena sulla quale viene eseguito il processo di *training*.

L'approccio delle *Sliding Windows* può essere utilizzato con qualsiasi rappresentazione di *features* e classificatori illustrati precedentemente. Molti sistemi basati su *template* usano questo approccio, come ad esempio descritto in [19]. Alcuni metodi inoltre adottano tale tipo di approccio in combinazione con DTW [9] o con grammatiche [50].

Rispetto al *Boundary Detection*, il metodo delle *Sliding Windows* è generalmente più intensivo dal punto di vista computazionale, in quanto coinvolge diverse valutazioni di tutti i classificatori. Inoltre talvolta vengono richieste finestre di dimensioni diverse, e questo non fa altro che aumentare il carico computazionale. Può anche accadere che si ottengano risultati impredicibili in presenza di categorie di azioni sconosciute. Tuttavia, tale metodo prevede meno assunzioni e può essere facilmente integrato con un qualsiasi classificatore senza la necessità di ulteriore segmentazione di *features*.

Grammar Concatenation

Precedentemente è stato presentato un metodo di rappresentazione di classi di azioni individuali mediante l'utilizzo di grammatiche. Questo suggerisce di adottare un approccio generale per la suddivisione di azioni mediante il *concatenamento* delle grammatiche per ottenere un modello delle transizioni tra le diverse azioni. Le grammatiche concatenate possono essere ottenute ad esempio unendo tutti i modelli in un unico nodo di inizio e di fine ed aggiungendo un loop tra questi due nodi. Anche azioni più complesse pos-

sono essere modellate dalle grammatiche concatenate. La segmentazione e il labeling di complesse sequenze di azioni vengono poi elaborate come un percorso a costo minimo nella rete utilizzando le tecniche di programmazione dinamica, ad esempio l'algoritmo di *Viterbi* per *HMM* [32].

Con questo tipo di approccio la segmentazione risulta elegante ed efficiente grazie proprio alle tecniche di programmazione dinamica. Bisogna tuttavia tenere presente che implementare una grammatica concatenata con molte azioni richiede un maggior processo di *training* dei dati. Nel campo del riconoscimento vocale questi dati sono presenti sotto forma di documenti di testo, trascrizioni di parole etc. Tali dati invece non esistono per quanto riguarda il riconoscimento di azioni, e spesso anche le transizioni tra azioni vengono definite manualmente oppure sotto forti assunzioni, come ad esempio assegnare la stessa probabilità ad ogni possibile transizione.

2.1.4 View-independent Action Recognition

Come descritto precedentemente, le considerazioni fondamentali sulla rappresentazione di modelli, sia 2D che 3D, hanno una lunga storia nel campo dell'*Action Recognition*, e i diversi tipi di approccio mettono in evidenza ed a confronto le caratteristiche di entrambe le metodologie.

Verranno presentati in questa sezione tre metodi principali per il confronto di tipo *view-independent*: *Normalization*, *Invariance* e *Exhaustive Search*. Le seguenti metodologie verranno suddivise a seconda che il campo di applicazione sia quello 2D o 3D.

View Normalization

Durante questa fase, ogni osservazione viene mappata in un sistema canonico comune di coordinate in un frame. Tale processo inizialmente si occupa di valutare i segnali che identificano la trasformazione dalla vista canonica dei frame a quella corrente, e successivamente di correggere le osservazioni

in accordo con le trasformazioni stimate. Il loro confronto avviene nelle coordinate normalizzate dei frame.

Normalizzazione 2D Si tratta di un pre-processo mediante il quale vengono rimosse le variazioni causate dal ridimensionamento e dalla traslazione. In particolare i modelli basati su immagini estraggono spesso una *ROI* rettangolare attorno al soggetto corrente, traslando e ridimensionando poi questa regione in un unico frame. *Rogez* [52] nel suo lavoro propose un metodo che stima l'orientamento 3D di una persona in base alla direzione in cui la stessa persona cammina nel piano 2D, utilizzando le informazioni riguardanti l'omografia e la calibrazione della videocamera. Assumendo solo una possibile rotazione orizzontale del corpo nello spazio 3D, la sagoma 2D della persona viene rispettivamente riportata in una vista frontale e parallela, per essere poi confrontata con l'insieme delle sagome canoniche nel data-set.

Normalizzazione 3D Dato un modello 3D, una rappresentazione indipendente dalla rotazione dei giunti può essere ricavata basandosi sull'orientamento globale del corpo. Solitamente il torso viene utilizzato come punto di riferimento e tutti i giunti vengono normalizzati ed orientati in accordo con l'orientamento del corpo. Un'altra via possibile è quella di rappresentare ogni giunto con un proprio sistema di coordinate individuale.

In conclusione, il processo di normalizzazione si basa principalmente sull'orientamento del corpo. In alcuni casi, come ad esempio la camminata, questa informazione può essere immediatamente ricavata.

View Invariance

Questo processo non si propone di valutare le variazioni dei punti di vista tra osservazione e modello, ma ricerca le *features* e confronta le funzioni che sono indipendenti dalle variazioni dei punti dei vista considerati.

View Invariance nel 2D Un semplice approccio è quello basato sugli istogrammi: invece di rappresentare le *features* estratte dalle immagini in una griglia geometrica fissa, viene salvata solo la frequenza di tali *features*. Il confronto tra punti corrispondenti venne frequentemente utilizzato per il confronto di tipo invariante tra coppie di osservazioni, come rappresentato in Figura 2.7. Invarianze di tipo geometrico possono essere ad esempio ottenute da 5 punti disposti su un piano, cosi come illustrato in Figura 2.7a [53]. Un approccio invece più recente che utilizza le corrispondenze tra punti o tra *features* viene descritto in [54], dove viene mostrato come le *features* rimangono stabili nonostante il cambiamento del punto di vista.

View Invariance nel 3D *Campbell* [55] e i suoi collaboratori studiarono 10 differenti rappresentazioni *view-invariant* basate sulle traiettorie 3D descritte dai giunti del corpo durante un'azione. *Cohen* e *Li* [56] proposero una rappresentazione basata su istogrammi 3D di forma cilindrica. *Weinland* [43] nel suo lavoro utilizzò invece una rappresentazione in 3D, basata su coefficienti di *Fourier* in coordinate cilindriche.

In conclusione, questo tipo di approccio rimuove la dipendenza dai punti di vista durante l'estrazione delle *features*. Elaborare una singola *feature* indipendente per ogni azione è sicuramente più efficiente che considerare tutti i possibili diversi punti di vista, ed inoltre non dipende da possibili errori dovuti all'orientamento.

Exhaustive Search

Invece di considerare una sola trasformazione, come avviene tipicamente per i metodi di normalizzazione, si può scegliere di cercare tra tutte quelle possibili.

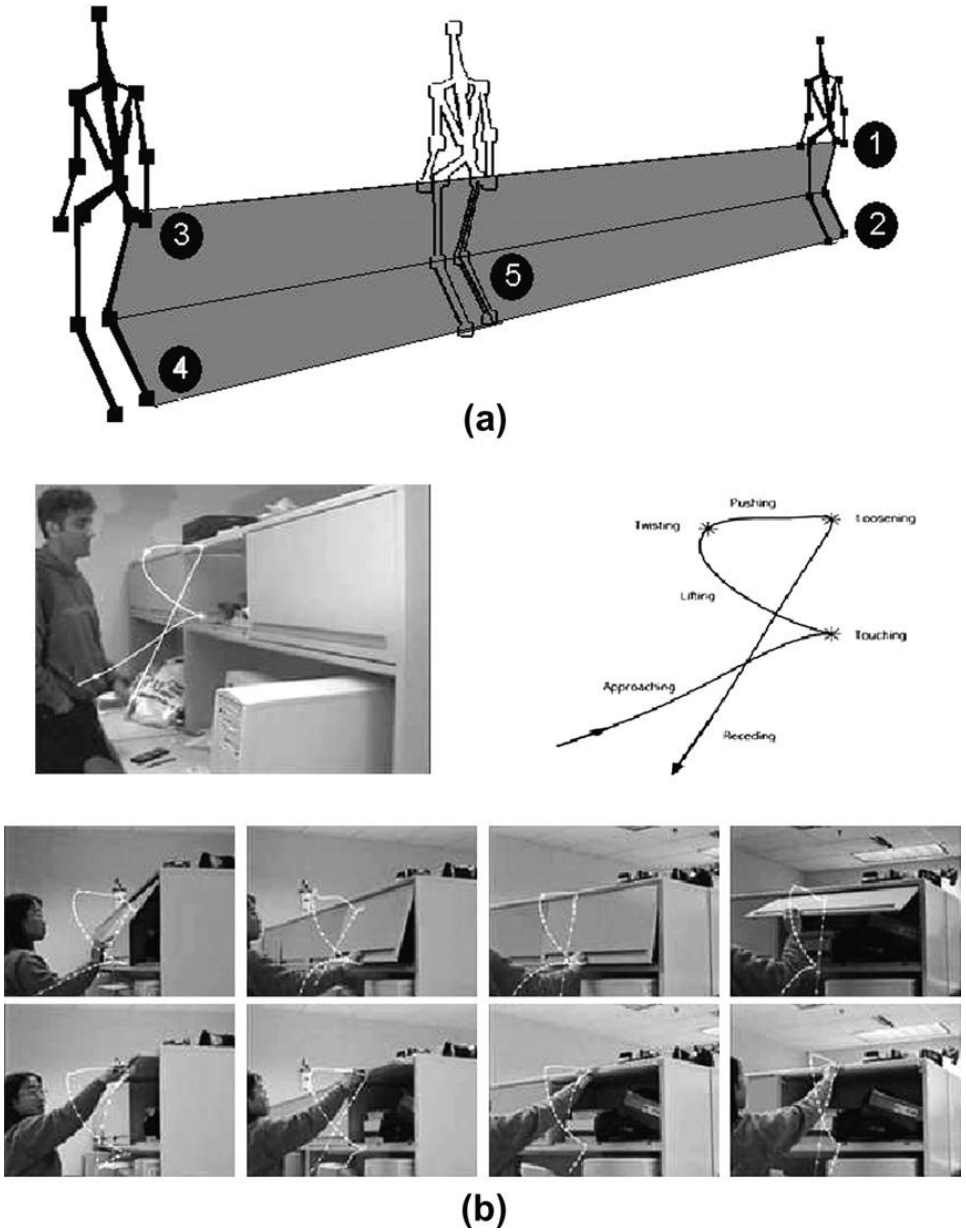

Figura 2.7: Metodi di invarianza: (a) Invarianza di tipo geometrico. (b) Invarianza tra le traiettorie descritte dal movimento di una mano.

Ricerca esaustiva con viste 2D Vengono utilizzate diverse camere fisse poste attorno al soggetto e registrata l'azione da diversi punti di vista. Il confronto avviene poi attraverso ogni registrazione salvata. Nel loro lavoro con *MHIs Bobick* e *Davis* [42] registrarono le azioni con diverse videocamere, ognuna sfalsata di 30 gradi rispetto alle altre sul piano orizzontale. Durante il processo di riconoscimento vengono utilizzate due videocamere sfalsate di 90 gradi, confrontando con tutte le coppie di registrazioni con gli stessi 90 gradi di sfalsamento. In modo analogo *Ogale* [57] e *Ahmad* e *Lee* [58] utilizzarono otto viste pre-registrate, ed una sola durante il processo di riconoscimento.

Ricerca esaustiva con modello 3D Per gestire in modo più flessibile le variazioni del set-up delle videocamere, è possibile utilizzare un modello basato su rappresentazione 3D. Da qualsiasi osservazione 3D, dati i parametri delle videocamere, si può ottenere un'osservazione nel piano 2D. Negli approcci di tipo *generativo* usati in *MOCAP*, modelli 3D parametrizzati vengono proiettati in modelli 2D. Questi modelli presentano esplicite variabili per la posizione e orientamento 3D globale, che vengono simultaneamente stimate con i restanti parametri di giunto, come mostrato in Figura 2.8.

Metodi simili furono proposti per l'*Action Recognition*, ad esempio l'approccio descritto in [59] utilizza un piccolo insieme di posizioni chiave in 3D, successivamente portate in 2D mantenendo il più possibile l'invarianza alle trasformazioni. Invece di utilizzare un approccio che produce le proiezioni nel 2D, *Souvenir* e *Babbs* [60] utilizzano direttamente una funzione analitica che prende in input la vista 3D e restituisce in output la corrispondente rappresentazione della sagoma nel piano 2D.

In conclusione, tali tipi di approccio hanno recentemente riscontrato sempre più interesse, soprattutto perché il loro costo computazionale diventa sempre meno pesante grazie al progresso tecnologico. Nessuno dei due dipende dall'orientamento del corpo, tuttavia dipendono fortemente dalla presenza di un data-set con registrazioni etichettate da diversi punti di vista, e solitamente richiedono assunzioni sullo spazio di ricerca, ad esempio restrizioni dovute a certe classi di trasformazione del punto di vista.

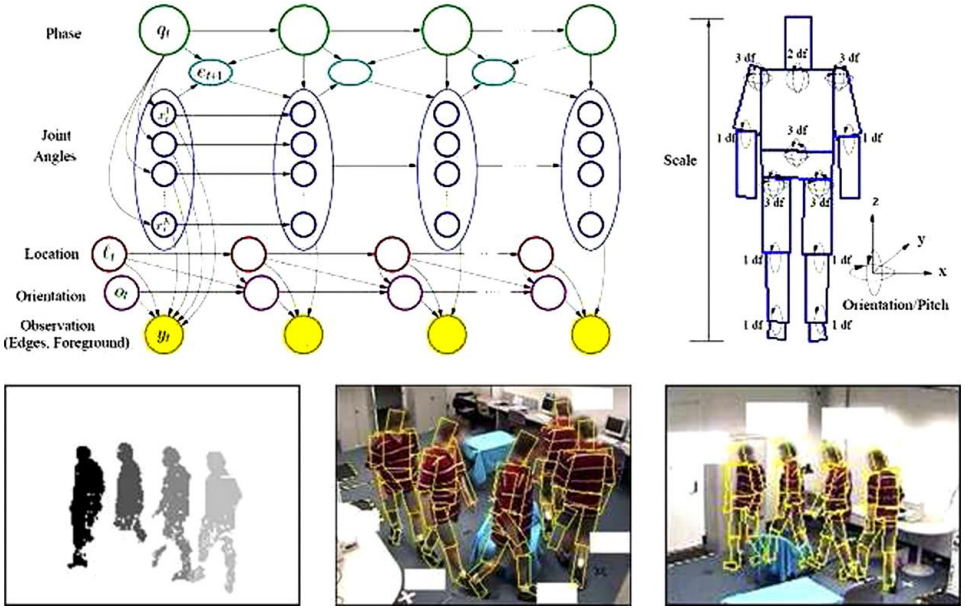

Figura 2.8: MOCAP generativo.

2.2 Applicazioni e dispositivi

In questa sezione verranno brevemente descritte alcune delle applicazioni principali ed i dispositivi di input utilizzati per il *Gesture Recognition*.

Tra le applicazioni principali in questo ambito:

Riconoscimento del linguaggio dei segni Così come il riconoscimento vocale può trascrivere parole in testo, vi sono alcune tipologie di software che permettono di riconoscere particolari gesti e trascrivere i sim-

boli rappresentati tramite il linguaggio dei segni in testo. Un esempio di questo tipo di applicazione è descritto in [61].

Robotica applicata all'assistenza sociale Si utilizzano sensori a doc (accelerometri, giroscopi) applicati al corpo di un paziente e leggendo i vari valori generati da tali sensori è possibile aiutare la riabilitazione dei pazienti.

Indicazioni direzionali attraverso il puntamento Identificare la direzione in cui una persona sta puntando può risultare molto utile ad esempio per identificare il contesto delle istruzioni, tale applicazione è di grande interesse soprattutto nel campo della robotica, come descritto in [62].

Controllo attraverso gesti facciali Si tratta di un'applicazione molto utile per gli utenti che non sono in grado fisicamente di utilizzare strumenti quali mouse o tastiera. Il tracciamento degli occhi può essere utilizzato per controllare il movimento del cursore. La nuova SDK 1.5 per *Kinect* implementa la possibilità di riconoscere le espressioni facciali.

Interfacce alternative per il computer Una nuova tipologia di interfacce basate su gesti ed azioni potrebbe sostituire quella tradizionale mouse-tastiera, utilizzando semplicemente una telecamera [63].

Tecnologia di gioco immersiva Esempio di questa applicazione è proprio *Kinect* di *Microsoft*, argomento principale di questa tesi.

Controller virtuali Utilizzare gesti come un meccanismo di controllo alternativo, ad esempio per dispositivi secondari come il televisore.

Applicazioni sportive Vi sono alcuni software utilizzati ad esempio nel tennis per rilevare la traiettoria della pallina e stabilire in questo modo la sua posizione rispetto alla riga. Un altro sport in cui vengono

utilizzati tali tipologie di software è il *Baseball* (Figura 2.9), come viene descritto in [64]. Tuttavia si tratta di un campo non molto diffuso, scopo di questa tesi è proporre un'applicazione del *Kinect* che venga utilizzato in ambito sportivo, soprattutto negli sport di combattimento, ma estendibile anche ad altri ambiti.

Figura 2.9: Baseball Scene Classification

I principali dispositivi di input che permettono il processo di *Gesture Recognition* sono telecamere a riconoscimento di profondità, per la stereovisione, controller remoti quali il *Wii Remote*, o anche telecamere singole. Tra i più importanti sicuramente il *Kinect*, che, nascendo come accessorio ludico per console, si è diffuso in numerosi campi, dalla robotica alla medicina. Tale strumento verrà approfondito nel capitolo successivo. Un altro accessorio importante è *Leap 3D*, uno strumento con maggior precisione del *Kinect* e costo minore ma non ancora in commercio. *Leap 3D* sarà invece descritto nel capitolo riguardante gli *Sviluppi Futuri*.

2.3 Kinect Skeleton Tracking

In questa sezione verrà approfondito in modo particolare l'algoritmo di *tracking* utilizzato dal *Kinect*, lo strumento oggetto di questa tesi.

Il tipo di approccio adottato si basa sostanzialmente sulla divisione degli oggetti in parti, cosi come descritto in [65] e [66]. L'obiettivo principale è focalizzato su due punti chiave: *efficienza computazionale* e *robustezza*. Come mostrato in Figura 2.10, una singola immagine di profondità in input viene

divisa a seconda della densità di probabilità di distanza dal sensore in parti del corpo diverse vicine ai punti di giunto di interesse. Proiettando i punti nello spazio vengono localizzati i modelli spaziali di ogni singola regione e si generano candidati *confidence-weighted* per la localizzazione nello spazio 3D di ogni giunto.

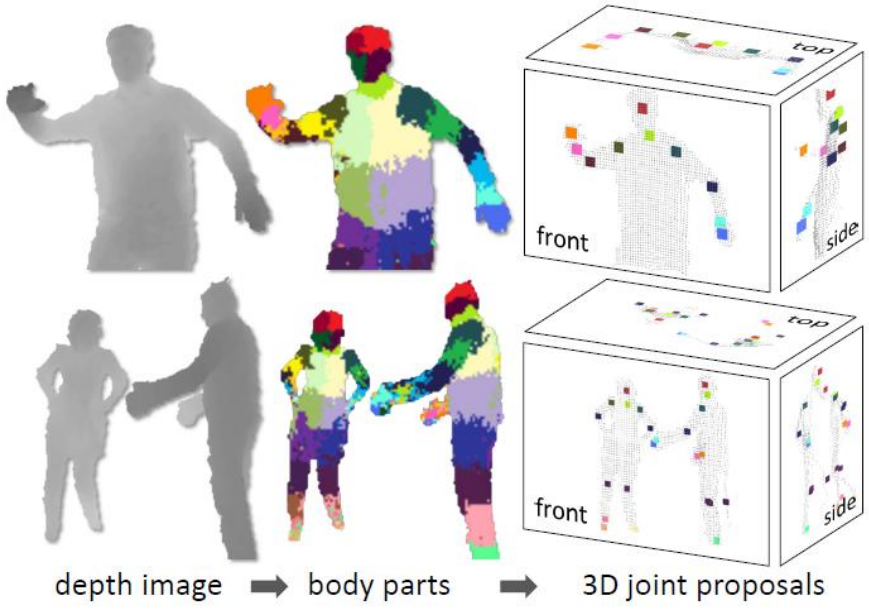

Figura 2.10: Kinect Algorithm

Si è implementato un classificatore di tipo *Random Decision Forest*, descritto in [67], in modo da evitare un eccessivo *overfitting* utilizzando a tal scopo un training-set di migliaia di immagini. Per garantire maggior efficienza il classificatore può essere eseguito in parallelo per ogni pixel sfruttando la potenza della GPU. La soluzione implementata ha permesso di raggiungere una velocità di 200 fps sulla *Xbox 360 GPU*, risultato di almeno un ordine di grandezza superiore a quelli attuali. Mediante tale tipo di approccio si è dimostrato come la localizzazione e riconoscimento di posizioni possa essere raggiunto con un più basso costo computazionale e maggior precisione rispetto alle metodologie attuali.

2.3.1 Data

In questa sezione si descriveranno i dati utilizzati per l'implementazione dell'algoritmo di *tracking* e quali benefici abbia portato la scelta di utilizzare *dati di profondità*, dati di *motion-capture(mocap)* ed un grande data-set di partenza.

Depth Data

La tecnologia basata su *depth images* ha fatto grandi progressi negli ultimi anni, raggiungendo un notevole risultato proprio con il lancio di *Kinect*. Le videocamere di profondità offrono diversi vantaggi rispetto ai sensori tradizionali: lavoro in scarsi livelli di luce, indipendenza da texture e colori, risoluzione di ambiguità del profilo durante le pose. Inoltre portano un grande miglioramento per quanto riguarda la separazione dello sfondo. L'aspetto più importante è però la possibilità di sintetizzare immagini realistiche di persone e questo permette di costruire un grande *training data-set* a basso costo.

Motion Capture Data

Il corpo umano è capace di eseguire una vastità di pose ed azioni difficili da simulare. Per questo si è deciso invece di costruire un enorme database di *mocap* di azioni umane, in modo da ricoprire nel modo più completo tutte le possibili azioni. Si tratta di un database con circa 500k frame che rappresentano centinaia di sequenze di azioni come la guida, il ballo, la corsa, etc. Visto che il classificatore non utilizza informazioni di tipo temporale, l'interesse è focalizzato esclusivamente sulle posizioni assunte e non sul movimento, infatti talvolta la variazione della posizione tra un frame e l'altro è talmente piccola che risulta assolutamente irrilevante. Bisogna quindi tralasciare molte posizioni inutili utilizzando il metodo del *Furthest Neighbor Clustering*, descritto in [70], dove la distanza tra le posizioni p_1 e

p_2 è definita come $max_j||p_1^j - p_2^j||_2$, cioè il massimo della *Distanza Euclidea* tra i giunti j. Si è utilizzato un sottoinsieme di 100k posizioni in modo tale che 2 posizioni non fossero più vicine di 5 cm.

Tutto questo processo si è reso necessario per iterare il processo di *Motion Capture*, allenare il classificatore, testare l'accuratezza delle posizioni dei giunti rilevate per raffinare il database con anche quelle posizioni che inizialmente sono state tralasciate. Il primo database ottenuto con questo tipo di approccio è il *CMU Mocap Database* [71].

Generalizzazione dati

Per poter assegnare un'etichetta alle immagini del campione è stato costruito un particolare filtro con due obiettivi principali: *realismo* e *varietà*. Infatti per rendere il modello reale i campioni devono essere vicini il più possibile alle immagini acquisite dalla videocamera ed inoltre ricoprire una gran parte delle variazioni che possono presentarsi durante i test. In Figura 2.11 sono rappresentate diverse posizioni ottenute tramite il processo di *training e test* e quelle acquisite direttamente dalla realtà.

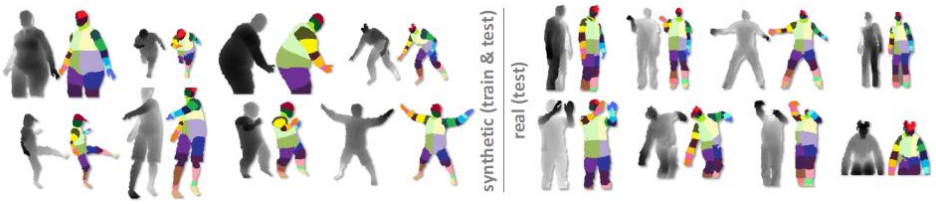

Figura 2.11: Dati reali e sintetizzati

2.3.2 Rappresentazione del corpo

In questa sezione verrà descritto il procedimento attraverso il quale viene ottenuta la ricostruzione delle diverse parti del corpo, l'estrazione delle *features* dalle informazioni proveniente dalla camera di profondità ed infine

l'applicazione del classificatore di tipo *Decision Forest* al riconoscimento di tali parti.

Body Part Labeling

La scelta di utilizzare una rappresentazione intermedia permette di trasformare il problema in uno che può essere risolto utilizzando efficienti algoritmi di classificazione.

Le diverse parti del corpo sono specificate in una *texture map* con l'obiettivo di "spellare" i soggetti durante le acquisizioni.

Depth Image Features

La feature ad un dato pixel x è dato dalla seguente formula:

$$f_\theta(I, x) = d_I\left(x + \frac{u}{d_I(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right) \tag{2.1}$$

dove $d_I(x)$ è la profondità al dato pixel x nell'immagine I ed i parametri $\theta = (u, v)$ descrivono i bilanci di u e v.

Ad esempio nella Figura 2.12 sono rappresentate due diverse *features* in diverse posizioni di pixel x. La *feature* $(f_\theta)_1$ è rivolta verso l'alto e l'equazione 2.1 restituirà dei valori alti per i pixel x vicini alla parte alta del corpo, ma valori vicini allo 0 nella parte inferiore. La *feature* $(f_\theta)_2$ aiuta invece ad identificare strutture come ad esempio gli arti.

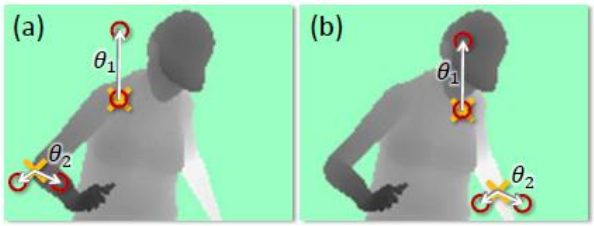

Figura 2.12: Depth Image Features

La scelta di questa tipologia di *features* è strettamente motivata dall'efficienza computazionale infatti: non è richiesto nessun processo di pre-elaborazione, per ogni *feature* servono al massimo 3 punti pixel e l'esecuzione di 5 operazioni aritmetiche, le *features* possono correttamente essere implementate sulla GPU. Avendo a disposizione un budget più elevato si possono implementare *features* più complesse, basate ad esempio su integrali, curvature o descrittori locali, come decritto in [26].

Randomized decision forests

Questo tipo di approccio ha permesso di ottenere classificatori più veloci ed efficienti, nonché facilmente implementabili sulla GPU. Come mostrato in Figura 2.13, una *forest* è un insieme di T alberi di decisione, ognuno formato da nodi foglia e nodi di decisione. Ogni nodo di ramificazione consiste di una *feature* f_θ ed una soglia τ.

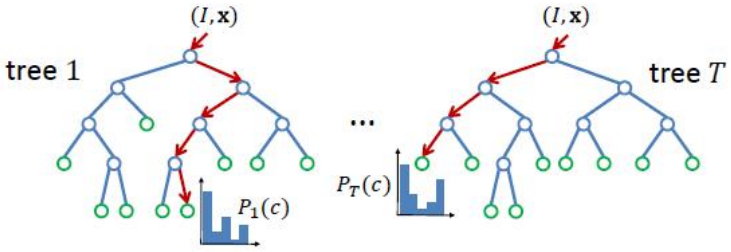

Figura 2.13: Randomized Decision Forests

Per classificare un pixel x nell'immagine I, si inizia dalla radice e si applica l'equazione 2.1 in modo iterativo, ramificando a destra o a sinistra in accordo col valore della soglia τ. Una volta raggiunto un nodo foglia nell'albero t, data la distribuzione conosciuta $P_t(c|I, x)$, l'etichetta c viene salvata. Viene calcolata poi la media delle distribuzioni di tutti gli alberi della *forest* per ottenere il classificatore finale:

$$P(c|I,x) = \frac{1}{T}\sum_{t=1}^{T}(P_t(c|I,x)) \tag{2.2}$$

Il processo di *training* di ogni albero avviene sulla base di un insieme di immagini casuali allenate e testate.

Posizione dei giunti

Il fine ultimo dell'algoritmo è quello di generare buoni candidati per rappresentare la posizione nello spazio 3D dei diversi giunti dello scheletro.

Una possibile opzione è quella di accumulare il centro globale 3D di probabilità per ogni regione, utilizzando le informazioni della camera di profondità. Tuttavia, i pixel fuori livello peggiorano il risultato della stima globale. Si è quindi scelto di utilizzare un approccio di tipo locale basato sull'algoritmo *Mean Shift*, descritto in [73]. Lo stimatore di densità per ogni parte del corpo è stato definito nel seguente modo:

$$f_c(\widehat{x}) \propto \sum_{i=1}^{N} w_{ic} exp\left(-\|\frac{\widehat{x} - \widehat{x_i}}{b_c}\|^2\right) \tag{2.3}$$

dove $\widehat{x}$ rappresenta una coordinata nello spazio 3D, N è il numero dei pixel, w_{ic} il peso di un pixel, $\widehat{x_i}$ la proiezione del pixel x_i nello spazio data una profondità $d_I(x_i)$, e b_c un'ampiezza di banda. Il peso di un pixel w_{ic} tiene conto sia della probabilità P rispetto al pixel che dell'area del pixel:

$$w_{ic} = P(c|I,x_i)d_I(x_i^2) \tag{2.4}$$

Questo accorgimento permette di ottenere un piccolo ma significativo miglioramento nella rappresentazione della posizione dei giunti.

Il metodo *Mean Shift* permette di trovare metodi operativi in modo efficiente. Tutti i pixel sopra una data soglia di probabilità λ_c vengono usati come punti di partenza per la parte c. Una stima di confidenza finale

viene ricavata come somma dei pesi dei pixel che hanno raggiunto il metodo. Questo tipo di approccio è sicuramente più affidabile.

2.4 Elettronica nello sport

Per quanto riguarda le arti marziali, al momento l'unico sport da combattimento (oltre alla scherma) che prevede l'utilizzo di strumenti elettronici che automatizzano il conteggio dei punti è il *Taekwon-Do WTF*.

In tale disciplina vengono utilizzati da circa 3 anni a questa parte speciali corpetti elettronici basati su tecnologia *Bluetooth*. Un esempio di corpetti è rappresentato in Figura 2.14.

Figura 2.14: Corpetti elettronici

Queste corazze vengono equipaggiate con un sensore in schiuma a canale d'aria *ACSF*[14]. Tale sensore è in grado di rilevare con che forza è stato portato il singolo colpo e può essere tarato sulla forza degli atleti in base alle categorie di peso.

[14] Air Channel Sensor Foam

Oltre al sensore su ogni corazza è posto un trasmettitore *bluetooth* che trasmette in tempo reale al computer di gara lo stato del sensore.

Un simile sistema ha ovviamente il vantaggio di non essere affetto dall'errore umano, ma ha anche degli svantaggi, o meglio porta gli atleti a cambiare il loro modo di combattere.

Oltre ai corpetti si stanno studiando anche guantini e caschetti con sensori elettronici, ma queste tecnologie non sono ancora in utilizzo.

Il *Taekwon-do WTF* rispetto al *Taekwon-Do ITF* presenta numerose differenze, una di queste è il fatto che nel secondo non si utilizzano corpetti in quanto talvolta risultano scomodi e rendono difficili i movimenti. Inoltre nel *Taekwon-Do ITF* sono permessi anche pugni al viso, quindi l'utilizzo del corpetto non permette sicuramente di tenere traccia di tutti i punti possibili.

Scopo di questa tesi è presentare una soluzione che possa far fronte all'errore umano e permetta in tempo reale, mediante l'utilizzo di sensori, di tenere il conto dei punti senza bisogno da parte degli atleti di indossare nessun dispositivo elettronico. Per questo scopo è stato implementato un classificatore basato su *template* che permette di riconoscere determinate azioni, come ad esempio il *pugno al viso*.

Capitolo 3

Caso di Studio

In questo capitolo verrà descritto il contesto per cui è stato studiato ed implementato il classificatore proposto. Il campo di applicazione è il combattimento sportivo dell'arte marziale *Taekwon-Do ITF*[1], di cui verrà presentata una breve descrizione. Si valuteranno poi i limiti ed i problemi attuali per quanto riguarda il conteggio dei punti durante il combattimento e presentata una proposta di soluzione per far fronte a ciò utilizzando il *Kinect* ed il nuovo classificatore.

3.1 Taekwon-Do ITF

Il *Taekwo-Do*, letteralmente "arte dei pugni e dei calci in volo", è un'arte marziale coreana basata principalmente sull'uso di tecniche di calcio, nonché l'arte marziale che conta il maggior numero di partecipanti in tutto il mondo.

Il *Taekwon-Do* è diviso in diverse federazioni: quella *ITF*, federazione privata ed indipendente, venne fondata il 22 Marzo 1966 in Corea Del Nord da parte del generale Choi Hong Hi. La federazione *WTF*[2] invece venne fondata in Corea del Sud il 28 Maggio 1973, diventando nel 2000 sport olimpico ufficiale.

[1] International Taekwon-Do Federation
[2] World Taekwon-Do Federation

Il *Taekwon-Do ITF* si compone di due parti: una tradizionale ed una sportiva. La parte del combattimento sportivo è oggetto di questa analisi.

3.1.1 Il combattimento sportivo

Il combattimento sportivo, la parte più dinamica e seguita di questa disciplina, si svolge su un quadrato di gara di dimensioni diverse a seconda dell'età degli atleti. Per i bambini è solitamente 6 per 6 metri, mentre per gli adulti 8 per 8 metri. Le regole principali sono le seguenti:

1. Non sono ammessi colpi sotto la cintura
2. Non sono ammessi colpi dietro la schiena, dietro la testa
3. Non sono ammesse ginocchiate o gomitate
4. Gli atleti devono indossare dei guantoni e dei calzari

Lo stile di combattimento che caratterizza il *Taekwon-Do ITF* si avvicina molto a quello della specialità *Light Contact* della disciplina *Kick Boxing*, infatti spesso durante le gare di combattimento possono prendere parte atleti di entrambe le discipline. I colpi sono dunque ammessi dall'altezza della cintura in su, utilizzando sia tecniche di mano che di piede , con il rispettivo punteggio:

- Pugno al corpo o al viso: **1 PUNTO**
- Calcio al corpo: **2 PUNTI**
- Calcio al viso: **3 PUNTI**

La direzione dell'incontro è affidata ad un arbitro centrale il cui compito è controllare che tutto si svolga secondo le regole, segnalare le ammonizioni, dividere gli atleti in determinate situazioni, etc.

Il conteggio dei punti è invece affidato a 4 arbitri posti ognuno ad un angolo del quadrato muniti di un controller collegato al pc di gara. Il punteggio quindi viene aggiornato in tempo reale e visibile su un display. Ogni arbitro assegna il proprio punteggio e a fine incontro vince l'atleta a cui la maggioranza degli arbitri attribuisce la vittoria.

3.2 Problemi e limiti

Come descritto precedentemente, il conteggio dei punti è affidato a 4 arbitri d'angolo. Questo aspetto presenta tuttavia diversi problemi.

Innanzitutto i punteggi sono distinti e separati, mentre dovrebbe essere unico in quanto si tratta di una valutazione oggettiva e non soggettiva in cui i punti devono essere tutti segnalati. Sostanzialmente in un sistema perfetto ogni arbitro dovrebbe vedere e segnare gli stessi punti che segnano anche gli altri arbitri in modo da ottenere per tutti quanti lo stesso punteggio finale. Questo non accade a causa diversi motivi, ecco i principali:

- Come in tutti gli sport purtroppo anche questa disciplina è affetta da imparzialità dovuta al comportamento umano, di conseguenza un arbitro tende a favorire un atleta della propria palestra/nazione.

- Un arbitro può anche trovarsi in una situazione in cui non riesce a vedere il colpo a causa del suo punto di vista che non gli permette di farlo, di conseguenza tenderà a non segnare tale punto anche se in realtà il punto effettivamente c'era. In Figura 3.1 sono rappresentati due differenti punti di vista della stessa azione, come si può notare nella immagine do destra non è possibile stabilire con certezza se il contatto è avvenuto o meno.


Figura 3.1: Differenti punti di vista

3.3 Proposta di soluzione

Per far fronte ai problemi descritti precedentemente bisogna cercare di creare un sistema che garantisca un metodo di conteggio dei punti univoco ed assolutamente oggettivo.

Si potrebbe ad esempio pensare ad un sistema in cui ci siano ancora 4 arbitri d'angolo ma il punteggio di ognuno contribuisca ad un unico punteggio totale. In questo modo si otterrebbe sicuramente un punteggio univoco ma sicuramente non si risolverebbe il problema legato all'imparzialità, anzi in tal modo si potrebbe anche accrescerlo. Purtroppo tale problema non può essere facilmente trascurato.

Scopo di questa tesi è presentare la base per una soluzione che sia in grado di far fronte sia al problema dell'imparzialità, sia costituire un sistema di punteggio univoco ed assolutamente oggettivo.

Scopo ultimo e finale del progetto è creare un sistema di *Gesture Recognition* basato su un classificatore che permetta di riconoscere azioni tra soggetti diversi, e riconoscere quindi azioni quali pugno al viso, pugno al corpo, calcio al viso, calcio al corpo ecc. Le variabili da considerare in questo contesto sono davvero moltissime e si richiede un lungo e pesante lavoro di creazione e popolamento del data-set di campioni.

La finalità è utilizzare inoltre 4 sensori *Kinect* posizionati agli angoli del quadrato di gara, come mostrato in Figura 3.2, che lavorino in parallelo e

per ogni punto riconosciuto, valutando opportunamente le *features* estratte e la combinazione di queste, assegnino il rispettivo punteggio.

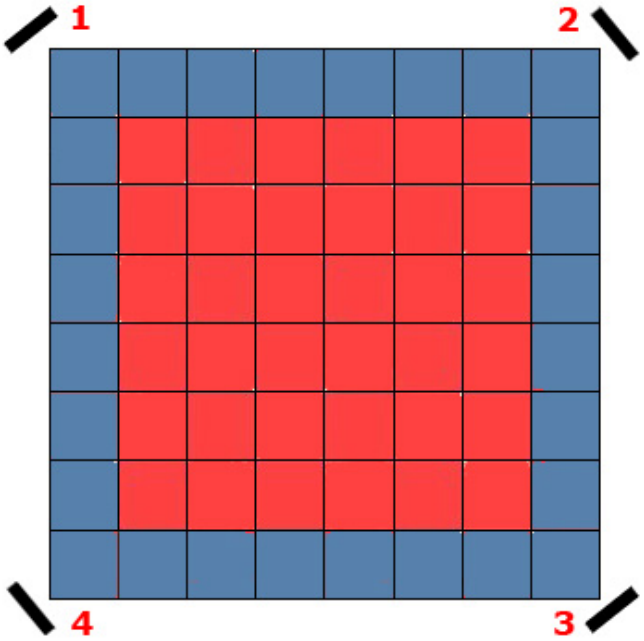

Figura 3.2: Disposizione sensori Kinect.

Questa tesi si propone di illustrare come il nuovo classificatore implementato sia in grado di gestire 2 soggetti diversi e riconoscere una delle possibili azioni che costituiranno il data-set finale, in particolare l'azione del *pugno al viso*. Quest'azione è sicuramente una delle più semplici da riconoscere tra tutte quelle possibili.

Capitolo 4

Strumentazione

In questo capitolo verranno descritte tutte le scelte hardware e software effettuate per l'implementazione del classificatore *KMC* e dell'applicativo. Verranno descritte quindi le caratteristiche principali riguardanti la dotazione hardware, cioè *Kinect*, e la dotazione software, comprensiva di ambiente di sviluppo, linguaggio di programmazione e librerie utilizzate.

4.1 Hardware

In questa sezione verrà presentato *Kinect* di casa *Microsoft* e verranno discussi i motivi che hanno spinto alla scelta di tale componente.

4.1.1 Kinect

Microsoft Kinect (inizialmente conosciuto con il nome *Project Natal*), è un accessorio per la console di gioco *Xbox 360* sensibile al movimento del corpo umano; a differenza del *Wiimote* della *Nintendo* e del *PlayStation Move* della Sony esso rende il giocatore stesso controller della console senza l'uso di strumenti, come invece accade per i concorrenti.

Sebbene in origine pensata per *Xbox 360*, Microsoft prevede di rendere nel prossimo futuro disponibile l'uso della periferica ai PC dotati del nuovo sistema operativo Windows 8. Kinect è dotato di telecamera RGB, doppio sensore di profondità a raggi infrarossi composto da un proiettore a infrarossi

e da una telecamera sensibile alla stessa banda. La telecamera RGB ha una risoluzione di 640 x 480 pixel, mentre quella a infrarossi usa una matrice di 320 x 240 pixel. Kinect dispone anche di un array di microfoni utilizzato dal sistema per la calibrazione dell'ambiente in cui ci si trova, mediante l'analisi della riflessione del suono sulle pareti e sull'arredamento. In tal modo il rumore di fondo viene eliminato ed è possibile riconoscere correttamente i comandi vocali. La barra del Kinect è motorizzata lungo l'asse verticale e segue i movimenti dei giocatori, orientandosi nella posizione migliore per il riconoscimento dei movimenti. Di fatto, la periferica permette all'utente di interagire con la console senza l'uso di alcun controller da impugnare, ma solo attraverso i movimenti del corpo, i comandi vocali o attraverso gli oggetti presenti nell'ambiente.

Microsoft dichiara che Kinect può seguire i movimenti di anche 4 giocatori simultaneamente, sia in piedi che seduti. Per ulteriori informazioni sul funzionamento di *Kinect* consultare [68].

Per quanto riguarda l'utilizzo su PC a Dicembre 2010 la società *Prime Sense*, una compagnia israeliana da tempo impegnata in ricerca e sviluppo di sistemi di controllo senza dispositivi fisici da impugnare e responsabile della tecnologia del sistema di telecamere di Kinect, ha rilasciato i driver open source per l'innovativa periferica Microsoft, compatibili con Windows e Linux (versione Ubuntu 10.10 in poi). Questi driver consentono di accedere alle funzioni audio, video ed ai sensori di profondità di Kinect e sono basati su un API completa, nota come *OpenNI (Open Natural Interactions)*[1].

OpenNI permette di catturare il movimento in tempo reale, il riconoscimento di gesti delle mani e dei comandi vocali. Il codice sorgente e la relativa documentazione sono disponibili sul sito del progetto OpenNI [82]. I driver non ufficiali sono stati sostituiti il 16/06/2011 dai driver ufficiali rilasciati da Microsoft, con licenza non commerciale per linguaggi *C#* e *C++*. Nel mese

[1] http://www.openni.org/

di Maggio 2012 sarà inoltre rilasciata la nuova versione *1.5 SDK per Kinect*, per le *Release Notes* relative alla nuova versione è possibile consultare [69].

In Figura 4.1 è possibile vedere i diversi componenti del *Kinect*, di seguito descritti:

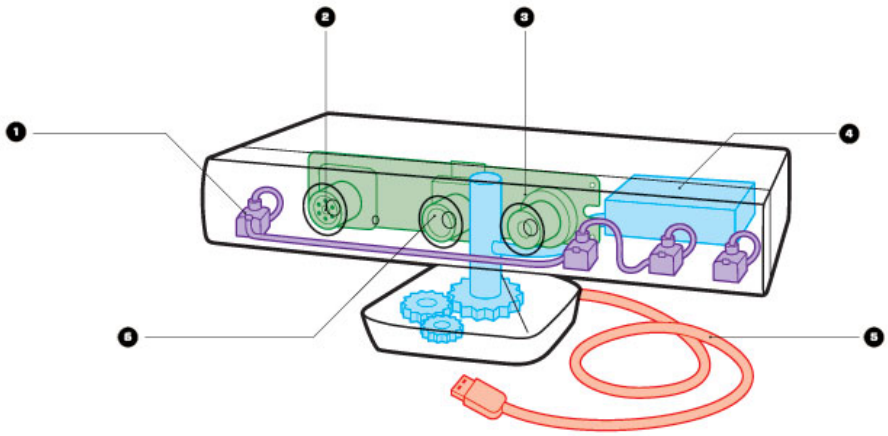

Figura 4.1: Componenti Kinect

1. **Microphone Array** Quattro microfoni con funzione di filtraggio del rumore del fondo.

2. **IR Emitter** Proietta un fascio di raggi infrarossi nell'ambiente, una volta che i raggi incontrano delle superfici vengono distorti e questa informazione viene letta dalla *Depth Camera*.

3. **Depth Camera** Analizza le informazioni dei raggi infrarossi per ricostruire una mappa tridimensionale dell'ambiente e degli oggetti rilevati.

4. **Tilt Motor** Permette di adeguare l'inquadratura a seconda del soggetto che si trova davanti, abbassando o alzando la visuale.

5. **USB Cable** Permette di collegare *Kinect* con *Xbox* oppure *PC*.

6. **Color Camera** Caratterizzata da una risoluzione di 640 x 480 pixel, permette di catturare le immagini. Insieme alle informazioni della *Depth Camera* permette di ricostruire il modello dell'ambiente e degli oggetti o delle persone inquadrate.

4.1.2 PC

Le caratteristiche del PC utilizzato per l'acquisizione, la registrazione ed il popolamento del data-set sono visualizzate in Figura 4.2

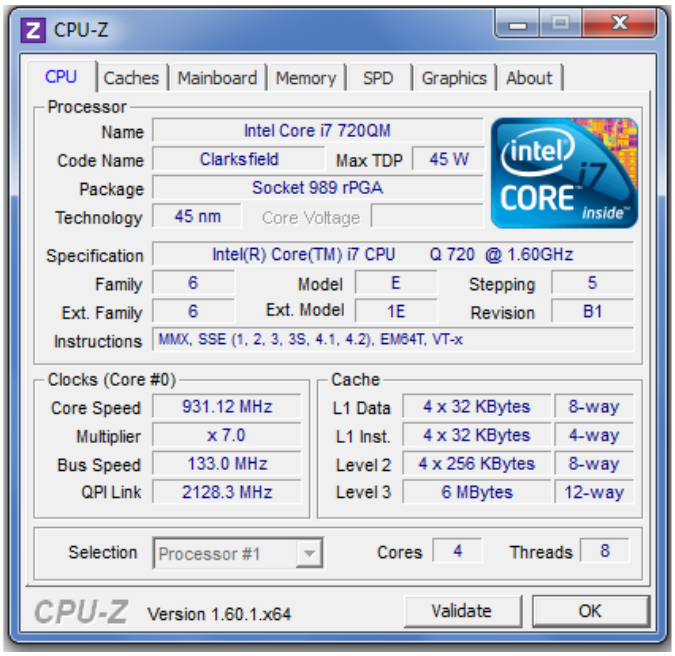

Figura 4.2: Caratteristiche CPU

4.2 Software

In questa sezione verranno descritti gli applicativi utilizzati per lo sviluppo dell'applicazione: linguaggio, ambiente, librerie etc.

4.2.1 Ambiente di sviluppo

L'ambiente di sviluppo utilizzato per l'implementazione è *Microsoft Visual Studio 2010*. Si tratta di un *IDE*[2] che supporta attualmente diversi tipi di linguaggio quali *C*, *C++*, *C#*, *F#*, *Visual Basic .NET* e *ASP .NET*, permettendo la realizzazione di applicazioni, siti e servizi web.

Inoltre è considerato un *RAD*[3], cioè un'applicazione atta ad aumentare la produttività del programmatore mediante servizi quali *IntelliSense*[4] ed un designer visuale.

Si tratta inoltre di un servizio multi-piattaforma: con esso è infatti possibile realizzare programmi per *server*, *workstation*, *pocket PC*, *smartphone* e *browser*.

Visual Studio integra il compilatore basato sul *.NET Framework*, che a differenza di quelli classici converte il codice sorgente in un codice intermedio *IL*[5]. *IL* rappresenta un nuovo linguaggio progettato per poter essere convertito in modo efficiente in codice macchina nativo su diversi tipi di dispositivi. Si tratta di un linguaggio di livello più basso rispetto a *Visual Basic .NET* o *C#*, ma ad un livello di astrazione più alto rispetto ai linguaggi macchina o *assembly*.

La scelta di tale software è dovuta in particolare al completo supporto per il linguaggio *C#*, descritto nella sezione successiva, ed alla ottima gestione dell'aspetto grafico degli applicativi grazie al designer visuale.

Rispetto alla versioni precedenti, l'ultimo *Visual Studio 2010* rilasciato in data 12 Aprile 2010 presenta le seguenti innovazioni:

- Sviluppo di applicazioni per il *.NET Framework 4.0*.
- Nuovo linguaggio di programmazione funzionale *F#*.

[2]Integrated Development Environment
[3]Rapid Application Development
[4]Forma di completamento automatico resa popolare da Visual Studio Integrated Development Environment, costituisce inoltre una documentazione per i nomi delle variabili, delle funzioni e dei metodi usando metadati e reflection.
[5]Intermediate Language

- Supporto programmazione parallela.
- Integrazione della libreria *jQuery*[6].

Per una panoramica più completa delle diverse versioni di *Visual Studio 2010* fare riferimento a [74].

4.2.2 Linguaggio C#

Il linguaggio scelto per l'implementazione è *C#*, un linguaggio di programmazione *object-oriented* sviluppato da *Microsoft* all'interno dell'iniziativa *.NET*, successivamente approvato come standard *ECMA*[7].

La sintassi prende spunto da quella del *Delphi*, e da *Java* e *Visual Basic* per quanto riguarda gli strumenti di programmazione visuale.

Il *C#* si può definire come il linguaggio che meglio degli altri descrive le linee guida sulle quali opera un programma *.NET*, *Microsoft* ha infatti creato il *C#* specificatamente per la programmazione sul Framework *.NET*, partendo da *C*, *C++* e *Java*. Si può pensare come un linguaggio *compilato* ed *interpretato* allo stesso tempo, vista infatti la sua integrazione con *.NET*, il codice sorgente viene solitamente compilato secondo i criteri di *JIT*[8] e trasformato in codice macchina solo su richiesta del caricamento o esecuzione del programma. Una seconda possibilità è la *Compilazione Ngen*, che permette di convertire il codice intermedio *CIL*[9] in codice macchina in una volta sola.

La scelta di tale linguaggio è dovuta sostanzialmente, oltre che alla potenza computazionale, alla integrazione con *Visual Studio* e con l'*SDK* ufficiale per *Kinect*.

[6]Libreria di funzioni javascript per le applicazioni web, con l'obiettivo di semplificare la programmazione lato client delle pagine HTML.

[7]European Computer Manufacters Association: Associazione fondata nel 1961 dedicata alla standardizzazione nel settore informatico e dei sistemi di comunicazione.

[8]Compilatore Just-In-Time: permette una compilazione detta traduzione dinamica, con la quale è possibile aumentare le performance dei sistemi di programmazione che utilizzano il bytecode traducendolo in codice macchina nativo in fase di run-time.

[9]Common Intermediate Language: Linguaggio di programmazione intermedio sviluppato da Microsoft

Per una panoramica sulle caratteristiche e specifiche di *C#* fare riferimento a [75].

4.2.3 Kinect SDK

Si tratta del pacchetto di applicazioni ufficiale rilasciato da *Microsoft* che mette a disposizione numerosi strumenti per poter utilizzare *Kinect* sul PC. La versione utilizzata per lo sviluppo dell'applicazione è la 1.0, mentre nel mese di Maggio 2012 è stata rilasciata la nuova versione 1.5, che oltre al tracking dello scheletro permette anche il riconoscimento facciale e porta diverse migliorie per quanto riguarda il riconoscimento vocale. Introduce inoltre la possibilità di effettuare il *tracking* di soggetti anche ad una distanza molto vicina (fino a 40 cm) e di collegare e gestire fino a 4 sensori *Kinect* sullo stesso pc. Questi aspetti sono molto importanti ed indispensabili per il fine ultimo dell'applicativo, quindi si renderà sicuramente necessaria la migrazione alla nuova versione. Si discuterà di questo nel capitolo riguardante gli sviluppi futuri.

Le principali caratteristiche della versione 1.0:

Raw Sensor Stream Accesso alle informazioni della camera di profondità, camera RGB e microfono.

Skeletal Tracking Possibilità di effettuare il rilevamento e *tracking* di una o due persone.

Funzionalità Audio Funzionalità audio quali filtrazione del rumore di fondo, eliminazione dell'effetto di eco, identificazione della sorgente sonora e integrazione con *Windows Speech Recognition API*.

Completa documentazione Fare riferimento a [76].

Per maggiori informazioni riguardo a *Kinect SDK*, fare riferimento a [76]. In aggiunta all'*SDK* è stata utilizzata anche la libreria *Coding4Fun*

Kinect Toolkit [77], che mette a disposizione utili funzioni per lavorare con le informazioni di profondità, di colore e con il processo di *tracking*.

4.2.4 Kinect Toolbox

Si tratta di una libreria su cui sono sono state apportate le opportune modifiche per adattare la soluzione in modo da poter perseguire il proprio fine. Tale libreria offre numerose funzionalità riguardanti principalmente il riconoscimento di azioni, posizioni, stabilità dello scheletro, registrazione e replay di sequenze video, flussi per gestire le informazioni della videocamere di profondità e quella RGB.

La feature più importante è la possibilità di eseguire il il processo di *Gesture Recognition* mediante l'utilizzo di *Template* [78]. Tale aspetto e le modifiche effettuate all'algoritmo verranno discusse nel capitolo successivo.

Per una panoramica completa e dettagliata delle funzionalità offerte da *Kinect Toolbox* fare riferimento a [79].

Capitolo 5

Sviluppo

Questo capitolo tratterà la fase di sviluppo dell'applicativo, affrontando aspetti quali l'implementazione del classificatore, la fase di creazione del data-set e la parametrizzazione mediante *XML*.

5.1 Classificatore

A differenza di altri classificatori *template-based*, *KMC* è in grado di gestire più soggetti (nel nostro caso 2) ed un numero di giunti variabile. Il tutto è reso configurabile grazie all'utilizzo di file di configurazione in *XML*, come verrà descritto successivamente.

Si tratta di un classificatore *Template-Based* definito dalle seguenti caratteristiche:

Template I *template* sono rappresentati dall'insieme di *features* che identificano un'azione, che può essere composta dal movimento di più giunti di soggetti diversi.

Features Una *feature* viene identificata dalla traiettoria spazio-temporale descritta dal movimento di un giunto del corpo di un soggetto sul quale viene effettuato il *tracking*. Viene definita da una serie di punti proiettati nel 2D mediante un processo di normalizzazione, descritto nella sezione successiva. Un esempio è mostrato in Figura 5.1

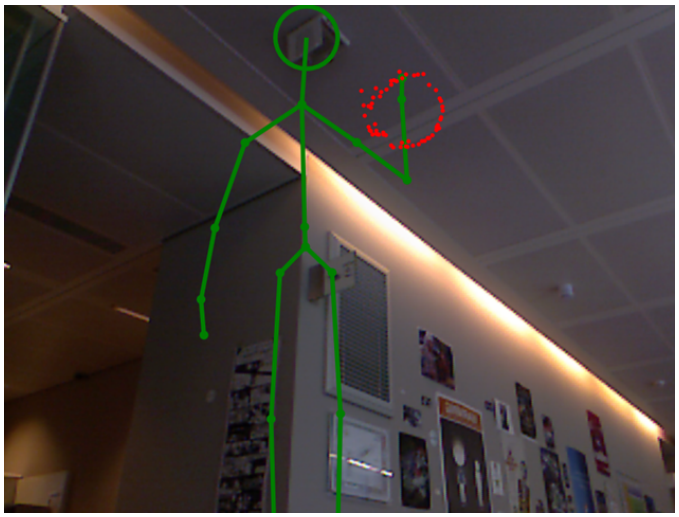

Figura 5.1: Feature

5.1.1 Normalizzazione dati 3D

I dati 3D, cioè le coordinate nello spazio *XYZ* vengono portati nello spazio 2D mediante un semplice processo di normalizzazione. Si è deciso di portare i dati 3D in 2D per semplificare il processo di riconoscimento vero e proprio, in modo da avere un minor carico computazionale. Da una parte questo certamente comporta u un grado di precisione minore nel rilevamento della posizione dei giunti, ma assumendo che il fine ultimo dell'applicativo coinvolga l'utilizzo di 4 sensori *Kinect* che lavorano in parallelo, si ritiene di poter far fronte a tale difetto.

5.1.2 Golden Section Search

Il confronto tra le osservazioni ed i *template* nel data-set potrebbe essere fatto utilizzando una semplice funzione che valuta la distanza media tra ogni punto, ma questo approccio risulta essere poco preciso. Per questo si è utilizzato un algoritmo chiamato *Golden Section Search*, descritto in [80].

Di seguito illustrata l'implementazione dell'algoritmo *Golden Section Search* in *C#*.

```
public static float Search(List<Vector2> current,
List<Vector2> target, float a, float b, float epsilon){

float x1 = ReductionFactor * a + (1 - ReductionFactor) * b;
List<Vector2> rotatedList = current.Rotate(x1);
float fx1 = rotatedList.DistanceTo(target);

float x2 = (1 - ReductionFactor) * a + ReductionFactor * b;
rotatedList = current.Rotate(x2);
float fx2 = rotatedList.DistanceTo(target);

do
  {
    if (fx1 < fx2){
     b = x2;
     x2 = x1;
     fx2 = fx1;
     x1 = ReductionFactor * a + (1 - ReductionFactor) * b;
     rotatedList = current.Rotate(x1);
     fx1 = rotatedList.DistanceTo(target);
    }else{
     a = x1;
     x1 = x2;
     fx1 = fx2;
     x2 = (1 - ReductionFactor) * a + ReductionFactor * b;
     rotatedList = current.Rotate(x2);
     fx2 = rotatedList.DistanceTo(target);
    }
  }
```

```
  while (Math.Abs(b - a) > epsilon);

  float min = Math.Min(fx1, fx2);

  return 1.0f - 2.0f * min / Diagonal;
}
```

Tale funzione ritorna un valore compreso tra 0 e 1.

5.2 Creazione data-set

Il processo di creazione ed ampliamento del *data-set* consiste nell'acquisire nuovi *template* che identificano un'azione assegnando per ognuna uno specifico tag. Nel caso in esempio è stata registrata l'azione associata al tag *pugno al viso*. Questo procedimento è stato gestito in modo automatico: inizialmente sono state registrate tutte le acquisizioni effettuate sui modelli, poi è stata effettuata un'analisi dei dati salvati controllando che non vi fossero errori come perdita di *tracking* etc, ed infine avviato un processo automatizzato che partendo dalle registrazioni memorizzava nel data-set i nuovi *template* per la specifica azione associando il rispettivo tag.

Una volta raccolti tutti i *template* necessari, a run-time il riconoscimento dei gesti avviene mediante la funzione *Match* implementata per ogni *RecordedPath*;

```
public bool Match(List<Vector2> positions, float threshold,
 float minimalScore, float minSize)
{
if (positions.Count < samplesCount)
return false;

if (!positions.IsLargeEnough(minSize))
```

```
return false;

List<Vector2> locals = GoldenSection.Pack(positions, samplesCount);

float score = GoldenSection.Search(locals, points,
-MathHelper.PiOver4, MathHelper.PiOver4, threshold);

Debug.WriteLine(score);

return score > minimalScore;
}
```

Il valore *minimalScore* è impostato di default a 0.8 ma può essere modificato a seconda delle proprie esigenze. Si possono infatti utilizzare valori di *score* differenti a seconda dell'azione considerata.

5.3 Parametrizzazione

In questa sezione verrà descritto il tipo di approccio utilizzato per poter permettere la parametrizzazione dell'applicativo mediante *XML*. In Figura 5.2 è rappresentato lo schema delle classi che vengono gestite tramite i parametri definiti in un file *XML*. In Figura 5.3 è invece raffigurata la struttura del file *XML* di configurazione.

Come si può notare dalla Figura 5.2 l'elemento principale definito nell'*XML* è *Configuration*, la cui classe associata si occupa di leggere i dati dal file e configurare le altre classi.

All'interno della classe *Configuration* è presente una collezione di classi *Gestures*, di conseguenza nel file *XML* di configurazione vengono definite da 1 a n *gestures* per le quali vengono impostati i seguenti parametri:

1. **color** Il colore che identifica a livello visuale il *tracking* dell'azione.

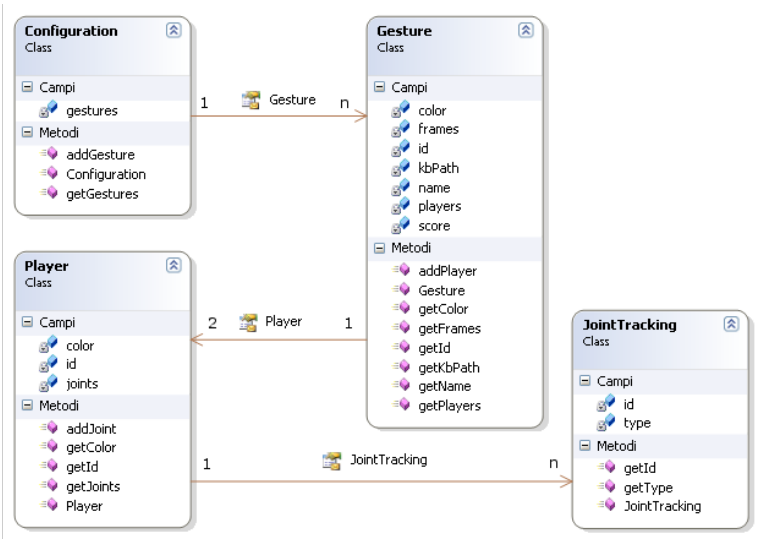

Figura 5.2: Class Diagram

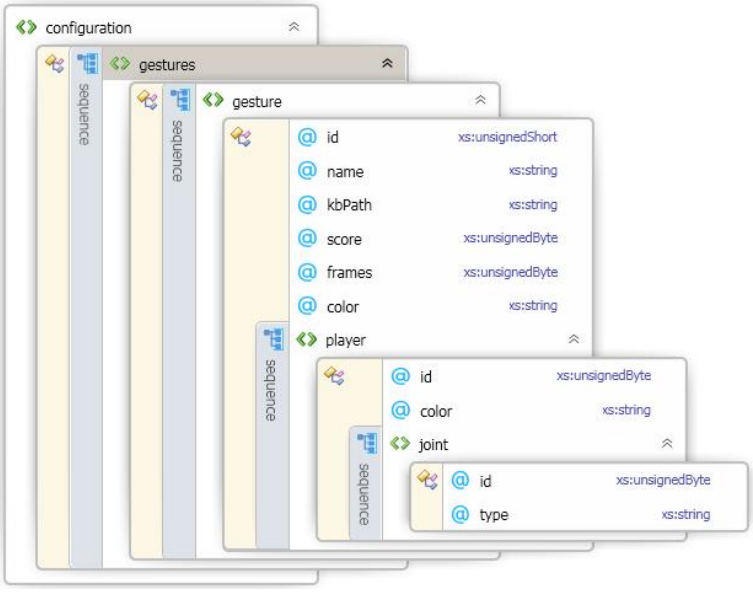

Figura 5.3: Struttura XML

2. **frames** Il numero di frame per un'azione.

3. **id** Id associato alla *gesture*.

4. **kbPath** Il percorso del file .save in cui sono memorizzati i *template* associati alla *gesture*.

5. **name** Nome e tag identificativo della *gesture*, lo stesso che viene visualizzato nel momento in cui viene riconosciuta.

6. **players** Collezione di classi di tipo *Player*, al momento possono essere gestiti al massimo 2 giocatori.

7. **score** Punteggio associato alla *gesture*. Ad esempio nel caso di un pugno al viso il punteggio assegnato è 1.

Come per la classe *Configuration*, così per la classe *Gesture* è definita una collezione di classi di tipo *Player*, nel file *XML* possono essere dunque definiti per ogni *gesture* da 1 a 2 giocatori per i quali è possibile definire i seguenti parametri:

1. **color** Colore associato al giocatore, importante per distinguere a livello visuale il giocatore che compie l'azione ed ottiene quindi il rispettivo punteggio.

2. **id** Id associato al giocatore.

3. **joints** Collezione di classi di tipo *JointTracking*, per ogni giocatore vengono definiti i giunti di cui è necessario eseguire il *tracking* per permettere il riconoscimento della rispettiva *gesture*.

Come descritto precedentemente, per ogni giocatore è quindi possibile definire da 1 a n *JointTracking* nel file di configurazione *XML*, descritti dai seguenti parametri:

1. **id** Id associato al giunto.

2. **type** Tipologia di giunto, ad esempio *testa*, *mano sinistra*, etc.

Nel file *XML* di configurazione è possibile definire tutti i parametri descritti ed il programma si occuperà di impostare l'applicativo nel modo opportuno. Se ad esempio per una data *gesture* avrò bisogno di considerare anche il piede sinistro, mi basterà semplicemente aggiungere tra gli elementi di tipo *joint* della determinata *gesture* un nuovo elemento con type corrispondente al *piede sinistro*.

Per una guida più approfondita su come configurare in modo opportuno i file *XML* dell'applicativo, fare riferimento all'appendice A.

Capitolo 6

Acquisizione Dati

In questo capitolo sarà descritta la fase di acquisizione dei dati necessaria per effettuare il popolamento del data-set del classificatore. Verranno trattati l'ambiente, i modelli utilizzati ed il procedimento adottato durante la fase di registrazione.

6.1 Ambiente

L'ambiente utilizzato per la registrazione delle azioni è rappresentato in Figura 6.1. Si tratta di un *tatami* delle dimensioni di 8 per 8 metri. L'illuminazione è fornita da 4 lampade poste sul soffitto che puntano sul centro della scena. I due soggetti si trovano ad una distanza di circa 4 metri dal sensore, cioè la distanza media a cui solitamente i soggetti si trovano durante un'azione di combattimento.

La scelta di tale ambientazione è dovuta al fatto di voler riprodurre nel modo più fedele possibile l'ambiente reale per cui è stato progettato l'applicativo. Inoltre presenta ulteriori vantaggi infatti, visto che il classificatore implementato è assolutamente indipendente dallo sfondo, non è assolutamente influente per quanto riguarda l'acquisizione del movimento dei soggetti. La caratteristica più importante è sicuramente l'illuminazione, infatti il *Kinect* lavora in condizioni ottimali quanto i soggetti sono ben visibili e distinguibili dallo sfondo, mentre il *tracking* non risulta invece ottimale in

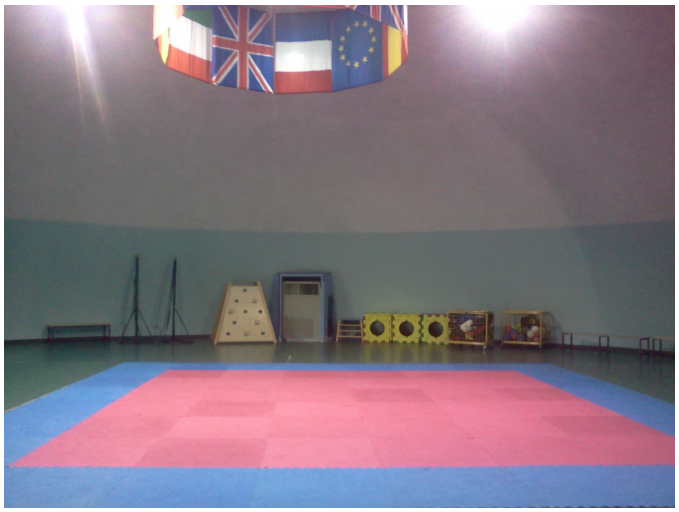

Figura 6.1: Ambiente di acquisizione

condizioni di luce scarna. Tuttavia, la presenza di 4 luci posizionate sopra i soggetti garantisce la corretta condizione di illuminazione e durante le fasi di registrazione non si sono riscontrati problemi di perdita di *tracking* legati a tale aspetto.

Un problema di tale ambiente potrebbe essere invece la distanza, infatti nel caso in cui i soggetti si trovino ad una distanza troppo ravvicinata o troppo lontana c'è il rischio di perdita di *tracking*. Come verrà descritto nel capitolo sugli sviluppi futuri, la nuova versione del *Kinect SDK* permette di gestire il *tracking* anche ad una distanza di 40 cm dal sensore. Un'altra soluzione, nel momento in cui viene perso il *tracking* da parte di un sensore, è lasciare momentaneamente l'elaborazione ai restanti 3 sensori previsti dal progetto iniziale, fino a quando i soggetti si ritroveranno ad una distanza adeguata per permettere il *tracking* simultaneo mediante tutti e 4 i sensori.

6.2 Modelli

In Figura 6.2 sono rappresentati i due modelli scelti per la fase di acquisizione dei *template* e in tabella 6.1 sono descritte le principali caratteristiche.


Figura 6.2: Modelli (a) e (b)

	Modello A	**Modello B**
Altezza	1,75 m	1,70 m
Peso	70 kg	58 kg
Sesso	M	F
Corporatura	Medio-Robusta	Media
Grado	I Dan	II Dan

Tabella 6.1: Caratteristiche modelli

Il classificatore implementato è assolutamente indipendente dal sesso, dalla corporatura e dal peso dei soggetti osservati, infatti l'unico aspetto rilevante ai fini dell'acquisizione è la velocità di esecuzione di un movimento e la sincronizzazione tra i 2 soggetti osservati, visto che il classificatore traccia nello spazio 2D la traiettoria descritta da uno specifico giunto in un arco di tempo prestabilito.

Il motivo principale per il quale sono stati scelti i modelli descritti precedentemente è dovuto sostanzialmente al fatto che entrambi hanno molta esperienza nella disciplina in questione, di conseguenza hanno permesso la registrazione di movimenti precisi ed eseguiti in modo corretto, su cui poter costruire un data-set di template per il classificatore, indipendente da sesso, corporatura e peso dei soggetti.

6.3 Procedimento

La fase di acquisizione dei dati per quanto riguarda l'azione etichettata come *pungo al viso* si è svolta seguendo i passi ivi descritti:

1. **Posizionamento:** i due soggetti si posizionano uno di fronte all'altro ad una distanza di circa 4 metri dal sensore in modo che venga inquadrato l'intero corpo, ed una distanza ravvicinata da combattimento assumendo una posa di guardia.

2. **Esecuzione azione:** uno dei due soggetti esegue la stessa azione (*pugno al viso*) un elevato numero di volte variando tra una volta e l'altra parametri quali la posizione di partenza, l'altezza della mano, il movimento compiuto ecc, ma sempre eseguendo la stessa specifica azione. Ogni sequenza, in particolare la traiettoria descritta dalla mano del soggetto che esegue il pugno e quella della testa dell'altro soggetto, viene registrata e salvata in un file il quale sarà poi processato per la creazione e memorizzazione di nuovi *template*. In questo modo

ogni azione è definita dalla traiettoria descritta dalla mano di uno dei due soggetti e dalla testa dell'altro. Inoltre per ogni sequenza vengono salvate le immagini acquisite tramite la camera *RGB* in modo da poter controllare eventualmente se vi siano stati problemi legati ad esempio a perdita di *tracking*. Un esempio di sequenza completa che identifica l'azione etichettata come *pugno al viso* è rappresentato in Figura 6.3.

3. **Salvataggio Template:** una volta salvate tutte le sequenze che identificano l'azione taggata come *pugno al viso*, queste sono state controllate ed elaborate in modo da rappresentare solo la determinata azione associata al tag del *pugno al viso* senza movimenti inutili aggiuntivi. Le sequenze sono state dunque tagliate in modo da riprodurre solo l'azione vera e propria. Infine tramite un processo automatico avviene il processo di ampliamento del *data-set*, durante il quale le sequenze registrate vengono utilizzate per la creazione di nuovi template e popolamento del data-set finale.

Figura 6.3: Sequenza di un'azione

Capitolo 7

Valutazione risultati

In questo capitolo verranno analizzati e discussi i risultati ottenuti, evidenziando in particolare pro e contro del nuovo classificatore implementato.

7.1 Popolamento Data-Set

Per la fase di creazione e popolamento del *data-set* iniziale sono state registrate 50 sequenze e memorizzati i rispettivi *Template* dell'azione etichettata come *pugno al viso* nel data-set, eseguite dal soggetto posto alla sinistra del sensore. I diversi *Template* si differenziano per posizione di partenza, orientamento del corpo, altezza della mano, velocità di esecuzione.

Esempi di alcuni dei *Template* salvati sono visualizzati in Figura 7.1.

Figura 7.1: Esempi di Template

7.2 Testing

Per la prima fase di *Testing* è stata registrata una sequenza video in cui al soggetto è stato richiesto di eseguire la stessa azione di pugno al viso 100 volte, variando tra un'esecuzione e l'altra parametri quali la posizione di partenza, l'altezza di partenza, l'orientamento del corpo. Una volta registrata l'esecuzione è stata riprodotta tramite l'applicativo e sono stati valutati i risultati ottenuti dal processo di *Recognition* tramite il confronto con i *Template* presenti nel data-set.

7.3 Analisi risultati

In questa sezione verranno analizzati i pro e contro del classificatore implementato valutando le possibili soluzioni per far fronte ai problemi riscontrati durante la prima fase di *Testing*.

Sul primo campione di 100 acquisizioni della stessa azione il classificatore è stato in grado di riconoscere correttamente l'azione del *pugno al viso* 76 volte, dimostrando quindi un valore di performance dell'**76 %** sul dato campione. Grazie alla possibilità di registrare e riprodurre le sequenze registrate all'interno dell'applicativo, è stato possibile analizzare mediante la funzione di *debugging* le principali cause della mancata rilevazione.

Sotto elencati i risultati di questa analisi:

- **Mancanza Template**

  12 azioni non sono state riconosciute perché non è stato trovato il *template* corrispondete nel data-set di partenza. Per far fronte a questo problema è necessario quindi provvedere ad ampliare il data-set di questa azione registrando nuove sequenze e memorizzando i rispettivi *template*.

- **Distanza**

6 azioni non sono state riconosciute per problemi relativi alla distanza dei soggetti, infatti da una certa distanza in poi il processo di *tracking* non avviene più in modo ottimale e le traiettorie descritte dai giunti considerati non vengono registrate in modo preciso. Per porre rimedio a questa problematica si prevede nel progetto finale l'uso di altri sensori che possano integrare l'informazione e permettere quindi il corretto riconoscimento dell'azione. Inoltre, come verrà descritto nel capitolo successivo, la versione dell'*SDK di Kinect* sarà migrata alla versione successiva, la quale prevede un miglioramento dell'algoritmo di *tracking* e la capacità di gestire i soggetti a distanze più lontane e più ravvicinate rispetto al sensore.

- **Velocità di esecuzione**

  6 azioni non stono state invece riconosciute per motivi legati alla velocità di esecuzione, infatti quando il pugno viene eseguito ad una velocità eccessivamente elevata le traiettorie dei giunti registrate non risultano complete e di conseguenza non viene trovata nessuna corrispondenza abbastanza fedele tra i *template* presenti nel data-set. Per far fronte a quest'ultima problematica è necessario integrare il data-set con ulteriori campioni per prevedere le diverse velocità di esecuzione. Inoltre è indispensabile definire nel file di configurazione il numero ottimale di frame richiesto per identificare una determinata azione, riducendolo o incrementandolo per poter gestire in modo ottimale le diverse velocità di esecuzione.

7.4 Considerazioni

Considerando che si tratta della prima fase di *testing* di un nuovo classificatore effettuato sul primo campione si possono considerare buoni i risultati ottenuti. Ovviamente per migliorare questi risultati è necessario ampliare il

data-set registrando nuovi *template* per la specifica azione, nonché l'implementazione ed estensione a tutte le altre azioni possibili e integrazione con le informazioni di altri sensori.

Valutando questi primi risultati ottenuti è possibile tracciare un'iniziale *overview* delle principali caratteristiche del nuovo classificatore, valutando punti di forza e debolezza.

In Tabella 7.1 sono riportati i principali pro e contro del classificatore.

Pro	**Contro**
Il classificatore implementato utilizza *features* indipendenti dallo sfondo, dal sesso, dalla corporatura dei soggetti acquisiti.	Il classificatore implementato è dipendente da velocità di esecuzione delle azioni, distanza dal sensore, distanza tra i soggetti.
A differenza dei classificatori attuali *KMC* permette il riconoscimento di azioni che coinvolgono 2 soggetti diversi.	Al momento attuale non permette la gestione di più di 2 soggetti, tale funzionalità verrà implementata nel momento in cui si renderà necessaria con la migrazione alla successiva versione dell'*SDK*.
Il classificatore è indipendente dalla posizione nel piano verticale dei soggetti che eseguono un'azione, infatti mediante il processo iniziale di normalizzazione dei dati nel piano 2D le traiettorie descritte dai giunti vengono centrate.	A causa del processo di normalizzazione dei dati nel piano 2D diminuisce sicuramente il grado di precisione. Soluzione a questo problema è l'integrazione di sensori aggiuntivi che lavorino in parallelo per garantire la massima precisione nel processo di *Gesture Recognition*.
La possibilità di eseguire il *Tracking* solamente dei giunti interessati permette di diminuire il carico di lavoro computazionale sulle risorse della CPU.	Il carico computazionale con l'aumentare dei campioni nel data-set risulta sempre più elevato in quanto si richiede un processo di *recognition realtime*. Rimedio a tale problematica è trovare il metodo più efficiente per parallelizzare le operazioni di *recognition* in modo da distribuire equamente il carico di lavoro tra le risorse a disposizione.
L'applicazione sviluppata che integra il classificatore è facilmente configurare grazie al file di configurazione *xml*, in cui è possibile definire la tipologia di azione, numero di frame, punteggio per la determinata azione, i giunti di cui eseguire il *tracking*.	
La possibilità di registrare e riprodurre sequenze di azioni permette mediante l'applicativo di automatizzare il processo di popolamento del data-set del classificatore, controllando eventuali errori e problematiche.	

Tabella 7.1: Pro e Contro classificatore

Capitolo 8

Sviluppi Futuri

In questo capitolo verranno discusse e valutate diverse soluzioni per permettere il miglioramento dell'applicativo ed inoltre consentirne la portabilità anche verso ambiti diversi rispetto a quello per cui nasce. Saranno inizialmente analizzate le soluzioni migliorative e successivamente i diversi campi in cui è possibile implementare il classificatore sviluppato.

8.1 Kinect SDK 1.5

L'applicativo è stato sviluppato utilizzando la versione 1.0 del *Kinect SDK* sviluppato da *Microsoft*. Nel mese di Maggio 20212 è stata rilasciata la nuova versione 1.5 che introduce diverse migliorie rispetto alla versione attuale. Tra le innovazioni principali:

Tracking di soggetti in posizione seduta Miglioramento del processo di *tracking*: possibilità di gestire anche solamente la parte superiore del corpo (10-giunti), nonché capacità di rilevare un soggetto in posizione seduta o accovacciata. Nel nostro applicativo specifico questa caratteristica non è fondamentale in quanto vengono considerati sempre soggetti in piedi, ma potrebbe essere utilizzata in ambiti diversi da quello sportivo.

Tracking ravvicinato Miglioramento della capacità di *tracking*: possibilità di rilevare soggetti in piedi o seduti fino ad una distanza di 40 cm dal sensore. In più, l'algoritmo di *tracking* è molto più veloce e meglio distribuito sulle risorse della CPU. Questa caratteristica risulterà sicuramente fondamentale in quanto sono possibili casi in cui i soggetti osservati possono trovarsi ad una distanza molto vicina al sensore.

Tracking facciale Possibilità di applicare una griglia virtuale sul viso di una persona permettendo cosi di riconoscere espressioni ed emozioni.

Developer Toolkit Si tratta di una raccolta di librerie, esempi, componenti per facilitare lo sviluppo delle applicazioni.

Kinect Studio Strumento che semplifica il testing delle applicazioni, offrendo funzionalità quali record, playback e debug di registrazioni.

Funzionalità di riconoscimento vocale Nuove lingue aggiunte al riconoscimento vocale.

Numero di sensori Possibilità di poter gestire fino a 4 sensori *Kinect* sulla stessa macchina. Questa funzionalità è sicuramente di fondamentale importanza in quanto il fine ultimo dell'applicativo prevede l'utilizzo di 4 sensori in parallelo, ognuno posto ad ogni angolo della scena di acquisizione.

Per maggiori informazioni riguardo alla versione 1.5 del *Kinect SDK* fare riferimento a [76].

8.2 Leap 3D

Leap 3D è un nuovo sistema di controllo tridimensionale presentato dall'azienda *Leap Motion* basato sui movimenti delle mani che promette un rilevamento delle gesture con una precisione di circa 0,01 millimetri, il che

equivale a 100 volte di più del *Kinect*. A differenza dell'approccio di *Microsoft*, *Motion Leap* ha creato un'area di lavoro tridimensionale di circa quattro metri cubi. Il tutto grazie ad un piccolo dispositivo USB con sensori e telecamere abbinato al software prodotto da *Leap Motion*, che è in grado di monitorare più oggetti contemporaneamente e di riconoscere le gesture.

Figura 8.1: Leap 3D

Il sistema così come è stato progettato sarà disponibile sia con Windows che con Mac OS X. Il sistema accetta anche selezioni tipiche dei dispositivi touchscreen, come per esempio il *pinch-to-zoom*.

Leap Motion è una startup, ma dispone dei finanziamenti che le consentiranno di terminare lo sviluppo e mettere in vendita il progetto a inizio 2013, al prezzo accessibile di circa 70 dollari.

Per quanto riguarda l'applicativo, si può pensare a come poter integrare e posizionare questo nuovo strumento in modo da ottenere un maggior grado di precisione nel rilevamento delle gesture, visto che il livello raggiunto è davvero alto. Questo potrebbe portare sicuramente un notevole miglioramento dal punto di vista della precisione.

Per maggiori informazioni riguardo a *Leap 3D* fare riferimento a [81].

8.3 Parallelismo

Al momento attuale il riconoscimento delle *gesture* è affidato unicamente alla CPU senza una vera gestione del parallelismo. Sarebbe importante pensare ad un metodo efficace per poter distribuire in modo ottimale l'utilizzo delle risorse in modo da ottenere un aumento delle prestazioni. Ad esempio, con un grande data-set di azioni diverse, sarebbe opportuno capire come suddividere queste informazioni tra i diversi core nel caso di CPU multi-core. Si può assegnare ad ogni core una quantità fissa di *template* su cui effettuare il processo di *recognition* oppure assegnare a core diversi il *tracking* di giunti differenti.

Inoltre, nel momento in cui verranno utilizzati 4 sensori in parallelo, bisogna sicuramente studiare il metodo migliore per parallelizzare le risorse per ottenere il massimo risultato in termini di tempo di computazione.

8.4 Campi di applicazione

Nonostante il campo di applicazione iniziale dell'applicativo sia quello sportivo, in particolare quello degli sport da combattimento, si possono sicuramente trovare numerosi altri ambiti in cui l'utilizzo del classificatore sviluppato possa risultare utile e portare diversi miglioramenti.

In generale gli ambiti in cui tale classificatore potrebbe risultare utile sono quelli che comportano interazione tra persone diverse. Sotto elencati alcuni dei campi di applicazione possibili:

Videogame Videogiochi come già quelli che esistono per *Xbox* in cui è richiesta l'interazione tra gli utenti, ad esempio giochi di ballo, sport, etc.

Sport Oltre alle arti marziali, anche in altri sport è possibile pensare di utilizzare il classificatore sviluppato, soprattutto in sport che richiedono sincronizzazione tra persone diverse, in modo da poterla valutare oggettivamente tramite l'applicativo.

Integrazione con sensori Le informazioni di tipo visivo possono essere integrate con altre tipologie di informazione in modo da ottenere una maggiore precisione nel processo di *Gesture Recognition*. Un esempio potrebbe essere l'utilizzo di segnali *EMG*[1], con i quali poter rilevare anche la forza e la potenza applicata durante i colpi.

[1] Elettromiografia

Capitolo 9

Conclusioni

Il tipo di classificatore implementato rispecchia le caratteristiche e gli obiettivi prefissati, infatti permette di effettuare il processo di *Gesture Recognition* tramite *Kinect* di azioni che coinvolgono soggetti diversi, utilizzando come *features* le traiettorie descritte nello spazio 2D dai giunti interessati in un arco temporale prestabilito.

Si è dimostrato come sia possibile riconoscere la più semplice azione del *pugno al viso* compiuta da un soggetto utilizzando un data-set di *template* registrati con i quali è stato effettuato il processo iniziale di popolamento del data-set del classificatore. Si tratta dello step iniziale per la realizzazione di un classificatore basato su *Kinect* che permetta di riconoscere qualsiasi tipo di azione che coinvolga soggetti diversi, utilizzando a tal scopo un enorme data-set di tutte le possibili tipologie di azioni.

Il tipo di classificatore proposto rappresenta una soluzione al problema riguardante l'imparzialità di un essere umano e costituisce un sistema di conteggio dei punti oggettivo ed unico, anche se caratterizzato da un grado di precisione non assoluto.

La scelta della dotazione hardware e software descritta nel quarto capitolo ha permesso la completa integrazione dei diversi componenti e l'implementazione dell'applicativo senza presentare nessun problema di incompatibilità.

Durante la fase di acquisizione dei dati si è prestata particolare attenzione a dettagli quali l'ambiente e la scelta dei modelli, in modo da riprodurre la situazione reale per cui è finalizzato il classificatore. Si sono riscontrati problemi legati soprattutto alla distanza dei soggetti dal sensore, in quanto l'attuale algoritmo di *tracking* di *Kinect* lavora in modo ottimale solo a determinate distanze mentre perde di precisione a distanze troppe lontane o vicine al sensore. Non vi è stato invece nessun tipo di problema legato a parametri quali illuminazione o distinzione dello sfondo.

Si è potuto constatare che il classificatore implementato risulta indipendente da aspetti quali sfondo, corporatura e sesso dei soggetti acquisiti, grazie all'utilizzo di *features* indipendenti.

Durante la prima fase di *testing* tuttavia si sono riscontrati diversi problemi dovuti principalmente a parametri quali velocità di esecuzione, distanza dal sensore ed altri aspetti descritti nel capitolo riguardante i risultati. Per ognuno dei problemi riscontrati è stata proposta una traccia di soluzione che verrà adottata per il perseguimento dello scopo ultimo e finale del progetto.

Lo step successivo è la registrazione di *template* per ogni possibile tipo di azione, eseguita in tutti i modi possibili da ogni prospettiva con diverse velocità di esecuzione. Tali *template* saranno poi utilizzati per l'ampliamento del data-set del classificatore.

Il progresso tecnologico e lo sviluppo di nuove tipologie di sensori, così come descritto nel capitolo riguardante gli sviluppi futuri, permetteranno inoltre in momenti successivi un aumento di performance del classificatore ed un maggior grado di precisione per quanto riguarda la fase di *Gesture Recognition*.

Bibliografia

[1] G.Johansson. **Visual perception of biological motion and a model for its analysis.** *Perception & Psychophysics.*, 1973.

[2] T.B.Moeslund, A.Hilton, V.Krüger. **A survey of advances in vision-based human motion capture and analysis.** *Computer Vision and Image Understanding.*, 2006.

[3] D.Marr, H.K.Nishihara. **Representation and recognition of the spatial organization of three-dimensional shapes.** *Philosophical Transactions of the Royal Society of London.*, 1978.

[4] D.Gavrila, L.Davis. **Towards 3-d model-based tracking and recognition of human movement.** *International Workshop on Face and Gesture Recognition.*, 1995.

[5] D.Ramanan, D.A.Forsyth. **Automatic annotation of everyday movements.** *EECS Department, University of California, Berkeley.*, 2003.

[6] L.W.Campbell, A.F.Bobick. **Recognition of human body motion using phase space constraints.** *International Conference on Computer Vision.*, 1995.

[7] Y.Guo, G.Xu, S.Tsuji. **Understanding human motion patterns.** *International Conference on Pattern Recognition.*, 1994.

[8] Y.Yacoob, M.Black. **Parameterized modeling and recognition of activities.** *International Conference on Computer Vision.*, 1998.

[9] T.Darrell, A.Pentland. **Space–time gestures.** *Conference on Computer Vision and Pattern Recognition.*, 1994.

[10] J.Yamato, J.Ohya, K.Ishii. **Recognizing human action in time-sequential images using hidden markov model.** *Conference on Computer Vision and Pattern Recognition.*, 1992.

[11] L.Wang, D.Suter. **Recognizing human activities from silhouettes: motion subspace and factorial discriminative graphical model.** *Conference on Computer Vision and Pattern Recognition.*, 2007.

[12] A.F.Bobick, J.W.Davis. **The recognition of human movement using temporal templates.** *Transactions on Pattern Analysis and Machine Intelligence.*, 2001.

[13] M.Blank, L.Gorelick, E.Shechtman, M.Irani, R.Basri. **Actions as space–time shapes.** *International Conference on Computer Vision.*, 2005.

[14] R.Polana, R.Nelson. **Recognition of motion from temporal texture.** *Conference on Computer Vision and Pattern Recognition.*, 1992.

[15] R.Polana, R.Nelson. **Low level recognition of human motion.** *NAM.*, 1994.

[16] R.Cutler, M.Turk. **View-based interpretation of real-time optical flow for gesture recognition.** *International Conference on Automatic Face and Gesture Recognition.*, 1998.

[17] P.Scovanner, S.Ali, M.Shah. **A 3-dimensional sift descriptor and its application to action recognition.** *ACM International Conference on Multimedia.*, 2007.

[18] A.Efros, A.Berg, G.Mori, J.Malik. **Recognizing action at a distance.** *International Conference on Computer Vision.*, 2003.

[19] L.Zelnik-Manor, M.Irani. **Event-based video analysis.** *Conference on Computer Vision and Pattern Recognition.*, 2001.

[20] C.Thurau, V.Hlavac. **Pose primitive based human action recognition in videos or still images.** *Conference on Computer Vision and Pattern Recognition.*, 2008.

[21] I.Laptev, P.Perez. **Retrieving actions in movies.** *International Conference on Computer Vision.*, 2007.

[22] D.Tran, A.Sorokin. **Human activity recognition with metric learning.** *European Conference on Computer Vision.*, 2008.

[23] T.Serre, L.Wolf, T.Poggio. **Object recognition with features inspired by visual cortex.** *Conference on Computer Vision and Pattern Recognition.*, 2005.

[24] H.Jhuang, T.Serre, L.Wolf, T.Poggio. **A biologically inspired system for action.** *International Conference on Computer Vision.*, 2007.

[25] I.Laptev, T.Lindeberg. **Space–time interest points,.** *International Conference on Computer Vision.*, 2003.

[26] P.Dollar, V.Rabaud, G.Cottrell, S.Belongie. **Behavior recognition via sparse spatio-temporal features.** *ICCCN '05: Proceedings of the 14th International Conference on Computer Communications and Networks.*, 2005.

[27] J.C.Niebles, L.Fei-Fei. **A hierarchical model of shape and appearance for human action classification.** *Conference on Computer Vision and Pattern Recognition.*, 2007.

[28] A.Gilbert, J.Illingworth, R.Bowden. **Scale invariant action recognition using compound features mined from dense spatio-temporal corners.** *European Conference on Computer Vision.*, 2008.

[29] I.Laptev, M.Marszalek, C.Schmid, B.Rozenfeld. **Learning realistic human actions from movies.** *Conference on Computer Vision and Pattern Recognition.*, 2008.

[30] Y.Ke, R.Sukthankar, M.Hebert. **Event detection in crowded videos.** *International Conference on Computer Vision.*, 2007.

[31] M.Fischler, R.Elschlager. **The representation and matching of pictorial structures.** *IEEE Transactions on Computers.*, 1973.

[32] L.R.Rabiner. **A tutorial on hidden markov models and selected applications in speec recognition.** *Proceedings of the IEEE.*, 1989.

[33] J.Yamato, J.Ohya, K.Ishii. **Recognizing human action in time-sequential images using hidden markov models.** *Computer Vision and Pattern Recognition.*, 1992.

[34] A.Wilson, A.Bobick. **Learning visual behavior for gesture analysis.** *International Symposium on Computer Vision.*, 1995.

[35] M.Brand. **Shadow puppetry.** *International Conference on Computer Vision.*, 1999.

[36] T.S.Wang, H.Y.Shum, Y.Q.Xu, N.N.Zheng. **Unsupervised analysis of human gestures.** *Pacific Rim Conference on Multimedia.*, London, UK, 2001.

[37] Y.Wang, H.Jiang, M.Drew, Z.N.Li, G.Mori. **Unsupervised discovery of action classes.** *Conference on Computer Vision and Pattern Recognition.*, 2006.

[38] C.Sminchisescu, A.Kanaujia, Z.Li, D.Metaxas. **Conditional models for contextual human motion recognition.** *International Conference on Computer Vision.*, 2005.

[39] C.Bregler. **Learning and recognizing human dynamics in video sequences.** *Conference on Computer Vision and Pattern Recognition.*, 1997.

[40] M.H.Yang, N.Ahuja. **Recognizing hand gesture using motion trajectories.** *Conference on Computer Vision and Pattern Recognition.*, 1999.

[41] K.M.Kitani, Y.Sato, A.Sugimoto. **An MDL approach to learning activity grammars.** *Proceedings of the Korea–Japan Joint Workshop on Pattern Recognition.*, 2006.

[42] A.F.Bobick, J.W.Davis. **The recognition of human movement using temporal templates.** *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 2001.

[43] D.Weinland, R.Ronfard, E.Boyer. **Free viewpoint action recognition using motion history volumes.** *Computer Vision and Image Understanding.*, 2006.

[44] M.D.Rodriguez, J.Ahmed, M.Shah. **Action mach a spatio-temporal maximum average correlation height filter for action recognition.** *Conference on Computer Vision and Pattern Recognition.*, 2008.

[45] F.Lv, R.Nevatia, M.Lee. **3D human action recognition using spatio-temporal motion templates.** *ICCV Workshop on Human–Computer Interaction.*, 2005.

[46] S.Carlsson, J.Sullivan. **Action recognition by shape matching to key frames.** *Workshop on Models versus Exemplars in Computer Vision.*, 2001.

[47] C.Schuldt, I.Laptev, B.Caputo. **Recognizing human actions: a local SVM approach.** *International Conference on Pattern Recognition.*, 2004.

[48] S.Nowozin, G.Bakir, K.Tsuda. **Discriminative subsequence mining for action classification.** *International Conference on Computer Vision.*, 2007.

[49] D.Marr, L.Vaina. **Representation and recognition of the movements of shapes.** *Philosophical Transactions of the Royal Society of London.*, 1982.

[50] A.Bobick, Y.Ivanov. **Action recognition using probabilistic parsing.** *Conference on Computer Vision and Pattern Recognition.*, 1998.

[51] J.M.Rubin, W.A.Richards. **Boundaries of visual motion.** *Technical Report, Massachusetts Institute of Technology, Cambridge.*, USA, 1985.

[52] G.Rogez, J.Guerrero, J.Martinez del Rincon, C.Orrite Urunuela. **- Viewpoint independent human motion analysis in man-made environments.** *British Machine Vision Conference.*, 2006.

[53] V.Parameswaran, R.Chellappa. **View invariants for human action recognition.** *Conference on Computer Vision and Pattern Recognition.*, 2003.

[54] I.Junejo, E.Dexter, I.Laptev, P.Perez. **Cross-view action recognition from temporal self-similarities.** *European Conference on Computer Vision.*, 2008.

[55] L.W.Campbell, D.A.Becker, A.Azarbayejani, A.F.Bobick, A.Pentland. **Invariant features for 3-d gesture recognition.** *International Conference on Automatic Face and Gesture Recognition.*, 1996.

[56] I.Cohen, H.Li. **Inference of human postures by classification of 3D human body shape.** *IEEE International Workshop on Analysis and Modeling of Faces and Gestures.*, 2003.

[57] A.Ogale, A.Karapurkar, G.Guerra-Filho, Y.Aloimonos. **View-invariant identification of pose sequences for action recognition.** *VACE.*, 2004.

[58] M.Ahmad, S.W.Lee. **HMM-based human action recognition using multiview image sequences.** *International Conference on Pattern Recognition.*, 2006.

[59] F.Lv, R.Nevatia. **Single view human action recognition using key pose matching and Viterbi path searching.** *Conference on Computer Vision and Pattern Recognition.*, 2007.

[60] R.Souvenir, J.Babbs. **Learning the viewpoint manifold for action recognition.** *Conference on Computer Vision and Pattern Recognition.*, 2008.

[61] T.Starnerm A.Pentland. **Visual Recognition of American Sign Language Using Hidden Markov Models.** *Massachusetts Institute of Technology.*

[62] K.Nickel, R.Stiefelhagen. **Visual Recognition of pointing gestures for human-robot interaction.** *Image and Vision Computing.*, 2007.

[63] L.Bretzner, T.Lindeberg. **Use your hand as 3-D Mouse.** *Conference on Computer Vision.*, 1998.

[64] M.Hung, C.Hsieh, Y.Zhu. **Scene Classification for Baseball Videos Using Spatial and Temporal Features.** *I-Shou University Dahsu.*

[65] R.Fergus, P.Perona, A.Zisserman. **Object class recognition by unsupervised scale-invariant learning.** *Proc.CVPR.*, 2003.

[66] J,Winn, J.Shotton. **The layout consistent random field for recognizing and segmenting partially occluded objects.** *Proc.CVPR.*, 2006.

[67] Ho, T.Kam. **Random Decision Forests.** *Poceedings of the 3rd International Conference on Document Analysis and Recognition.*, 1995.

[68] **http://www.xboxway.com/speciali/archive/2010/06/21/speciale_come_funziona_kinect_xbox_360.aspx** *Funzionamento Microsoft Kinect.*, 2010.

[69] **http://msdn.microsoft.com/en-us/library/jj131036** *Microsoft Kinect for Windows SDK and Toolkit - v1.5 Release Notes.*, 2012.

[70] T.Gonzales. **Clustering to minimize the maximum intercluster distance.** *Theor.Comp.Sci.*, 1985.

[71] **http://mocap.cs.cmu.edu/** *CMU Mocap Database.*

[72] S.Belongie, J.Malik, J.Puzicha. **Shape matching and object recognition using shape contexts.** *IEEE Trans. PAMI.*, 2002.

[73] D.Comaniciu, P.Meer. **Mean shift: A robust approach toward feature space analysis.** *IEEE Trans. PAMI.*, 2002.

[74] **http://www.microsoft.com/visualstudio/en-us/products/2010-editions** *Overview Microsoft Visual Studio.*, 2010.

[75] **http://msdn.microsoft.com/en-us/library/618ayhy6** *C# Reference.*

[76] **http://www.microsoft.com/en-us/kinectforwindows** *Kinect for Windows.*

[77] **http://c4fkinect.codeplex.com** *Coding4Fun Kinect Toolkit.*

[78] **http://c4fkinect.codeplex.com** *Gestures and Tools for Kinect.*, 2011.

[79] **http://kinecttoolbox.codeplex.com** *Kinect Toolbox.*, 2011.

[80] **The Golden Section Search Method** *MCS*, University of Illinois at Chicago, 2005.

[81] **http://www.leapmotion.com** *Leap Motion.*, 2012.

[82] **http://www.openni.org/** *Sito ufficiale librerie Open-NI.*

Appendice A

Guida all'applicativo

Questa appendice ha lo scopo di fornire una breve guida all'interfaccia grafica dell'applicativo illustrando i suoi diversi elementi e come occorre configurare il file di *configurazione xml* per poter implementare il riconoscimento di una nuova *Gesture*.

A.1 GUI

In Figura A.1 è rappresentata l'interfaccia grafica dell'applicativo.

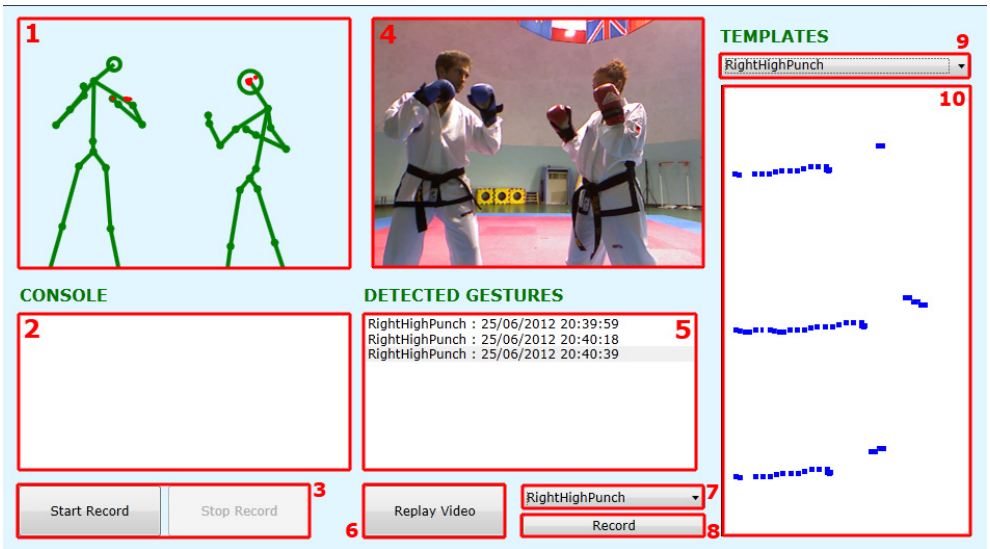

Figura A.1: Interfaccia grafica

Di seguito descritti i vari elementi:

1. **Gesture View** In questa finestra sono visualizzati gli *Skeleton* dei soggetti acquisiti e tracciato il *Tracking* dei giunti che competono per il riconoscimento della determinata *Gesture*.

2. **Information Console** Vengono qui riportati i vari messaggi di informazione, come ad esempio il salvataggio/riproduzione di un file di *Replay*.

3. **Start/Stop Record** I seguenti pulsanti servono per avviare e/o fermare il processo di registrazione di una sequenza video in un file con estensione *.replay*.

4. **RGB View** Questa finestra mostra l'informazione della videocamera RGB. Tale componente si rende utile nel momento di registrazione delle sequenze video in quanto oltre alle traiettorie dei diversi giunti vengono memorizzati anche i frame catturati dalla videocamera RGB. Tali immagini servono sostanzialmente a controllare che il processo di registrazione sia avvenuto in modo corretto.

5. **Gesture Console** Vengono riportate le *Gestures* riconosciute identificate dal rispettivo nome definito nel file di configurazione e riportato il *Timestamp* relativo al momento dell'identificazione.

6. **Replay** Questo pulsante permette di selezionare una sequenza *.replay* salvata precedentemente ed avviarne la riproduzione all'interno della *Gesture View*. Questa funzione è fondamentale perché permette di effettuare in un primo momento tutte le registrazione necessarie, e successivamente avviare il vero e proprio processo di popolamento del data-set di campioni utilizzando le sequenze salvate.

7. **Selezione Gesture** Questo menù a tendina permette di selezionare la *Gesture* corrente di cui si intende ampliare il data-set mediante i

relativi pulsanti. Le *Gestures* che vengono visualizzate in questo menù vengono recuperate dal rispettivo file di configurazione.

8. **Start/Stop Record Template** Questi pulsanti permettono di controllare il processo di ampliamento del data-set del classificatore, consentono infatti la registrazione e memorizzazione di nuovi *Template* nel data-set associati alla *Gesture* corrente selezionata nel menù sopra.

9. **Selezione Template** Questo menù a tendina consente di selezionare la *Gesture* di cui visualizzare i diversi template presenti nel data-set. Le diverse *Gestures* che è possibile selezionare dal menù sono recuperate dal file di configurazione.

10. **Template View** Vengono qui visualizzati graficamente i *Template* associati alla *Gesture* selezionata.

Nella sezione successiva verrà descritto come utilizzare il file di *configurazione xml* per implementare il riconoscimento di una nuova *Gesture*.

A.2 Configurazione Gesture

Per aggiungere una nuova *Gesture* al classificatore è necessario configurare opportunamente il file *configuration.xml*

Di seguito illustrato un frammento di codice del file di configurazione utilizzato per l'azione del *pugno al viso*, utilizzata come caso di studio per il classificatore:

```
<gestures>
   <gesture id="1" name="RightHighPunch"
    kbPath="RightHighPunch.save" score="1"
    frames="30" color="Red">
      <player id="1" color="Red">
         <joint id="1" type="Head" />
```

```
      </player>
      <player id="2" color="Blue">
         <joint id="2" type="HandRight" />
      </player>
   </gesture>
</gestures>
```

Per aggiungere una nuova *Gesture* occorre quindi configurare il file xml procedendo nel modo seguente:

1. Aggiungere un nuovo elemento di tipo *gesture* per i quali definire i seguenti attributi:
   (a) **id** Identificativo della *Gesture*.
   (b) **name** Nome e Tag della *Gesture*, lo stesso che comparirà come voce di menù nella selezione delle *Gestures* e dei *Template*.
   (c) **kbPath** File in cui sono memorizzati i *Template* associati alla *Gesture*.
   (d) **frames** Numero di frame che rappresentano la durata temporale dell'azione.
   (e) **score** Punteggio assegnato al momento dell'esecuzione dell'azione. Nel nostro esempio il punteggio di 1 viene assegnato al soggetto che esegue il pugno.
   (f) **color** Colore associato alla *Gesture*, utilizzato per visualizzare graficamente le traiettorie descritte dai giunti utilizzati per l'azione.
2. Aggiungere 1 o 2 nuovi elementi di tipo *player* per la *Gesture*, definendo per ognuno i seguenti parametri:
   (a) **id** Identificativo del *player*.

(b) **color** Colore che identifica il soggetto, necessario solamente per l'identificazione visuale.

3. Aggiungere per ogni elemento di tipo *player* un nuovo elemento di tipo *joint* definendo i due seguenti attributi:

   (a) **id** Identificativo del *joint*.

   (b) **type** Tipologia di giunto, in Figura A.2 sono rappresentati i diversi tipi di giunto che è possibile definire e in Tabella A.1 riportato il loro significato.

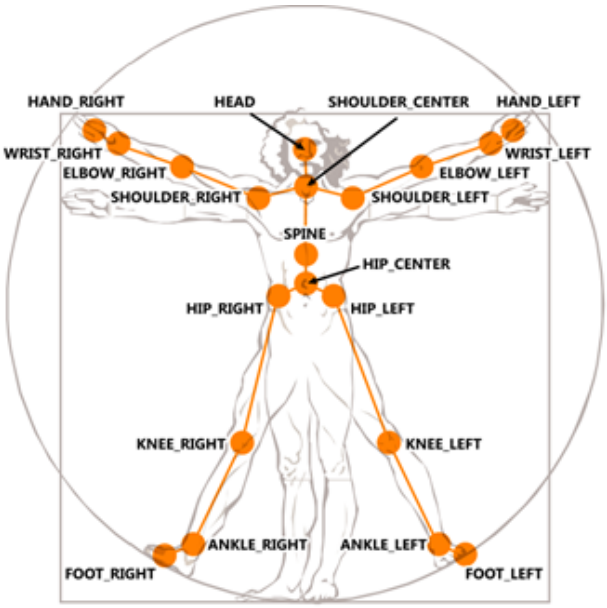

Figura A.2: Tipi di Joint

Type	**Descrizione**
AnkleLeft	Caviglia sinistra
AnkleRight	Caviglia destra
ElbowLeft	Gomito sinistro
ElbowRight	Gomito destro
FootLeft	Piede sinistro
HandLeft	Mano sinistra
HandRight	Mano destra
Head	Testa
HipCenter	Centro tra le anche
HipLeft	Anca sinistra
HipRight	Anca destra
KneeLeft	Ginocchio sinistro
KneeRight	Ginocchio destro
ShoulderCenter	Centro tra le spalle
ShoulderLeft	Spalla sinistra
ShoulderRight	Spalla destra
Spine	Schiena
WristLeft	Polso sinistro
WristRight	Polso destro

Tabella A.1: Tipi di giunto