ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

SEDE DI CESENA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA BIOMEDICA

ANALISI SPERIMENTALE SULL'UTILIZZO DEL MICROSOFT KINECT ONE COME SISTEMA DI BODY TRACKING PER LA REALTÀ VIRTUALE IN RIABILITAZIONE

Tesi in:

INGEGNERIA DELLA RIABILITAZIONE

Candidato:

Luca Soldati

Relatore:

Prof. Lorenzo Chiari

Correlatore:

Ing. Marco Pirini

Sessione III
Anno Accademico
2014-2015

“ Ma l'immaginazione ha

il volo dell'angelo e del lampo:

varca i mari dove noi rischiammo di naufragare,

le tenebre in cui si perdettero le nostre illusioni,

i pregiudizi in cui fu sommersa la nostra felicità”

- Alexandre Dumas (padre) -

Indice

Introduzione

Il Kinect V1 rappresentò nel 2010 una piacevole ed entusiasmante scoperta per il mondo video ludico. Con la possibilità di controllare avatar o oggetti a schermo senza necessità di marker o controller ha ben presto attirato l' attenzione di esponenti di altri settori tra cui il biomedicale. Con la nascita della seconda versione, totalmente rivoluzionato nel Firmware e nell'Hardware, si sono superati molti limiti del predecessore e, ad oggi, uno studio su questo nuovo sensore rappresenta un successivo passo avanti nella scoperta delle sue potenzialità. Definita la telecamera di profondità con migliori prestazioni tra quelle low-cost, il Kinect v2 presenta numerose altre caratteristiche tra cui un algoritmo di tracking interno capace di stimare la posizione di 25 punti anatomici nello spazio, 4 microfoni per determinare comandi vocali e 3 tipi di sensori di visione: a colori, ad infrarossi e di profondità. In questo progetto di tesi sarà innanzitutto presentato il sensore e sarà fatta una panoramica sull'uso della realtà virtuale in ambito riabilitativo. In seguito sarà analizzato l'algoritmo di Body tracking, valutandone il comportamento in diverse situazioni pratiche e poi stimandone la precisione in statica. Sarà presentato un filtraggio per limitare il rumore in tempo reale e valutarne i pro ed i contro in funzione delle caratteristiche impostabili. Le casistiche in cui potrebbe essere utilizzato il Kinect in clinica sono molto numerose per cui è stato necessario limitarsi ad eseguire un certo numero di prove. Nel corso dei capitoli saranno mostrate quelle effettuate, le opinioni ed i consigli su come si ritiene sia stato giusto procedere. Negli ultimi capitoli saranno presentate le metodologie con cui gli algoritmi integrati del Kinect permettono di ricavare una stima dell'orientamento delle parti anatomiche nello spazio ed alcune considerazioni circa le

implicazioni pratiche di tali metodologie, anche in base alle osservazioni sul campo ottenute durante i mesi di realizzazione di questo progetto. Lo scopo è naturalmente determinare se e come sia possibile utilizzare il Microsoft Kinect come unico sistema di motion tracking del paziente in applicazioni cliniche di riabilitazione, quali limiti ci sono nel suo utilizzo e quali categorie di scenari e prove potrebbe supportare.

Capitolo 1

La realtà virtuale in riabilitazione

Mai come ora, la realtà virtuale ha trovato un così forte interesse e sviluppo in diversi settori ed ambiti applicativi. Nata agli inizi degli anni '70, la realtà virtuale ha visto un interesse sempre più crescente fino al termine degli anni '80, quando le idee associate ad essa erano arrivate ad essere tanto fantasiose, quanto purtroppo era scarsa la tecnologia per realizzarle. Erano gli anni della fantascienza o sci-fi, gli anni in cui si formarono quelle idee futuristiche che non potendo essere realizzate per ovvi limiti tecnologici, portarono al progressivo disinteressamento verso questa vera e propria scienza. Dopo trenta anni tuttavia lo sviluppo scientifico è maturato abbastanza per rispolverare quelle idee e ritrovare lo stesso spirito che ha contraddistinto quegli anni, nel produrre progetti innovativi. La sostanziale differenza è che ora, forti di questo travolgente progresso tecnologico, la realtà virtuale è potuta tornare alla ribalta divenendo una promettente forma di investimento per settori come il militare, l'intrattenimento ed il medicale. Focalizzando l'attenzione in quest'ultimo settore e specialmente in riabilitazione clinica, la direzione presa è quella di cercare di migliorare sempre di più l'esperienza del paziente, giocando su diversi fattori psicologici, che saranno discussi a breve in questo capitolo. Con questo, non si intende sostituire la figura del clinico o del fisioterapista che opera la riabilitazione, ma anzi se ne vuole migliorare le condizioni lavorative, ottimizzandone i tempi di lavoro ed incrementando l'efficacia del trattamento **[1]**.

La realtà virtuale si basa su tecnologie volte a creare un ambiente interattivo con il quale l'utente può interagire per simulare attività del

mondo reale, eseguire esercizi creati ad hoc o giochi di fantasia. Queste tecnologie, fermo restando che sono sia di input che di output, vanno a formare la così detta interfaccia uomo-macchina. La funzione primaria di questa è quindi quella di rendere l'esperienza dell'utente quanto più coinvolgente possibile; per farlo è possibile lavorare a diversi gradi di complessità della strumentazione in uso e dei software impiegati. A questo proposito, l'elemento principale deve essere un computer, un elaboratore con capacità grafiche tali da poter processare la rappresentazione in tempo reale dell'ambiente virtuale ed avere una potenza tale da gestire le informazioni ricevute da tutte le periferiche, anche contemporaneamente **[2]**. Per percepire i cambiamenti dell'ambiente virtuale ed essere quindi completamente coinvolti in esso esistono diversi dispositivi di output che vanno ad interagire con i nostri cinque sensi. Si può cominciare da dispositivi di visualizzazione come monitor, occhiali e caschetti con display, continuando poi con dispositivi di ritorno uditivo come casse e altoparlanti dislocati nell'ambiente reale, dispositivi di ritorno aptico per il tatto, fino ai più raramente utilizzati dispositivi per il ritorno olfattivo e gustativo **[2]**.
Nei successivi paragrafi di questo capitolo saranno approfonditi sia i sistemi di output sia, in misura maggiore, quelli di input essendo, proprio uno di questi, il tema centrale di questa tesi. È importante tuttavia dire che ad oggi, nel 2016, tutte queste periferiche, ed in generale tutte la tecnologie che ruotano attorno alla realtà virtuale, sono ben lungi dall'essere arrivati a maturazione completa. Anzi, si può proprio dire che siano ancora nella prima fase di sviluppo tanto più che sono evidenti i progressi in corso con sempre più novità all'ordine del giorno. Per sistemi di ritorno aptico o di forza ad esempio ancora non ci sono progetti adeguati, per i visori invece sempre più colossi dell'elettronica e dell'intrattenimento come Sony, Htc, Microsoft, stanno sviluppando occhiali HMD (Head-Mounted Display) che possono decodificare il movimento oculare o del capo e contemporaneamente inviare il segnale visivo allo/agli schermo/i. Prima di iniziare quindi una

panoramica sulle tecnologie in uso in realtà virtuale, è importante parlare di quei fattori in precedenza citati che rendono promettente e vantaggiosa la realtà virtuale in clinica rispetto ad una tradizionale riabilitazione con solo trattamento da parte di fisioterapista o altro medico specializzato. La principale caratteristica dei sistemi VR è che possono adattare l'esperienza del paziente a quelle che sono le sue reali capacità fisiche **[3]**. Per la maggior parte delle persone è difficile avere una propriocezione del proprio corpo e dei propri movimenti tale da sapere sempre esattamente come ci stiamo muovendo durante un esercizio o che posizione stiamo tenendo se dobbiamo rimanere immobili in una posizione statica. Senza parlare poi di valutazioni oggettive sulla disposizione dei carichi o sul valore di angoli articolari. Solo l'uso di determinati strumenti, di determinati sensori permette di avere un quadro completo e continuo di questi tipi di situazioni e la realtà virtuale in questo gioca un ruolo chiave. Basti pensare alla possibilità per un paziente di poter vedere un avatar di se stessi a schermo che si muove esattamente come lui. Senza fornire numeri o dati probabilmente inspiegabili per una persona non del settore, e senza dover continuamente aggiustare la postura del paziente, gli si può dare un supporto visivo per potersi autoregolare. Questo permette di poter poi giudicare in maniera più oggettiva l'efficacia del trattamento in base alle potenzialità dell'utente stesso, aumentando di conseguenza le chance di un migliore biofeedback **[4]**. La possibilità di monitorare e misurare le prestazioni dell'utente in real-time rende molto efficace il biofeedback e di conseguenza anche la valutazione delle prove e dei miglioramenti (o dei peggioramenti) è più quantificabile utilizzando ad esempio dei punteggi, come riscontrabile in numerosi articoli in letteratura. Paraskevoupolos et al. **[3]** ad esempio ha condotto uno studio sull'efficacia di un trattamento in realtà virtuale per pazienti malati di Parkinson ricevendo dagli stessi giudizi molto positivi sull'esperienza. Molti hanno apprezzato l'alto coinvolgimento ed hanno espresso tra le proprie considerazioni, il

desiderio di poter "sfidare" altri pazienti o anche solo il computer al gioco cui sono stati sottoposti ed il desiderio che questi videogiochi (è più corretto chiamarli exerGames o serious games) simulino attività della vita quotidiana come il giardinaggio o la guida. Logicamente esistendo diverse patologie più o meno gravi che necessitano di riabilitazione, anche gli esercizi proposti tramite la realtà virtuale sono i più svariati sempre per il discorso dell'adattamento del sistema alle condizioni del paziente. È provato che possono trarre beneficio da riabilitazione tramite realtà virtuale ad esempio pazienti che hanno avuto ictus **[4]**, pazienti che devono recuperare in generale l'abilità motoria a degli arti **[5]**, pazienti che devono eseguire una neuro-riabilitazione ai primi stadi del recupero **[6]**, ma anche anziani, bambini e chiunque in genere abbia bisogno di lavorare sulla postura o sull'equilibrio **[3]**. Si possono avere videogiochi già esistenti sul mercato (off-the-shelf) adattati tramite l'uso di middleware agli esercizi di riabilitazione **[3]** oppure exerGames costruiti da zero con motori grafici o altre piattaforme.

Tornando ai vantaggi dei sistemi VR troviamo infine quello dato dalla possibilità di portare la riabilitazione direttamente a casa del paziente con sistemi integrati completi di tutte le periferiche necessarie a far svolgere gli esercizi in maniera autonoma. I vantaggi di questa operazione sono sia sui costi a lungo termine, sia sull'efficacia del trattamento. È dimostrato che avendo l'attrezzatura sempre a disposizione, il paziente è spinto ad esercitarsi per un numero maggiore di volte e con più impegno per velocizzare il recupero motorio, evitando i tempi di attesa e la necessità di dover lavorare su appuntamento, cosa che comporta inevitabili costi economici e di tempo.

Dispositivi di ritorno

Se l'ambiente virtuale è costruito per simulare un ambiente reale, le periferiche di ritorno in generale sono le responsabili restituzione ai nostri cinque sensi degli stimoli in esso generati. Per la maggior parte, le cliniche lavorano con ritorno visivo e uditivo ed in misura minore con ritorno aptico e di forza. Sistemi più sofisticati lavorano anche con olfatto e gusto, ma sono casi molto rari. È possibile quindi distinguere i sistemi in immersivi e non immersivi a seconda che diano l'impressione di osservare un vero ambiente 3D (sensazione definita "presenza") o che diano l'impressione di osservare l'ambiente semplicemente da una finestra **[1]**. Tipici del primo caso sono i monitor HMD come il celebre Oculus Rift, del secondo monitor e proiettori. Per il ritorno audio il discorso è più semplice perché dipende da come è distribuito l'impianto nella stanza o se è utilizzato ad esempio solo l'audio del computer con delle cuffie. Esistono poi sistemi molto ingombranti, ma che operano a 360 gradi con anche ritorno di tipo propriocettivo come il CAVE (Cave Automatic Virtual Environment) system o il CAREN (Computer Assisted Rehabilitation Environment). Come già detto i dispositivi di feedback tattile, di pressione e di forza sono ancora lontani dall'essere utilizzabili con efficacia sia per questioni di ingombro, che li rendono poco pratici da usare, sia per la scarsa resa dei dispositivi fin ora trovati. Sono utilizzati dispositivi pneumatici, dispositivi elettronici o piezoelettrici disposti su guanti o su veri e propri esoscheletri. Essendo molto facile trovare tante tipologie di strumenti per il ritorno sensoriale, sarebbe quasi impossibile elencarli tutti e visto che il tema di questa tesi non riguarda questa tipologia di dispositivi, per non divagare troppo è opportuno spostare l'attenzione sulla descrizione delle attuali tecnologie di motion tracking esplorando le diverse tipologie di sensori esistenti fino ad arrivare al vero fulcro di questa tesi: il Microsoft Kinect.

Dispositivi di Motion Tracking

Entrando nel vivo della questione è possibile analizzare le tecnologie più in uso per realizzare il body tracking anche detto body motion tracking o semplicemente motion tracking ovvero il riconoscimento del corpo e quindi dei movimenti dell'utente. Possiamo distinguere quattro tipologie di dispositivi di tracking basati su altrettante diverse tecnologie: magnetico, con estensimetri, con sensori inerziali ed ottico.

- **Tracking Magnetico**

  Si basa sull'induzione di corrente su un sensore ad opera di un campo magnetico generato da un'antenna emettitrice. Il campo magnetico è prodotto da tre dipoli magnetici ortogonali (tre bobine) ed è percepito a diverse intensità a seconda della posizione e dell'orientamento dal sensore che avrà anch'esso tre bobine di sensing **[7]**. Sebbene questi sistemi abbiano un costo contenuto e siano caratterizzati da una buona accuratezza, hanno il grosso inconveniente di essere suscettibili a distorsioni di campo dovuto alla presenza di oggetti metallici e di altri campi elettromagnetici (anche quelli prodotti dalla strumentazione stessa) **[4]**.

- **Tracking con estensimetri**

  Sono utilizzati per lo più per un preciso tracking dell'arto superiore, spesso anche solo della mano, disponendo gli estensimetri in appositi esoscheletri o guanti come il CyberGlove. Questo porta ad

una soluzione che sì, ha un'accuratezza eccellente, ma che visti gli ingombri risulta una soluzione abbastanza obsoleta. Sistemi moderni come il Leap motion permettono un tracking della mano paragonabilmente buono **[8]** senza aver bisogno di indossare appunto alcunché.

- **Tracking con sensori inerziali**

Questa tecnologia si basa sull'uso di sensori inerziali come accelerometri e giroscopi per realizzare il motion tracking tramite modelli biomeccanici e tecniche di sensor-fusion. Naturalmente maggiore è il numero di sensori utilizzati, maggiore è l'informazione ottenibile, specie con l'utilizzo di strumenti potenti come il filtro di kalman. Tuttavia dal solo utilizzo di questi non è generalmente possibile ottenere in termini affidabili informazioni spaziali di posizione, ma solo ottenere informazione su angoli e spostamenti.

- **Tracking ottico**

Questi sistemi posso dividersi ulteriormente in due categorie: con marker e markerless. Sicuramente il sistema più accurato è quello basato sull'uso di marker, posizionati opportunamente in diversi punti del soggetto di cui visualizzare i movimenti, ed un numero variabile, solitamente da 4 a 32, di telecamere ad infrarossi per rilevarli **[9]**. Sono sistemi molto costosi e che richiedono molto spazio, ma che grazie alla elevata accuratezza ed al numero elevato di frame acquisibili (da 30 a 2000) al secondo vengono spesso utilizzati in clinica per misure a bassissimo errore o anche come

gold standard per valutare altre tecnologie. I sistemi con marker si dividono poi in sistemi attivi e passivi a seconda che il marker abbia un led che emette luce o che rifletta solo l'infrarosso delle telecamere. In entrambi i casi comunque rimane il problema di dover far indossare i marker o tute con gli stessi già integrati al paziente e la necessità di lavorare esclusivamente in un laboratorio o comunque una struttura adeguatamente attrezzata allo scopo. A questo proposito stanno emergendo sempre più ricerche e progetti su sistemi markerless che individuano la silhouette del corpo umano e ne seguano i movimenti automaticamente tramite algoritmi di tracking. Si può partire da modelli più semplici che, con l'uso di telecamere RGB, permettono il tracking bidimensionale (senza informazioni di profondità) a sistemi più complessi che misurano i movimenti del soggetto in tutte e tre le dimensioni spaziali **[10]**. Per questi sono necessarie telecamere di profondità che lavorino con gli infrarossi. Anche in questo caso è possibile lavorare con una o più telecamere, sincronizzate **[10]** o meno **[11]**, aumentando tuttavia i costi e soprattutto la complessità computazionale dell'algoritmo di tracking. Le moderne telecamere di profondità a tempo di volo (TOF) risultano essere le più accurate, ma anche le più costose mentre, per quanto riguarda gli algoritmi di tracking, è possibile trovarne molti esempi in letteratura **[9]**. Attorno al 2010 sono apparsi in campo videoludico sistemi economici, ma con racchiuso un potenziale altissimo per l'uso in clinica: il Nintendo Wii Motion e la Nintendo Balance Board, il Sony Move ed il Microsoft Kinect. Di questi il Microsoft Kinect rappresenta proprio un reale sistema economico (il più economico sul mercato **[9]**) di motion tracking di tipo ottico e markerless con telecamera RGB, di profondità e con già implementato nel suo firmware l'algoritmo di tracking. Dal 2010 ad oggi diversi studi sono già stati fatti sul Kinect, tuttavia nel 2014 è stato rilasciato il Kinect One ovvero la seconda versione dello stesso, ma essendo cambiato sia l'hardware

che le librerie ad esso associate, ad oggi è stato necessario ricominciare da capo ogni studio su questo nuovo sensore. Così ogni passo avanti rappresenta un'interessante novità per analizzare a tutto tondo le capacità e le potenzialità per la sua utilizzazione in clinica **[12]**. Nel prossimo capitolo verrà presentato quindi il Kinect, sia nella prima versione che nella seconda, illustrando quelle che sono le principali caratteristiche che è stato possibile reperire in letteratura.

Capitolo 2

Il Kinect

Il Microsoft Kinect, inizialmente presentato come progetto Natal, è una periferica nata per console Xbox360 nel 2010, che permette al giocatore di interagire con la console senza l'uso di controller o altri accessori di intermediazione. Anche se nato per il mondo videoludico, ben presto l'interesse da parte di esponenti di molti altri settori tra cui il medicale ha spinto dapprima l'azienda americana a rilasciare nel 2011 i driver per poter utilizzare il Kinect con il proprio computer ed in seguito nel 2012 a produrre direttamente una versione Kinect For Windows. In questa versione è stato incorporato l'adattatore per collegare il sensore al PC tramite USB 2.0 ed il relativo SDK (Software Development Kit) con codici d'esempio e librerie complete per l'utilizzo. L'API (Application User Interface) e quindi l'SDK, è stato reso disponibile allo scaricamento dal sito ufficiale della Microsoft ed è stato periodicamente aggiornato fino alla versione 1.8. Nel 2014 infine è stata rilasciata la versione stand alone del Kinect One precedentemente solo inclusa nella nuova console Xbox One (uscita nel novembre 2013). Questo nuovo dispositivo, nato come progetto Durango e chiamato anche Kinect 2.0 o Kinect V2, è stato profondamente modificato nell'hardware e significativamente aggiornato e cambiato nel firmware. Microsoft, avendone capito il potenziale e soprattutto l'interesse fuori dal mondo videoludico ha deciso di rilasciare subito anche la versione per PC, tuttavia ad oggi ha cambiato la sua politica decidendo di vendere separatamente il Kinect per Xbox One e l'adattatore per collegare questo al PC tramite USB 3.0. Entrambi i Kinect hanno un costo di mercato di circa 200 €. In tab.

1 è possibile vedere un confronto tra i requisiti di sistema delle due versioni.

Kinect	**Kinect One**
Windows 7 o superiore	Windows 8 o superiore
Processore 32 bit o 64 bit	Processore 64 bit
2 GB RAM	4 GB RAM
Dual-Core 2.66 GHz o superiore	Dual-Core 3.1 GHz o superiore
USB 2.0	USB 3.0
Scheda grafica che supporti DirectX 9	Scheda grafica che supporti DirectX 11

Tab. 1 – Confronto tra i requisiti di sistema del Kinect e il Kinect One.

Hardware

La prima versione del Kinect è composta da una telecamera RGB con risoluzione di 640x480 pixel a 30 Hz incrementabile a 1280x1080 a discapito di un abbassamento del frame-rate. Possiede inoltre una telecamera di profondità composta da un proiettore di infrarossi ed una telecamera monocroma CMOS con risoluzione di 320x240 pixel, infine un array di 4 microfoni per l'ascolto di comandi vocali. Per entrambe le telecamere l'angolo di visione è di 57.5 gradi in orizzontale e 43.5 in verticale con tuttavia la possibilità di estendere l'ultima di 54 gradi grazie alla pedana di inclinazione, dotata di un motorino che fa ruotare il sensore per centrare automaticamente l'utente.

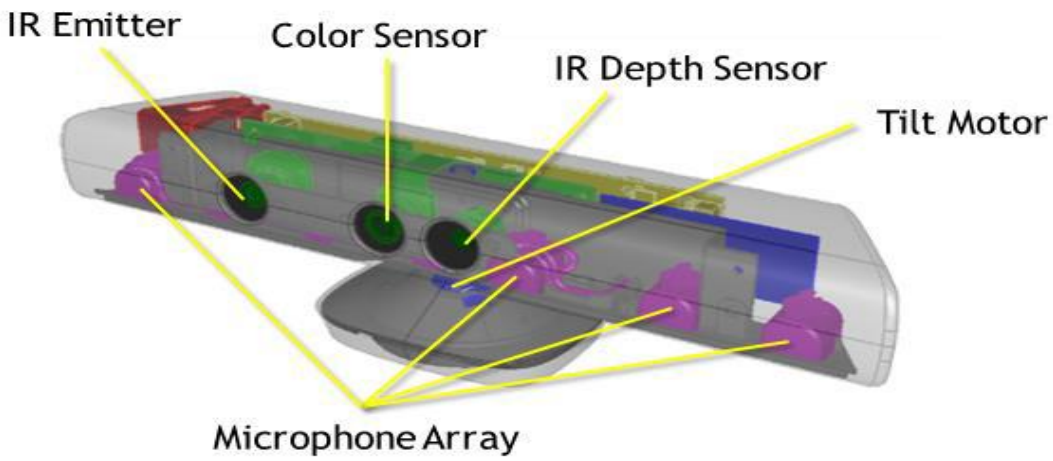


Fig.1 – Componenti caratteristiche del Kinect.

L'angolo di ascolto invece è di 100 gradi con la possibilità di discriminare l'origine con una risoluzione di 10 gradi ed una soglia che cancella automaticamente rumori sotto i 20 DB **[13].**

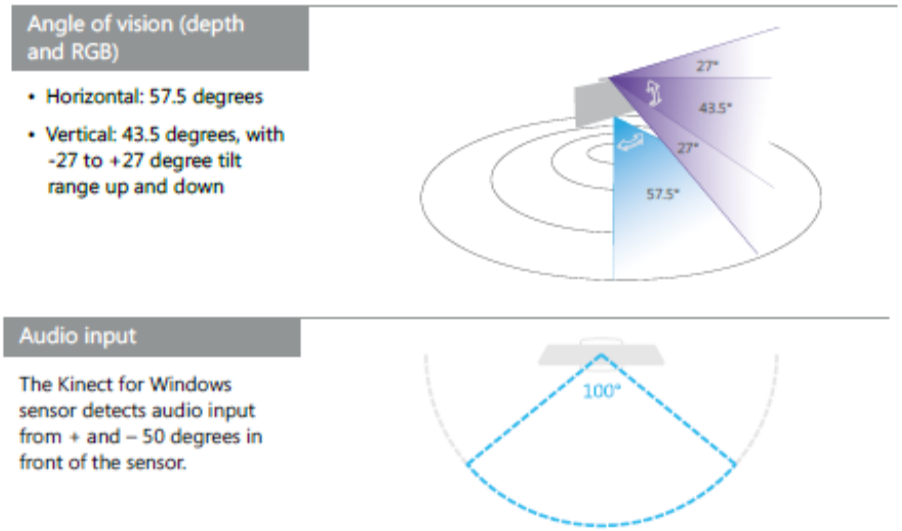


Fig.2 – Angolo di visione e di ascolto del Kinect per 360.

IL Kinect One presenta numerosi cambiamenti rispetto al primo, in particolare per quanto riguarda le telecamere. La telecamera RGB è stata promossa ad una versione HD con risoluzione di 1920x1080

sempre a 30 Hz. Il sistema proiettore-telecamera di profondità è stato sostituito invece con una tecnologia a tempo di volo, con risoluzione di 512x424 a 30 Hz. È stata così abbassata anche la latenza che per il Kinect era superiore ai 100 ms mentre per il V2 è di circa 60 ms.

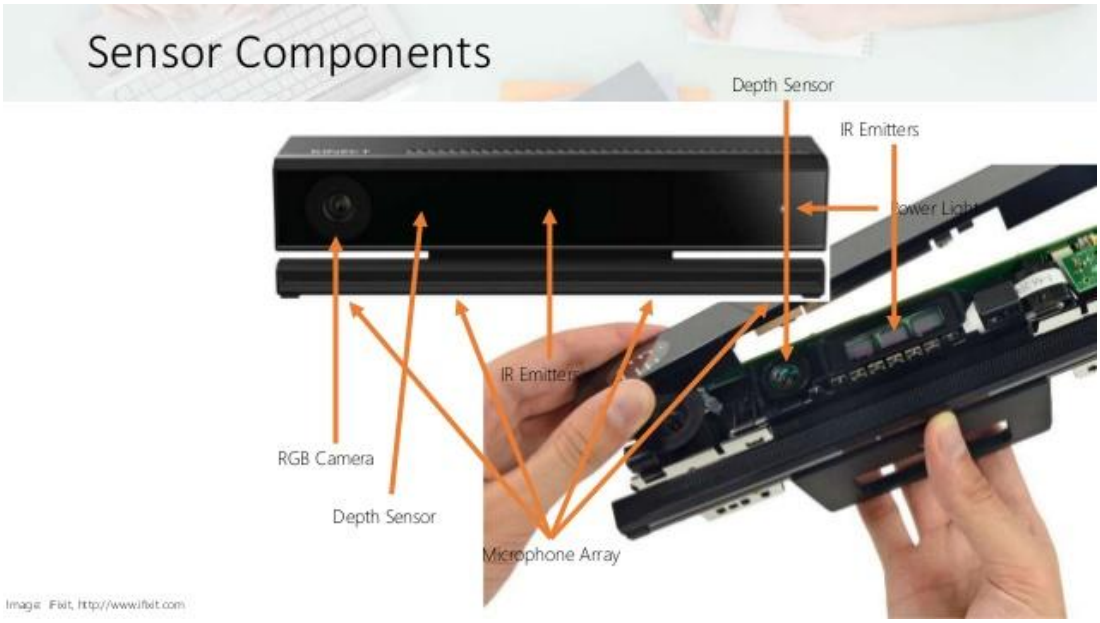
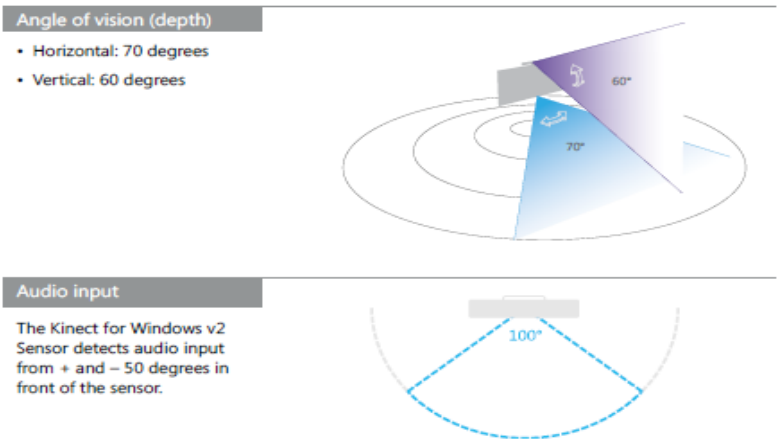


Fig.3 – Componenti caratteristiche del Kinect One.



Anche il Kinect One contiene 4 microfoni, ma questa volta con la capacità di discriminare l'origine del rumore di 5 gradi su un ventaglio di 100 gradi d'ascolto di fronte al sensore

Fig. 4 – Angolo di visione e di ascolto del Kinect One.

(± 50). Non è presente questa volta il motorino di inclinazione poiché l'angolo di visione è stato notevolmente incrementato: 60 gradi in verticale e 70 in orizzontale **[14]**.
In Tabella 2 sono quindi riassunte le caratteristiche fondamentali di entrambi i sensori per poterne apprezzare appieno le differenze.

Component	V1	V2
Technique	Structured-light	Time-of-flight
Depth Sensor	1.8 to 3.5 m	1.3 to 3.5 m
IR Depth Image	320 x 240	512 x 424
Colour Image	640 x 480	1920 x 1080
Infra-red Image	no IR	512 x 424
Audio Stream	16 kHz, 16-bit	48 kHz, 16-bit
Field of View hor:	57 degrees	70 degrees
Field of View ver:	43 degrees	60 degrees
Minimum Latency	102 ms	20-60 ms

Tab. 2 – Caratteristiche del Kinect (sx) e del Kinect One (dx).

Funzionamento

Sono descritte a questo punto le differenze sostanziali tra la tecnica della luce strutturata e quella a tempo di volo per determinare la profondità dello spazio 3D misurata dal Kinect. Nella prima, il proiettore invia un fascio di infrarossi strutturato con un pattern ben definito di punti e ricava poi le informazioni sulla profondità dalle deformazioni subite da questo pattern nell'incontrare oggetti nello spazio. Naturalmente più un oggetto è vicino al sensore, più grande è il diametro dei punti proiettati su esso. La telecamera a tempo di volo (in inglese time of flight camera, TOF-camera) invece, è uno

strumento che permette di stimare in tempo reale la distanza tra la telecamera e gli oggetti o la scena inquadrati, misurando il tempo che occorre ad un impulso luminoso per percorrere il tragitto telecamera-oggetto-telecamera (tempo di volo appunto). Il Kinect One infatti dispone di un emettitore di infrarossi a fianco della telecamera a infrarossi come si può vedere in fig. 3 e poiché la misura della distanza è conservata indipendentemente su ciascun pixel, non solo la risoluzione è maggiore, ma anche la precisione **[15]**. Per eventuali approfondimenti sulla questione, si rimanda all'articolo di Sarbolandi et al. **[16]** in cui gli autori hanno realizzato un interessante studio sul confronto delle due tecniche utilizzando proprio i due Kinect, annotando i pro e contro in diverse prove sperimentali.

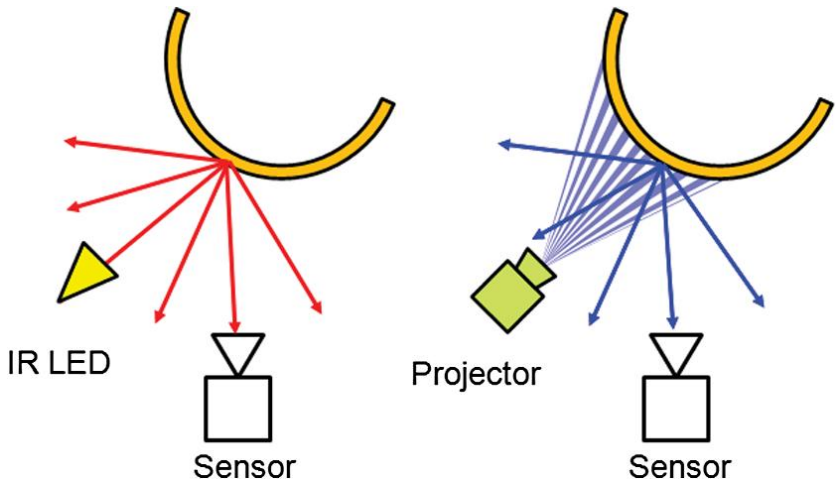


Fig. 5 – Principio fisico di funzionamento delle telecamere di profondità. A sx la tecnica a tempo di volo, a dx la tecnica della luce strutturata. Il sensore è una telecamera monocromatica per la lettura degli infrarossi con risoluzione maggiore in quella del Kinect V2.

Attenzione che non sempre, tuttavia, la maggior risoluzione della seconda versione lo rende il vincitore indiscusso. Nella maggior parte dei casi, specialmente se si vuole eseguire il body tracking, l'hand

tracking o il face tracking, è preferibile il Kinect One, in altri casi, ad esempio la scansione 3D di oggetti fisici, alcuni software di terze parti, tra cui anche i più diffusi e consigliabili, raccomandano l'uso del primo Kinect al secondo (ad esempio Skanect, http://skanect.occipital.com/?s=kinect+v2&submit=Go).

Entrambe le versioni hanno implementato nel Firmware l'algoritmo per il motion tracking del corpo umano, algoritmo basato su tecniche di machine learning con 1 milione di esempi come training set **[17]**. Alcuni autori han tuttavia preferito utilizzare altri modelli presi dalla letteratura sfruttando solo le informazioni date dalla telecamera di profondità **[18]** ad esempio nei casi di utilizzo in cui l'algoritmo del Kinect non è progettato per riconoscere la figura umana. L'algoritmo di tracking del Kinect V1 riconosce la silhouette di una persona dalle informazioni della telecamera di profondità, informazione contenuta dunque nelle dimensioni dei punti proiettati, in seguito quindi identifica 20 punti chiave, denominati joint e ne segue il movimento. Il V2 ne riconosce invece 25, andando ad aggiungere ai precedenti Joint anche i pollici ed il centro delle dita della mano come si può vedere in figura:

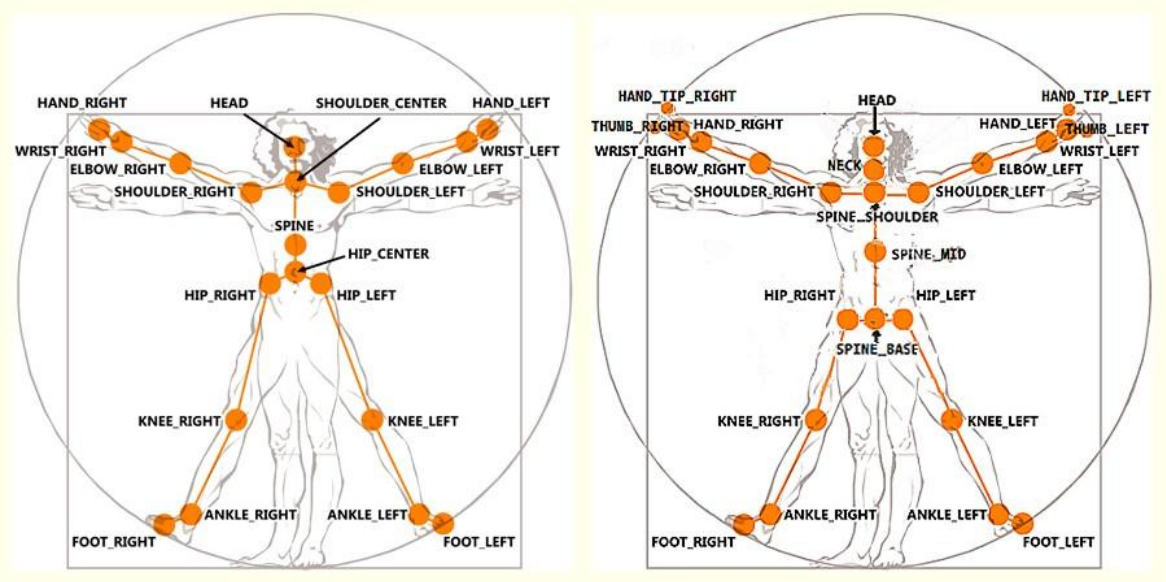


Fig. 6 – Joints riconosciuti dal Kinect V1 a sx e dal Kinect V2 a dx.

Si possono notare anche differenze sui joint che percorrono la spina dorsale con l'aggiunta del joint Neck nella seconda versione.
Per quanto riguarda la precisione del Kinect per 360 sia per il sensore di profondità che per l'algoritmo di tracking, diversi studi son stati fatti utilizzando sistemi di tracking ottico con marker come gold standard per il confronto **[9, 12, 19, 20, 21].** I vari studi riportano che mediamente l'errore commesso dal Kinect nell'individuare un joint non supera l'ordine dei centimetri, errore che tuttavia può propagarsi ed aumentare se ad esempio l'informazione è utilizzata per calcolare angoli articolari **[9]**. Per il Kinect V2 invece gli studi di ingegneria inversa sul sensore non sono ancora arrivati ad essere così approfonditi. Si potrebbe pensare di continuare a considerare le misure di precisione analizzate per il primo sensore e ritenerle comunque valide per il secondo, ma così facendo si commetterebbe una grossa imprudenza. Considerando gli aggiornamenti notevoli nell'hardware e nel firmware e visto che la precisione del primo Kinect è si buona, ma assolutamente non eccellente, si ritiene più corretto ricominciare la ricerca e verificare almeno con quanto margine di affidabilità il Kinect One possa sostituire nella pratica clinica strumentazioni accurate come quelle basate sui marker o sul tracking magnetico. In letteratura sono già comparsi naturalmente articoli sul Kinect v2 ed alcuni di questi sono stati utilizzati come punti di partenza per alcuni argomenti di questo progetto di tesi. A partire dal prossimo capitolo verranno presentati i procedimenti seguiti per analizzare la precisione in statica ed in dinamica dell'algoritmo di tracking del Kinect One fino ad arrivare a stillare una sorta di elenco di linee guida, o per meglio dire di consigli sviluppati con l'esperienza, per l'utilizzo di questo sensore in clinica.
Tornando all'argomento di questo capitolo, si possono citare differenze ulteriori insite nel codice dei sensori: il Kinect 1.0 disponeva infatti di due modalità di utilizzo, Full skeleton mode e Seated mode, quest'ultima con il tracking dei soli 10 joint superiori. Nel Kinect One questa modalità seduto è stata eliminata ed il tracking avviene sempre

full body, anche con l'utente seduto, purché sia visibile al sensore il viso dell'utente ed il joint denominato Mid-Spine (vedi figura 6). Le parti di corpo eventualmente non viste sono costruite poi dal modello interno al firmware del Kinect. La seconda differenza è sul numero di persone traccabili contemporaneamente: 2 per il Kinect e 6 per il Kinect One. Il tracking di più persone tuttavia spinge molto al limite le prestazioni del PC collegato al Kinect (per questa tesi è stato utilizzato un hp pavillon 15, processore intel core i5 @ 2.60 GHz-3.1GHz, 4 GB RAM) anche a seconda del programma utilizzato.

Per entrambi i sensori la telecamera di profondità ha come valore minimo 80 cm, esiste tuttavia una modalità chiamata Near Mode che permette di ridurre questo limite a 50 a scapito dell'accuratezza. Nel Kinect v1 come per la Seated mode vengono traccati solo i 10 joints in alto. In realtà nel Kinect One non è una vera e propria modalità attivabile, ma il sensore ha capacità di traccare il corpo da 0.5 cm a 4.5 m. É riportato sull'human Interface Guideline dei due sensori **[13, 14]** che le zone migliori per il traccaggio sono tra 1 ed i 4 m circa, studi più accurati hanno invece dimostrato che il sensore di profondità è più preciso entro i 3 metri **[22]**. Un ultimo discorso va espresso sul face tracking, disponibile per entrambi i Kinect, ma nettamente superiore nel secondo **[15]**, che ovviamente grazie alla maggior risoluzione permette di individuare (grazie anche ad un classificatore interno al firmware) l'apertura/chiusura degli occhi e l'espressione felice/triste del viso. Parlare del face tracking del Kinect richiederebbe la stesura di un'altra tesi, per ulteriori chiarimenti o curiosità si rimanda al sito MSDN **[23, 24]**. Riepilogando questo capitolo in vista del successivo si può quindi dire che per il Kinect V1 è possibile trovare in letteratura tantissimi studi di ingegneria inversa o prove sperimentali che analizzano e dimostrano le più svariate caratteristiche del sensore **[25]**. Molti articoli inoltre danno consigli utili su come, ad esempio, incrementare la precisione del tracking o su come sfruttare le informazioni della telecamera di profondità per poi utilizzare algoritmi

sviluppati autonomamente. Per il Kinect V2 tutti queste informazioni, nel limite delle mie capacità di ricercarle, non sono state ancora analizzate, o almeno solo parzialmente. Ci sono studi sulla precisione della telecamera di profondità, che si dimostra essere la migliore tra quel quelle low-cost, ma poco riguardo il tracking; nel prossimo capitolo verranno presentati i software utilizzati e verrà discussa una prima modalità di valutazione qualitativa del sensore. In seguito verrà studiata la precisione in statica in alcune situazioni ambientali e si cercherà la distanza ottimale dal sensore in cui è preferibile far lavorare l'utente.

Capitolo 3

Descrizione dei software e prima analisi

In questo capitolo verrà presentato il software creato per poter lavorare con il Kinect One. Per non limitarsi alle funzionalità di quelli disponibili nell'SDK è stato più proficuo costruirne uno da zero utilizzando il linguaggio di programmazione C# e l'ambiente di sviluppo Visual Studio 2013. Il software così creato permette la visualizzazione della telecamera RGB, infrarossi e di profondità a scelta, permette la visualizzazione del body tracking plottando in real-time i joints collegati tra loro ed infine il salvataggio di molte informazioni utili per un analisi off-line seguente.

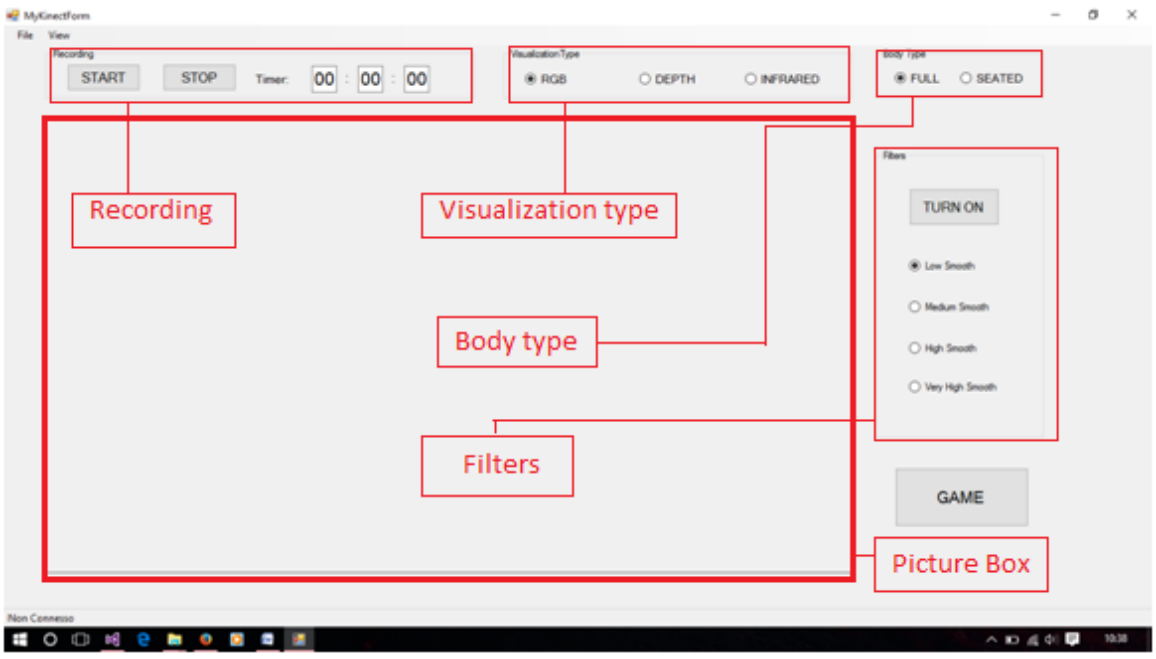


Fig. 7 – Panoramica del Software realizzato.

Per questo progetto è stato scelto di programmare in C# realizzando un Software di tipo Windows Form che avesse le caratteristiche necessarie a realizzare la prima analisi sulla precisione. In seguito il codice è stato aggiornato con l'aggiunta del filtraggio delle posizioni dei Joint (argomento del capitolo 5) ed infine un piccolo gioco realizzato per valutarne l'efficacia in movimento e per testare le possibilità offerte dal filtraggio.
Si può vedere in Figura 7 che per la visualizzazione del flusso di frame è stata scelta una Picture box le cui dimensioni sono 960x540 (l'esatta metà del formato della camera RGB).
Dopo aver aggiunto le librerie del Kinect alle referenze, per accedere alle informazioni del Kinect collegato al PC ed eventualmente spegnerlo in caso di chiusura del programma, sono stati generati 2 eventi rispettivamente all'apertura del Form e alla chiusura:

```
private void Form1_Load(object sender, EventArgs e)
        {

            try
            {

                KinectSensor KinSensor = KinectSensor.GetDefault();
              CoordinateMapper coordinateMapper =
             KinSensor.CoordinateMapper;


                if (KinSensor == null)
                {
                    return;
                }

                KinSensor.Open();
              MultiSourceFrameReader KinReader =
            KinSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Color
            |    FrameSourceTypes.Depth | FrameSourceTypes.Infrared |
            FrameSourceTypes.Body);
```

```
                    // connect up the video event handler

                    KinReader.MultiSourceFrameArrived +=
                    Reader_MultiSourceFrameArrived;
                }
                catch
                {
                  MessageBox.Show("Inizializzazione fallita", "Camera
                 viewer");
                    Application.Exit();
                }
            }

        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
            {
                if (KinReader != null)
                {
                    KinReader.Dispose();
                }

                if (KinSensor != null)
                {
                    KinSensor.Close();
                }
            }
```

Le due classi, `KinectSensor` e `MultiSourceFrameReader`, permettono di accedere alle informazioni del sensore e di leggere il flusso dei dati delle camere e del body tracking. Rispetto al Kinect V1 è stata tolta la possibilità di scegliere a quale Kinect accedere (se ad esempio sono collegati più Kinect allo stesso PC) tanto che viene utilizzato il solo comando getDefault(). In genere tutto l'API del Kinect One risulta più semplificato rispetto a quello del V1, tuttavia per motivazioni varie sono state eliminate anche alcune cose che prima erano presenti e sarebbero state utili anche in questa versione, come i filtri che vedremo tra due capitoli. A questo punto per leggere lo stream dei

frame viene lanciato l'evento: `KinReader.MultiSourceFrameArrived += Reader_MultiSourceFrameArrived`.

```
private void Reader_MultiSourceFrameArrived(object sender,
MultiSourceFrameArrivedEventArgs e)
        {
            try
            {
                var reference = e.FrameReference.AcquireFrame();

                // Apre un frame RGB

                using (var frame=
                reference.ColorFrameReference.AcquireFrame())
                   {

                       //Codice per trasformare il frame in una bitmap da
                       //mettere nella picture box
                   }

                // Apre un frame della camera di profondità

                using (var frame=
                reference.DepthFrameReference.AcquireFrame())
                   {

                       //Codice per trasformare il frame in una bitmap da
                       //mettere nella picture box
                   }

                // Apre un frame di tipo infrarosso

                using (var frame=
                reference.ColorFrameReference.AcquireFrame())
                   {

                       //Codice per trasformare il frame in una bitmap da
                       //mettere nella picture box
                   }
```

```
            //apre un Frame di tipo Body

            using (var frame=
            reference.BodyFrameReference.AcquireFrame())
                {
                    if (frame != null)
                    {

                        Bodies = new
                        Body[frame.BodyFrameSource.BodyCount];

                        frame.GetAndRefreshBodyData(Bodies);

                        foreach (var body in Bodies)
                        {
                            if (body != null)
                            {

                                // Disegna lo scheletro.


                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message + ex.StackTrace);
                }
            }
```

Per il codice completo, si rimanda all'appendice A.

Con uno switch si rende possibile selezionare i 3 tipi di immagine selezionabili dal Form nel gruppo di bottoni denominato Visualization type, nel menu a tendina invece sotto il bottone View è possibile decidere se vedere il Body e/o il flusso di frame delle telecamere. In

Figura 8 è possibile vedere le 4 modalità di visione complete anche di Body tracking.

Member	Value	Description
AnkleLeft	14	Left ankle
AnkleRight	18	Right ankle
ElbowLeft	5	Left elbow
ElbowRight	9	Right elbow
FootLeft	15	Left foot
FootRight	19	Right foot
HandLeft	7	Left hand
HandRight	11	Right hand
HandTipLeft	21	Tip of the left hand
HandTipRight	23	Tip of the right hand
Head	3	Head
HipLeft	12	Left hip
HipRight	16	Right hip
KneeLeft	13	Left knee
KneeRight	17	Right knee
Neck	2	Neck
ShoulderLeft	4	Left shoulder
ShoulderRight	8	Right shoulder
SpineBase	0	Base of the spine
SpineMid	1	Middle of the spine
SpineShoulder	20	Spine at the shoulder
ThumbLeft	22	Left thumb
ThumbRight	24	Right thumb
WristLeft	6	Left wrist
WristRight	10	Right wrist

È stata introdotta anche una sorta di Seated mode fittizia, selezionabile dal Button Group: BodyType, rendendo visibili solo i joints superiori; è nettamente preferibile utilizzare questa configurazione nel caso si eseguano esercizi da seduto per non confondere l'utilizzatore. A questo riguardo, viene mostrato nella tabella a fianco l'elenco dei 25 Joint così come sono denominati e numerati dal Kinect. Poiché le coordinate dei Joint sono date in metri e si riferiscono alle posizioni rispetto alla telecamera di profondità, secondo una terna ortogonale che ha l'origine al centro di questa, esiste una classe, denominata CoordinateMapper, nella libreria del Kinect che permette di coordinare le informazioni di profondità a quelle delle telecamere ad infrarossi ed RGB.

Tab. 3 – Enumeration dei Joints de Kinect One secondo lo schema fornito da Microsoft.

In questo modo è facile proiettare le informazioni spaziali 3D sulle immagine delle telecamere 2D, scalando opportunamente i dati e trasformando i metri in pixel. Le posizioni dei Joint sono riferite ad una terna ortogonale che ha l'origine al centro del sensore di profondità e asse X, diretto orizzontalmente verso destra (guardando il Kinect da davanti), Y verticale, positivo verso l'alto e Z orizzontale diretto frontalmente al sensore e con verso positivo uscente.
La parte adibita alla registrazione in questo software è predisposta per dare 2 secondi di tempo alla persona per sistemarsi ed in seguito registrare 30 secondi di tracking.

Fig. 8 – In alto a sx telecamera RGB con risoluzione 1920x1080, in alto a dx telecamera ad infrarossi con risoluzione 512x424, in basso a sx camera di profondità con risoluzione di 512x424, in basso a dx body tracking.

Al termine della registrazione vengono salvati in due file .txt le posizioni dei 25 joint in coordinare x,y,z, lo stato di Tracking del Joint (joint traccato o ricostruito dall'algoritmo) e l'eventuale filtraggio.

```
-0,157775 -0,3211971 1,109844 1 0 0 0
-0,173303 0,01920975 1,152899 1 0 0 0
-0,1847593 0,33702 1,176272 1 0 0 0
-0,1888895 0,4942981 1,176803 1 0 0 0
-0,333443 0,197319 1,134839 1 0 0 0
-0,5257314 -0,007154748 1,121022 1 0 0 0
-0,5020921 0,2601171 1,021409 1 0 0 0
-0,4954457 0,3260415 1,016553 1 0 0 0
0,0003272554 0,2025689 1,194709 1 0 0 0
0,1777735 -0,01558507 1,230166 1 0 0 0
0,192075 0,2489515 1,118876 1 0 0 0
0,1871908 0,3409842 1,116927 1 0 0 0
-0,2346522 -0,309755 1,060266 1 0 0 0
-0,3003222 -0,7360373 1,058817 0 0 0 0
-0,3679493 -1,140199 1,087902 0 0 0 0
-0,3547319 -1,17495 0,9608827 0 0 0 0
-0,07046872 -0,3112994 1,08541 1 0 0 0
-0,05541038 -0,7422782 1,096758 0 0 0 0
-0,04346319 -1,150838 1,138341 0 0 0 0
-0,02041913 -1,186049 1,0129 0 0 0 0
```

```
1  SpineBase  Tracked  non filtrato
1  SpineMid  Tracked  non filtrato
1  Neck  Tracked  non filtrato
1  Head  Tracked  non filtrato
1  ShoulderLeft  Tracked  non filtrato
1  ElbowLeft  Tracked  non filtrato
1  WristLeft  Tracked  non filtrato
1  HandLeft  Tracked  non filtrato
1  ShoulderRight  Tracked  non filtrato
1  ElbowRight  Tracked  non filtrato
1  WristRight  Tracked  non filtrato
1  HandRight  Tracked  non filtrato
1  HipLeft  Tracked  non filtrato
1  KneeLeft  Inferred  non filtrato
1  AnkleLeft  Inferred  non filtrato
1  FootLeft  Inferred  non filtrato
1  HipRight  Tracked  non filtrato
1  KneeRight  Inferred  non filtrato
1  AnkleRight  Inferred  non filtrato
1  FootRight  Inferred  non filtrato
```

*Fig. 9 – Esempi di file *.txt salvati dal programma. A sx, in ordine, coordinate x,y,z, joint traccato (1) o ricostruito (0) e coordinate dei Joint filtrate (0 0 0 se il filtraggio non è presente). A dx, in ordine, numero di frame, nome del joint, traccato (Tracked) o ricostruito (Inferred) ed infine tipo di filtro applicato.*

Per valutare la precisione del Kinect dai dati registrati si è reso necessario sviluppare un altro software, questa volta in Matlab, per implementare l'analisi offline. Nella fase iniziale è stata creata una GUI (GRAPHIC USER INTERFACE) che dopo aver caricato un file .txt della tipologia con solo valori numerici, permettesse di rivedere il body tracking frame-per-frame, attraverso una slide bar, in 2D (x-y) o in 3D. È stata introdotta la possibilità di selezionare un joint e vederne il plot di distanza dall'origine nel tempo, di velocità e di accelerazione (calcolate con le differenze finite) e lo stato di Tracking ad ogni frame. Infine sono stati inseriti i bottoni per visualizzare il nome e la posizioni dei Joint come in figura 6 (References) e per visualizzare il numero di frame in cui ogni joint non è stato traccato dal Kinect e le relative deviazioni standard in x, in y ed in z.

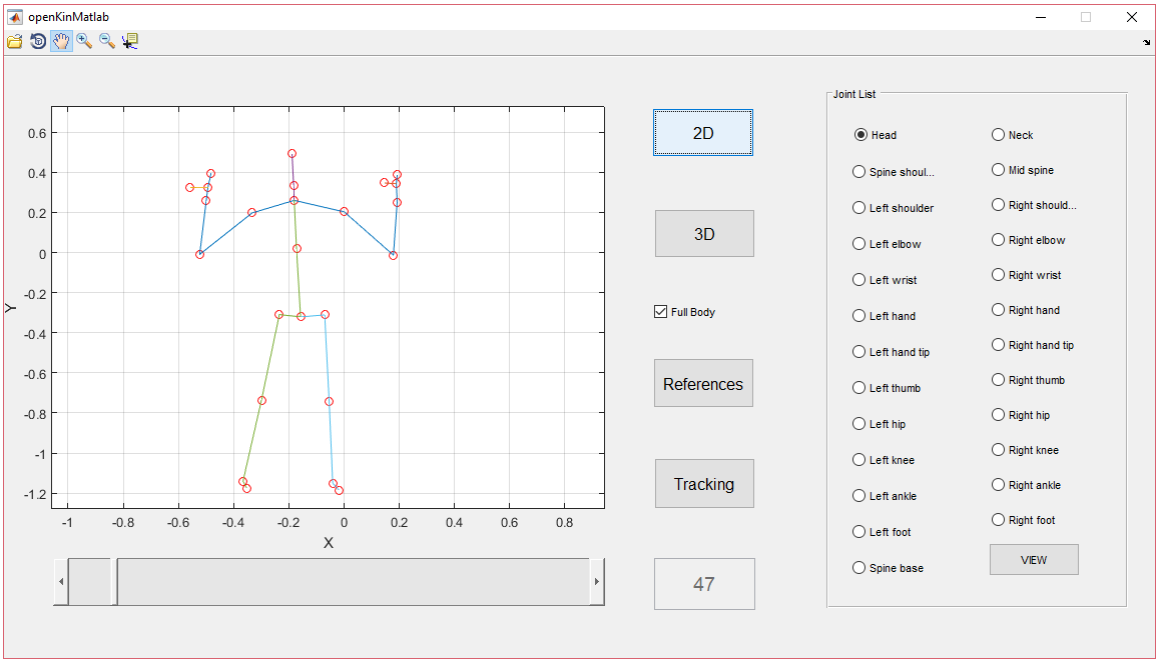


Fig. 10 – Panoramica della GUI Matlab per l'analisi offline.

Per i dettagli sul codice completo si rimanda all'Appendice B.

I processi decisionali che hanno portato a questo design finale sono stati in realtà lunghi ed elaborati. Inizialmente infatti l'idea era di concentrarsi su un metodo per classificare i Jitter, degli artefatti più o meno gravi dovuti all'algoritmo di tracking che portano alcuni joint a "saltare" da una posizione ad un'altra con uno spostamento che non ha corrispondenze con alcun gesto dell'utente. Il problema dei Jitter è forse il più grave nell'utilizzo del Kinect e le cause di questi sono assumibili esclusivamente al modello interno al Firmware che, se riconosce una figura umana (Body) davanti a sè, ad ogni Frame è programmato per dare in uscita le coordinate di tutti i 25 Joint, anche di quelli che per qualche motivo non sono visibili dalla camera. Poiché, per loro natura, i Jitter sono impredicibili, la prima soluzione ideata è stata di realizzare un classificatore che in tempo reale li individuasse per escluderli o anche solo contarli. Così nella GUI è stata inserita la possibilità di selezionare un Joint per vederne la cinematica.

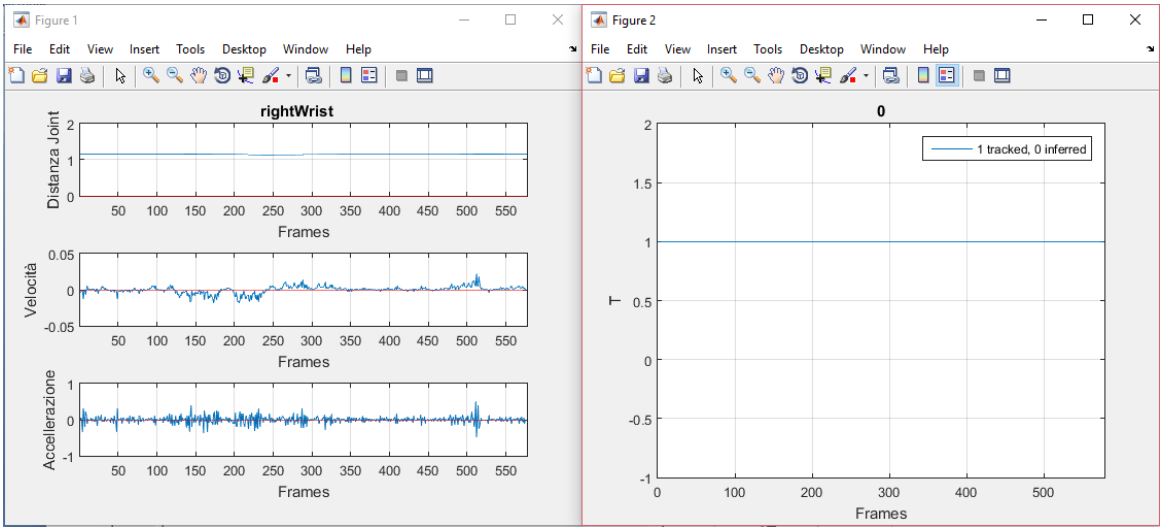


Fig. 11 – A sx plot di distanza del Joint dall'origine, velocità calcolata come derivata della distanza ed accelerazione calcolata come derivata della velocità. A dx stato di traccaggio del medesimo Joint.

In questo modo dopo diverse prove in cui sono stati eseguiti semplici esercizi come la presa di un oggetto o il cammino davanti alla camera sono stati ottenuti molti dati che tuttavia non hanno portato ai risultati sperati. È stato chiaro fin da subito infatti che le ampiezze dei Jitter erano troppo variabili per poterli semplicemente classificare tramite una soglia. In particolare durante esercizi dinamici dove ogni Joint ha entità diverse di movimento è risultato difficile individuare con chiarezza un Jitter dallo spostamento stesso e dal rumore bianco che comunque è sempre presente in un sensore. La soluzione avrebbe dovuto tener conto contemporaneamente degli spostamenti in termini di distanza assoluta del Joint e nelle tre coordinate x, y, z. Il classificatore avrebbe dovuto farlo inoltre per tutti i 25 Joint in real-time per poter essere applicato alla pratica clinica. Nella ricerca di una soluzione meno computazionalmente onerosa è stato naturale spostarsi dalla classificazione al filtraggio. È stata presa in considerazione anche l'analisi in frequenza cercando di individuare dei possibili range di frequenze che corrispondessero ad un Jitter dallo

spettro di potenza. L'idea iniziale è stata di analizzare il segnale nella banda 0-15 Hz, dato che 15 Hz è pari alla metà della frequenza di campionamento, ricordando che il Kinect acquisisce i frames a 30 Hz. Se effettivamente fosse stato chiaro vedere ad un certa frequenza un picco evidente nelle registrazioni in cui il fenomeno è accaduto, con un semplice filtro passa banda sarebbe stato possibile eliminarlo senza dover più ricorrere ad un classificatore nel dominio del tempo. Purtroppo non è stato così in quanto non è stato possibile individuare nessuna banda specificamente impattata dal fenomeno "Jitter". Essendo il fenomeno un salto casuale della posizione di un Joint ad un'altra, nel tempo di un solo frame, distante anche qualche centimetro, le componenti spettrale associabili ad esso sicuramente vengono a trovarsi oltre i 15 Hz, probabilmente verso i 30 Hz andandosi a mischiare sotto forma di aliasing con le componenti spettrali a bassa frequenza.

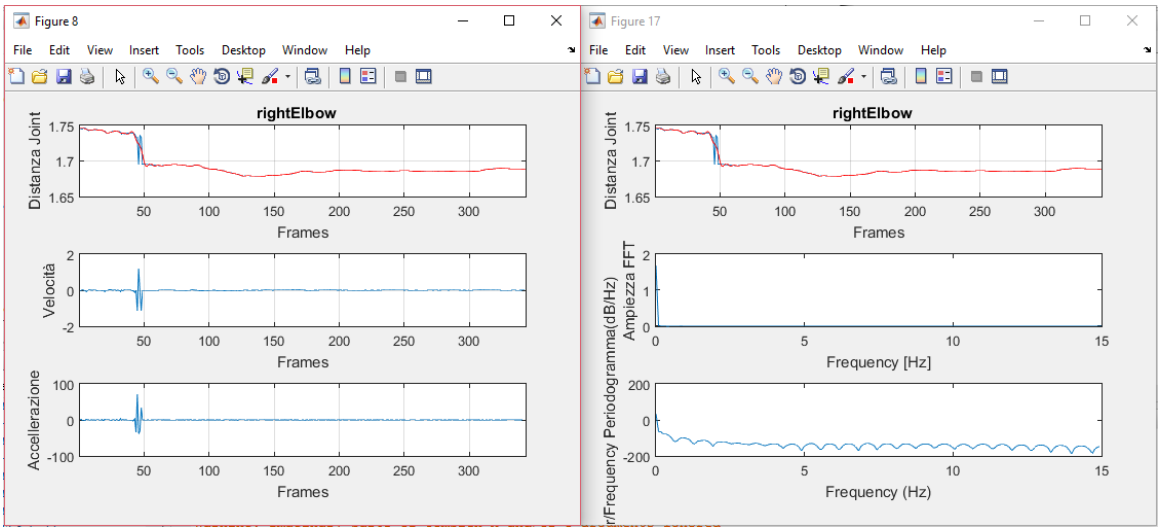


Fig. 12 – Dettaglio del fenomeno denominato Jitter al frame 50. A sx analisi nel tempo; in questo caso il Jitter sarebbe facilmente identificabile con una soglia. A dx in ordine dall'alto al basso sono plottati il segnale originale, l'FFT del segnale ed infine lo spetto di potenza. In rosso tentativo di filtraggio passa basso con filtro di Butterworth con frequenza di taglio a 5 Hz. Registrazione effettuata durante una posa statica.

Con un filtro passa-basso a 15 Hz prima della trasformata di Fourier non si è riusciti a migliorare il segnale e, come si può vedere in rosso nella figura precedente, per eliminare i picchi di Jitter è stato necessario abbassare la frequenza di taglio a 5 Hz. Nell'ottica però di voler verificare le condizioni migliori in cui lavorare, piuttosto che filtrare i Jitter si sarebbe preferito contarli in un numero predefinito di secondi. Tuttavia l'impossibilità di isolare il fenomeno da altre componenti spettrali ha reso questa ricerca poco proficua. Non è chiaro se ad ostacolare ciò abbia contribuito solo il rumore di fondo (rumore bianco), il leggero tremore del soggetto o altri fattori. Quello che è chiaro invece è che ragionando anche solo in statica non risulta sempre facile realizzare un processo per individuare automaticamente i Jitter. Pensare di applicare poi lo stesso metodo in esercizi dinamici rende il problema molto più complesso, in quanto questi fenomeni indesiderati sono nascosti anche dal movimento dell'utente oltre che dal rumore. A questo punto l'analisi alle derivate ed in frequenza ha permesso almeno di capire che con un opportuno filtraggio, ed in determinate condizioni, è possibile limitare almeno in parte l'impatto di questi eventi. Si è deciso così di affidarsi totalmente all'uso di filtraggi in tempo reale, descritti nel seguito.

Prima di arrivare a questo, però si è deciso di fare un passo indietro ed investigare le migliori condizioni in cui allestire gli esperimenti. Facendo diverse prove in statica in particolare si è voluto ricercare la miglior distanza dal Kinect per l'utente, la miglior altezza cui tenere il sensore per poterlo utilizzare in più situazioni possibili e le eventuali influenze negative sulle misure inficiabili a fattori come l'illuminazione ambientale, la presenza di oggetti sulla scena o le caratteristiche antropometriche della persona. Contemporaneamente si è potuto dare anche una stima sulla precisione dell'algoritmo di tracking in statica.

Capitolo 4

Analisi sulla precisione dell' algoritmo di Body Tracking in statica

Studiando l'articolo "Evaluating and Improving the Depth Accuracy of Kinect for Windows v2" **[22],** in cui viene analizzata la precisione della telecamera di profondità del Kinect One, emerge che il sensore mostra ottime performance all'interno di una certa area di lavoro, peggiorando all'allontanarsi da questa.

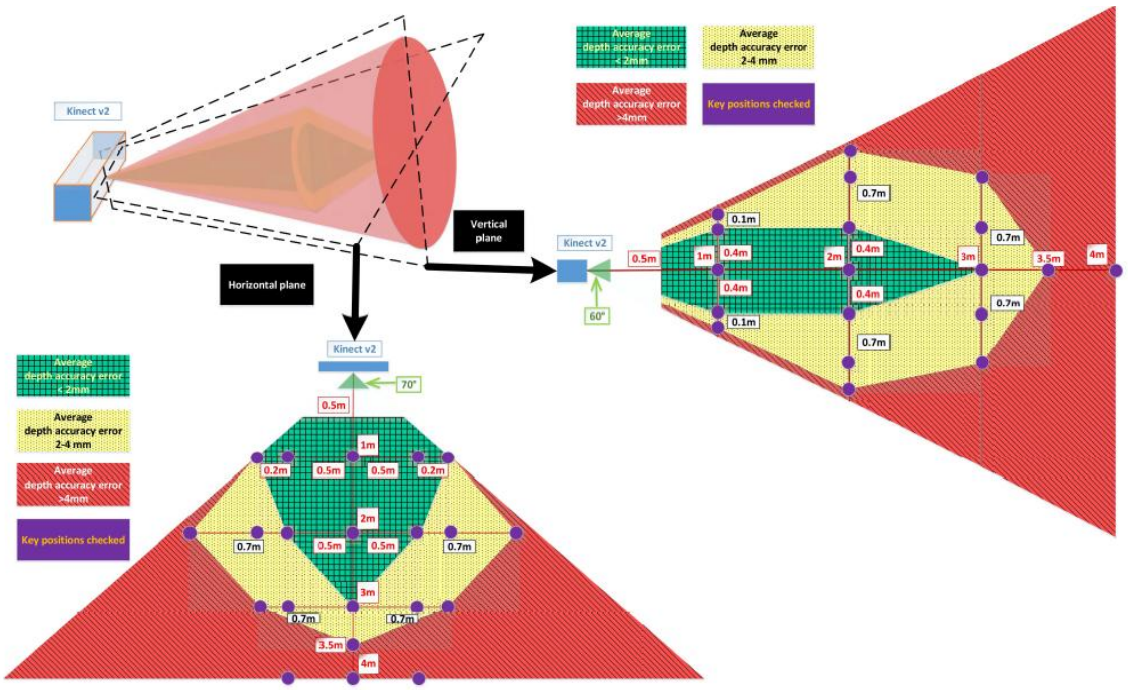


Fig. 13 – Misura della precisione del sensore di profondità del Kinect One ad opera di Yang et al. [22]. Per le misure è stato posto uno schermo quadrato di 48 cm di lato nelle posizioni indicate dai punti viola, valutando l'errore sulla misura di profondità

della superficie. Nelle zone versi l'errore è risultato essere minore di 2 mm, tra i 2 e i 4 mm nelle zone gialle e maggiore di 4 mm nella rossa.

Gli autori riportano inoltre che ad influenzare le performance del Kinect possono essere elementi come la presenza di oggetti riflettenti (come gli specchi o le finestre) che deviano o non riflettono gli infrarossi inviati dal Kinect rendendo impossibile la misura di profondità in quei punti. In maniera simile, creano problemi di interpretazione degli infrarossi anche la presenza di oggetti coperti da materiale luce-assorbente o anche l'arrivo al sensore di infrarossi non prodotti dal proprio emettitore, come nel caso di luce solare diretta o da altre sorgenti sulla scena.
Nei vari articoli letti si parla sempre di accuracy senza mai specificare se è intesa nel senso comune o nella definizione ISO-5725; tuttavia per l'interesse di questa tesi, entrambe le definizioni possono essere considerate valide. L'idea di stimare la presenza di piccoli errori sistematici nelle posizioni dei Joint, utilizzando sistemi molto accurati come gold standard, non sembrerebbe molto interessante, almeno dal punto di vista clinico. Non è ben chiarito inoltre dalla documentazione tecnica Microsoft l'esatto punto in cui il Joint dovrebbe essere posizionato dal Firmware del Kinect in posizioni ottimali. Nell'articolo di Shotton **[17]** è riportato che dopo aver rimosso il background la figura umana è divisa in parti (cluster) e le posizioni dei Joint sono calcolati come centroidi di questi tramite una variante dell'algoritmo di clustering Mean Shift. Tuttavia essendo un metodo basato sull'elaborazione delle immagini e non su parti anatomiche reali, risulterebbe difficile il posizionamento dei marker per effettuare i confronti. In secondo luogo l'algoritmo del Kinect è fornito di una classe per coordinare le informazioni di camera con quelle di Body, per cui visualizzando a schermo i Joint e contemporanemate i frame della camera RGB o infrarossi, è possibile con ragionevole certezza verificare che le posizioni di questi siano corrispondenti alle controparti

anatomiche. Naturalmente in questo modo non è possibile dare una valutazione quantitativa sull'accuratezza, ma almeno è possibile vedere che le posizioni stimate non siano totalmente errate. Per questi motivi si è scelto di analizzare la precisione, dando per valida l'accuratezza delle misure.

Partendo dallo studio proposto ad inizio capitolo, si è deciso di proseguire analizzando le deviazioni standard sulle posizioni dei 25 joint in 30 secondi di acquisizione, per valutare in questo modo la precisione dell'algoritmo di tracking.

Le registrazioni sono state effettuate con 3 soggetti: M.M., maschio, altezza 1.70m e peso 70 Kg, A.S. maschio, altezza 1.85 e peso 90 Kg, G.S. femmina, altezza 1.70 e peso 50 Kg.

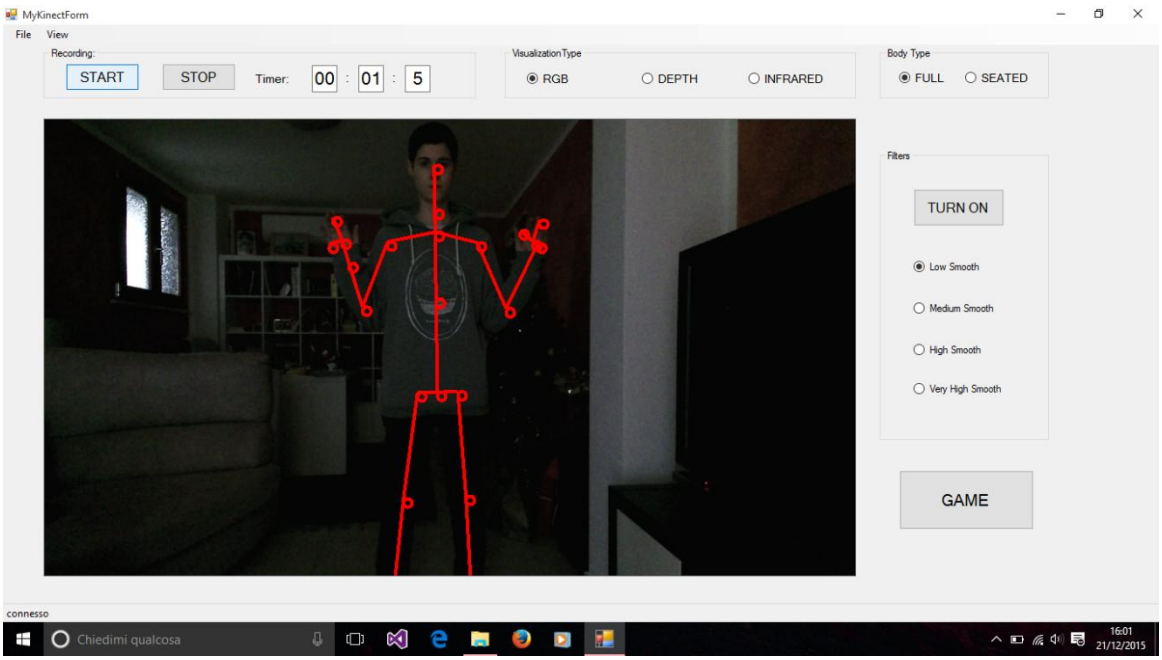


Fig. 14 – Posizione tenuta dai 3 soggetti nelle acquisizioni statiche.

Le misure sono state effettuate tenendo il soggetto in posizione statica a mani in alto, come in figura precedente.

Questa posizione è risultata essere la più comoda da tenere per i 30 secondi di registrazione, facendo vedere al contempo la maggior parte dei joint alla camera. Sono state effettuate misure variando distanza

ed altezza del Kinect secondo lo schema: altezza 1 m, distanze 1.0 m, 1.5 m, 2.0 m, 2.5 m, altezza 1.5 m stesse distanze, altezza 2 m, stesse distanze. Le misure sono state effettuate con un regolare metro a nastro metallico posto a terra e con le punte dei piedi del soggetto poste alla distanza di misura.

Per realizzare più facilmente le misure si è realizzato un altro Form in C# per vedere unicamente la camera RGB ed il Body tracking, un tasto per registrare un numero di secondi predefinito e due textbox per impostare altezza del sensore e distanza dell' utente per nominare automaticamente il file di salvataggio. I dati ottenuti dalle registrazioni sono stati divisi quindi in gruppi con altezza fissa o distanza fissa e per ogni Joint è stata valutata la distanza dall'origine ad ogni frame. Per ogni prova, misurando la deviazione standard della distanza, sono state così ottenute delle matrici 25x4 (nei gruppi ad altezza fissa) o 25x3 (nei gruppi a distanza fissa). Su ciascuna di queste matrici è stata calcolata la media per colonna ed è così stata individuata la misura di distanza, o di altezza, che permettesse di avere le deviazioni standard minori. In seguito è stato eseguito un test ANOVA ad un fattore sulle stesse per determinare se effettivamente le altre misure fossero significativamente diverse in senso statistico. Il test è in seguito stato ripetuto dividendo la matrice 25xn in 3 sottomatrici contenenti: la prima i 12 Joint di arto superiore (spalla compresa), la seconda i 5 Joint di testa e tronco e l' ultima gli 8 Joint di arto inferiore (anca inclusa). Questa divisione del corpo sembra molto interessante da investigare sia perché in questo modo è possibile vedere se il tracking di un distretto corporeo presenta, in media, minore rumore degli altri, sia per valutare se effettivamente è diverso valutare tutto il Body o delle sottocategorie. La mano con i Joint hand, handTip e Thumb meriterebbe un gruppo a sè, in quanto generalmente questi Joint hanno le deviazioni standard più alte (sono più rumorosi lungo z), tuttavia si è preferito includerli semplicemente nell'arto superiore. Una stima di questo genere si pensa possa giovare a chi studia il tracking di

arto superiore, settore generalmente più investigato per l'uso del Kinect **[25]**. I test ANOVA sono stati eseguiti separatamente sulle 3 categorie per tutte le misure effettuate. Affinché sia verificata l' ipotesi H0, ovvero che le misure sino tutte statisticamente uguali, si è controllato che F0 calcolato con il test fosse minore del valore di F da tabella con errore pari a 0.01. Per le prove con Kinect ad altezza fissa i valori di F sono: 3.95 se si confrontano tutti i 25 Joint, 4.31 per l' arto superiore (12 Joint x 4 misure), 5.29 per il tronco (5 Joint x 4 misure) e 4.57 per l' arto inferiore (8 Joint x 4 misure). Quando invece ad essere fissa è la distanza dell' utente valgono: 4.98 per tutto il corpo, 5.39 per l' arto superiore, 5.78 l'arto inferiore ed infine 6.93 per il segmento tronco-testa. In caso di differenza significativa tra le parti, si è deciso di effettuare un confronto tra i dati, eseguendo ripetutamente dei t-test con intervallo di confidenza al 99%, tra le coppie di misure. Per brevità si valutano solo i dati accoppiati con la misura con la media delle deviazioni standard minore. I risultati sono di seguito esposti nella forma:

prova: arto superiore = misura con media dev.std. minore (dev.std.) → F0, si accetta o rigetta H0, c'è o non c'è differenza significativa tra coppie di dati.

ES. Altezza Kinect 1.0 m:

arto superiore = 1.0 m (3.8 mm) → F0=0.8, si accetta H0, nessuna differenza significativa con le altre distanze.

G.S.

Altezza Kinect 1.0 m:

Total Body = 1.0 e 2.0 (3.2 mm) → F0=16.3, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.0-2.0 e 1.0-2.5 m.

Arto superiore = 2.5 m (3.9 mm) → F0=17.0, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.0-2.0 e 1.0-2.5 m.

Tronco = 1.0 m (1.9 mm) → F0=49.6, si **rigetta H0**, nessuna differenza significativa limitata alle misure 1.0, 2.0 e 2.5 m.

Arto inferiore = 1.0 m (1.3 mm) → F0=1.39, si **accetta H0**, nessuna differenza significativa.

Altezza Kinect 1.5 m:

Total Body = 2.0 (4.6 mm) → F0=0.80, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.5 m (7.0 mm) → F0=0.48, si **accetta H0**, nessuna differenza significativa.

Tronco = 2.0 m (3.5 mm) → F0=6.55, si **rigetta H0**, significativamente differente dalle altre distanze.

Arto inferiore = 2.0 m (1.6 mm) → F0=0.87, si **accetta H0**, nessuna differenza significativa.

Altezza Kinect 2.0 m:

Total Body = 1.5 m (3.4 mm) → F0=2.46, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.5 m (4.8 mm) → F0=0.44, si **accetta H0**, nessuna differenza significativa.

Tronco = 1.5 m (2.5 mm) → F0=7.10, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.0-1.5 e 1.5-2.5 m.

Arto inferiore = 1.50 m (2.0 mm) → F0=1.89, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 1.0 m:

Total Body = 1.0 m (3.2 mm) → F0=2.11, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.0 m (5.1 mm) → F0=0.98, si **accetta H0**, nessuna differenza significativa.

Tronco = 1.0 m (1.9 mm) → F0=11.2, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-2.0 m.

Arto inferiore = 1.0 m (1.3 mm) → F0=11.4, si **rigetta H0**, significativamente differente dalle altre altezze.

Distanza Utente 1.5 m:

Total Body = 2.0 m (3.4 mm) → F0=11.0, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.5-2.0 m.

Arto superiore = 2.0 m (4.8 mm) → F0=10.6, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 2.0-2.5 m.

Tronco = 2.0 m (2.5 mm) → F0=34.6, si **rigetta H0**, significativamente differente dalle altre altezze.

Arto inferiore = 2.0 m (2.0 mm) → F0=0.89, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 2.0 m:

Total Body = 1.0 m (3.7 mm) → F0=2.87, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.0 m (5.2 mm) → F0=0.94, si **accetta H0**, nessuna differenza significativa.

Tronco = 1.5 m (3.5 mm) → F0=3.89, si **accetta H0**, nessuna differenza significativa.

Arto inferiore = 1.0 m (1.5 mm) → F0=2.45, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 2.5 m:

Total Body = 1.0 m (3.2 mm) → F0=3.05, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.0 m (3.9 mm) → F0=3.24, si **accetta H0**, nessuna differenza significativa.

Tronco = 1.0 m (2.2 mm) → F0=31.2, si **rigetta H0**, significativamente differente dalle altre altezze.
Arto inferiore = 1.5 m (2.1 mm) → F0=0.49, si **accetta H0**, nessuna differenza significativa.

M.M.

Altezza Kinect 1.0 m:
Total Body = 1.0 m (5.3 mm) → F0=0.36, si **accetta H0**, nessuna differenza significativa.
Arto superiore = 1.5 m (7.4 mm) → F0=0.31, si **accetta H0**, nessuna differenza significativa.
Tronco = 1.0 m (4.3 mm) → F0=2.57, si **accetta H0**, nessuna differenza significativa.
Arto inferiore = 1.0 m (1.2 mm) → F0=3.33, si **accetta H0**, nessuna differenza significativa.
Altezza Kinect 1.5 m:
Total Body = 2.0 m (11.2 mm) → F0=0.07, si **accetta H0**, nessuna differenza significativa.
Arto superiore = 1.5 m (12.7 mm) → F0=0.01, si **accetta H0**, nessuna differenza significativa.
Tronco = 1.0 m (11.1 mm) → F0=0.46, si **accetta H0**, nessuna differenza significativa.
Arto inferiore = 2.0 m (7.0 mm) → F0=0.34, si **accetta H0**, nessuna differenza significativa.
Altezza Kinect 2.0 m:
Total Body = 1.0 m (10.7 mm) → F0=2.27, si **accetta H0**, nessuna differenza significativa.
Arto superiore = 1.0 m (7.8 mm) → F0=6.44, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

Tronco = 1.00 m (7.2 mm) → F0=10.8, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.
Arto inferiore = 1.50 m (11.6 mm) → F0=0.26, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 1.0 m:
Total Body = 1.0 m (5.3 mm) → F0=2.93, si **accetta H0**, nessuna differenza significativa.
Arto superiore = 2.0 m (7.8 mm) → F0=1.10, si **accetta H0**, nessuna differenza significativa.
Tronco = 1.00 m (4.3 mm) → F0=7.15, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-2.0 m.
Arto inferiore = 1.00 m (1.2 mm) → F0=3.78, si **accetta H0**, nessuna differenza significativa.
Distanza Utente 1.5 m:
Total Body = 1.0 m (6.4 mm) → F0=3.69, si **accetta H0**, nessuna differenza significativa.
Arto superiore = 1.0 m (7.5 mm) → F0=4.81, si **rigetta H0**, nessuna differenza significativa limitata alle misure 1.0 e 1.5 m.
Tronco = 1.0 m (4.6 mm) → F0=12.2, si **rigetta H0**, significativamente differente dalle altre altezze.
Arto inferiore = 1.0 m (5.7 mm) → F0=0.47, si **accetta H0**, nessuna differenza significativa.
Distanza Utente 2.0 m:
Total Body = 1.0 m (5.9 mm) → F0=19.8, si **rigetta H0**, significativamente differente dalle altre altezze.
Arto superiore = 1.0 m (8.8 mm) → F0=12.3, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-2.0 m.
Tronco = 1.0 m (6.1 mm) → F0=6.49, si **accetta H0**, nessuna differenza significativa.
Arto inferiore = 1.0 m (1.5 mm) → F0=7.09, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

Distanza Utente 2.5 m:

Total Body = 1.0 m (8.5 mm) → F0=10.4, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

Arto superiore = 1.0 m (9.4 mm) → F0=7.83, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

Tronco = 1.0 m (11.7 mm) → F0=12.8, si **rigetta H0**, significativamente differente dalle altre altezze.

Arto inferiore = 1.0 m (4.5 mm) → F0=6.07, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

A.S.

Altezza Kinect 1.0 m:

Total Body = 2.5 m (3.4 mm) → F0=25.2, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.0-2.5 e 1.5-2.5 m.

Arto superiore = 2.5 m (4.5 mm) → F0=46.0, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.0-2.5 e 1.5-2.5 m.

Tronco = 2.5 m (3.0 mm) → F0=38.8, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-2.5 m.

Arto inferiore = 1.0 m (1.7 mm) → F0=2.97, si **accetta H0**, nessuna differenza significativa.

Altezza Kinect 1.5 m:

Total Body = 2.5 m (2.5 mm) → F0=10.2, si **rigetta H0**, significativamente differente dalle altre distanze.

Arto superiore = 2.5 m (3.1 mm) → F0=11.2, si **rigetta H0**, significativamente differente dalle altre distanze.

Tronco = 2.5 m (1.8 mm) → F0=15.4, si **rigetta H0**, significativamente differente dalle altre distanze.

Arto inferiore = 2.5 m (1.8 mm) → F0=0.82, si **accetta H0**, nessuna differenza significativa.

Altezza Kinect 2.0 m:

Total Body = 1.5 m (4.7 mm) → F0=12.0, si **rigetta H0**, nessuna differenza significativa limitata coppia 1.5-2.5 m.

Arto superiore = 2.5 m (6.7 mm) → F0=8.6, si **rigetta H0**, nessuna differenza significativa limitata alle coppia 1.5-2.5 m.

Tronco = 1.5 m (3.2 mm) → F0=14.6, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.5-2.5 m.

Arto inferiore = 1.5 m (2.3 mm) → F0=5.32, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 1.5-2.0 e 1.5-2.5 m.

Distanza Utente 1.0 m:

Total Body = 1.0 m (3.6 mm) → F0=16.9, si **rigetta H0**, significativamente differente dalle altre altezze.

Arto superiore = 1.0 m (4.6 mm) → F0=11.9, si **rigetta H0**, significativamente differente dalle altre altezze.

Tronco = 1.0 m (4.4 mm) → F0=5.37, si **accetta H0**, nessuna differenza significativa.

Arto inferiore = 1.0 m (1.7 mm) → F0=4.69, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 1.5 m:

Total Body = 1.0 m (4.4 mm) → F0=2.62, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.0 m (5.3 mm) → F0=2.25, si **accetta H0**, nessuna differenza significativa.

Tronco = 2.0 m (3.7 mm) → F0=2.89, si **accetta H0**, nessuna differenza significativa.

Arto inferiore = 1.0 m (2.1 mm) → F0=1.27, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 2.0 m:

Total Body = 1.5 m (7.1 mm) → F0=3.78, si **accetta H0**, nessuna differenza significativa.

Arto superiore = 1.5 m (9.3 mm) → F0=5.68, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.
Tronco = 1.5 m (8.2 mm) → F0=4.87, si **accetta H0**, nessuna differenza significativa.
Arto inferiore = 1.5 m (3.2 mm) → F0=0.86, si **accetta H0**, nessuna differenza significativa.

Distanza Utente 2.5 m:

Total Body = 1.5 m (2.5 mm) → F0=15.9, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.
Arto superiore = 1.5 m (3.1 mm) → F0=5.58, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.
Tronco = 1.5 m (1.8 mm) → F0=27.3, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.
Arto inferiore = 1.5 m (1.8 mm) → F0=10.5, si **rigetta H0**, nessuna differenza significativa limitata alla coppia 1.0-1.5 m.

Il codice MATLAB per ricavare queste misure oltre che i plot del trend di ogni Joint è mostrato in Appendice C.

La cosa più interessante che si evince da questi dati è che effettivamente a parità di altezza del Kinect non è possibile dire con certezza quale misura sia la migliore in assoluto. Per ogni soggetto infatti le misure a precisione migliore sono completamente diverse sia considerando tutti i 25 Joint che i gruppi separati.
Riassumendo: per G.S., analizzando separatamente braccia, tronco e gambe, i risultati portano per il 55% delle prove (5/9) non c'è un effetto significativo dell'altezza sulla precisione statica. Per M.M. la percentuale sale al 77% mentre per A.S. è solo del 22%. Tuttavia in questa prova le acquisizioni fatte a 2.0 m. mostrano sempre deviazioni standard di molto superiori alle altre, per un motivo purtroppo sconosciuto. Grazie ai t-test si vede bene che c'è spesso somiglianza statistica con le altre misure e probabilmente con valori migliori delle

prove con il Kinect a quota 2 metri si sarebbe potuto avere una percentuale più alta. Si è deciso di lasciarla comunque in quanto la prova è stata effettuata con gli stessi accorgimenti di tutte le altre, per cui se l'errore è imputabile al caso si ritiene giusto considerarlo anche come monito per eventuali studi futuri. Le percentuali invece, che mostrano una significativa diversità statistica della misura più precisa da tutte le altre, sono piuttosto basse: l'11% per G.S., 0% per M.M. ed il 22% per A.S. Nel confronto tra misure a stessa distanza e diversa altezza del sensore si può notare invece che si ha un peggioramento generale all'aumentare dell'altezza, con l'aggravante dell'aumento progressivo del numero di falsi positivi (Joint considerati traccati anche se non erano visibili) e falsi negativi (joint visibili al Kinect ma che son stati considerati non visibili per più Frames). Questi sono stati valutati grazie ad un confronto tra lo status dei Joint riportato dal Kinect e la visualizzazione, tramite telecamera RGB, di ciò che il Kinect potesse realmente visualizzare. Sostanzialmente anche qui è possibile parlare di percentuali mostrando che per G.S. le prove in cui è possibile accettare H0 sono il 58% (7/12) mentre le prove in cui la misura più precisa è risultata essere statisticamente diversa da tutte le altre sono il 25%; per M.M. sono rispettivamente 33% e 16%, mentre per A.S. sono rispettivamente 58% e 8%. Come è possibile intuire la somma delel percentuali non fa 100% in quanto sono escluse quelle prove in cui la misura più precisa è risultata essere statisiticamente simile ad almeno un'altra misura di distanza o di altezza. Si può così notare come, nelle prove a distanza fissa, si abbia prevalentemente la vincita di misure ad 1 metro e ad 1 metro e mezzo di altezza del Kinect. Nelle prove ad altezza fissa non è invece possibile delineare un così chiaro risultato. Se ne evince che, mentre è preferibile tenere il Kinect ad una altezza bassa (seppure alzarlo nella maggior parte dei casi non comporta risultati statisticamente diversi con significatività dell'1%), nelle registrazioni fatte tra il metro e i due metri e mezzo di distanza

dell'utente, la posizione non comporta significative differenze nella precisione del motion tracking.

Effettivamente guardando la totalità dei risultati, le prove in cui si rifiuta H0 sono comunque molte, tuttavia per stillare una vera e propria statistica sarebbe necessario fare decine e decine di prove di questo tipo, cosa che praticamente richiederebbe una tesi a sé. Per questo si è deciso di accontentarsi dei risultati ottenuti visto e considerato che, sia che si accetti H0, sia nei casi in cui si rifiuta (ipotesi di prove statisticamente differenti), i test statistici dimostrano come sia impossibile definire una posizione ottimale o almeno un trend tra collocamento del Kinect e dell' utente.

Per analizzare meglio i valori a diverse distanze si è deciso di eseguire altre 2 registrazioni con M.M. ed A.S. con altezza fissata a 1 m e 2 m. Questa volta però, di durata di 10 secondi e con spostamento di 20 cm, partendo da 1 metro di distanza dal Kinect fino a 3 metri. La posa tenuta dai soggetti è rimasta la stessa. I risultati non mostrano differenze sostanziali con quanto già detto, se non un leggero aumento delle percentuali per le prove che risultano accettare H0. Anche in queste prove è risultato evidente come non ci sia nessun trend tra le deviazioni standard e la distanza come è possibile vedere in figura:

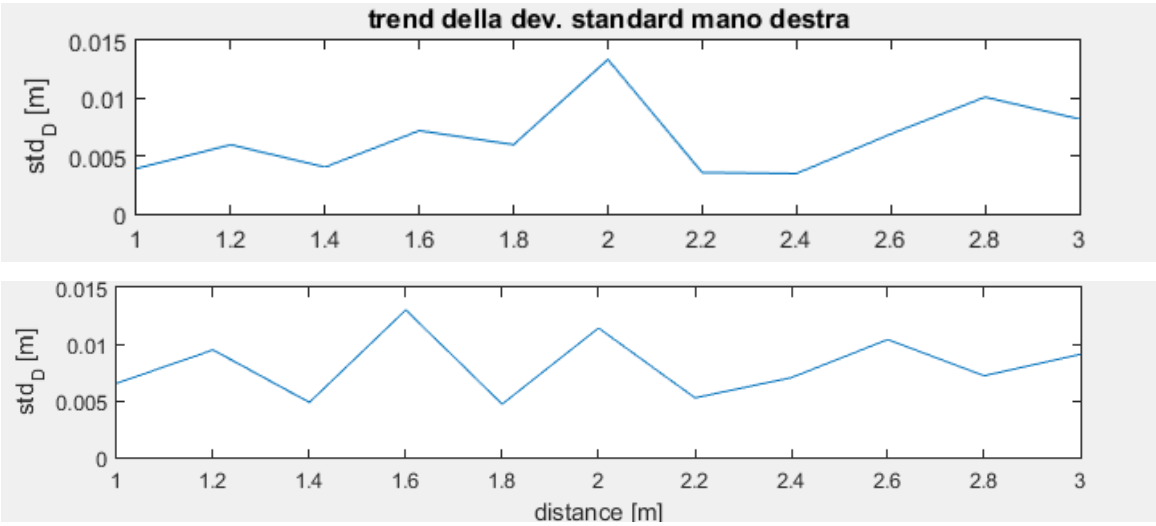


Fig. 15 – Trend della deviazione standard del Joint mano destra per 11 distanze con Kinect posto ad altezza 1 metro (in alto) e 2 metri (in basso) per A.S.

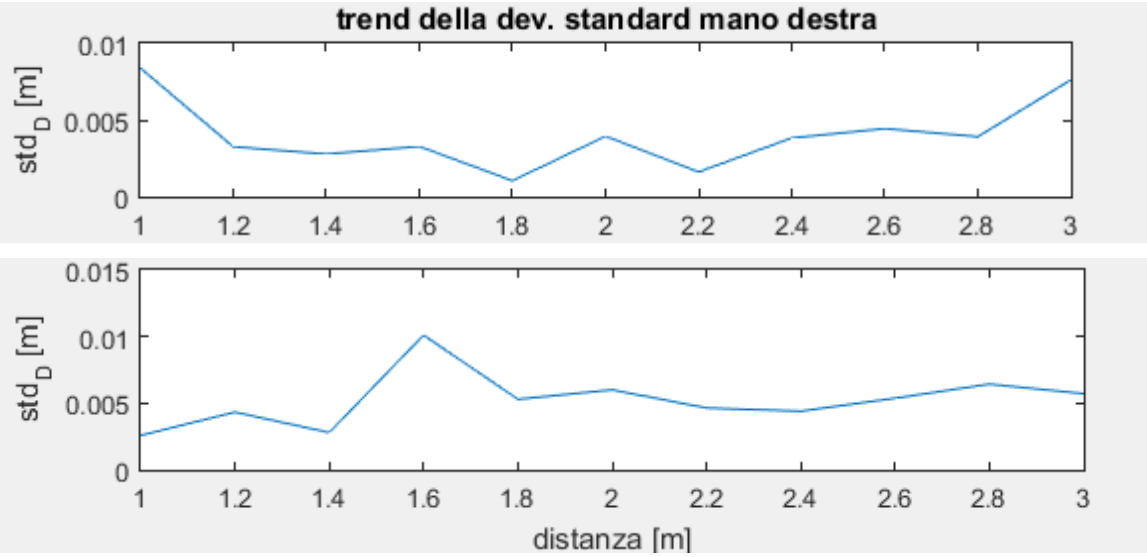


Fig. 16 – Trend della deviazione standard del Joint mano destra per 11 distanze con Kinect posto ad altezza 1 metro (in alto) e 2 metri (in basso) per G.S.

Un' interessante osservazione che è possibile fare infine è che valutare le differenze statistiche tra tutti i 25 joint piuttosto che dividere il corpo in settori, porta ad un maggior numero di esperimenti statisticamente non differenti. D'altro canto si può anche osservare che nella divisione in 3 gruppi dei Joint, il gruppo denominato Body e contenente i Joint di spina dorsale collo e testa, è quello che mostra meno prove statisticamente simili, separando maggiormente la situazione con maggiore precisione dalle altre.
Altre osservazioni a riguardo di questo esperimento sono rimandate al capitolo 7.

Analisi per diverse condizioni di luce

Dalla letteratura e soprattutto dall'esperienza pratica è risultato che la luce, in particolare la luce solare possa influenzare in maniera negativa il riconoscimento della figura umana da parte del Kinect. È capitato

molto spesso ad esempio che, nei casi in cui l'utente sia stato posizionato a fianco di una finestra da cui entrasse luce solare, il Kinect abbia erroneamente riconosciuto in questa un altro Body tracciando a schermo un secondo scheletro del tutto fasullo. In altri casi è capitato che i riflessi di luce su oggetti alle spalle dell'utente modificassero il tracking del Body in alcune parti del corpo deformandone alcune sezioni. È stata così presa in considerazione l'idea di realizzare un altro esperimento per valutare l'influenza dell'illuminazione ambientale. Studi in letteratura **[26]** confermano che gli errori della telecamera di profondità, che grazie alla tecnologia TOF può essere utilizzata anche in ambiente outdoor, aumentano sensibilmente spostandosi da ambiente indoor con solo luce artificiale ad ambiente con anche luce solare. È riportato inoltre che in caso di luce solare diretta nel campo visivo del sensore i dati perdono totalmente di validità con un 70% di affidabilità ad un metro di distanza e lo 0% oltre i due metri e mezzo.
Così l'ultimo confronto è stato fatto mantenendo una distanza della persona e l'altezza del sensore simbolicamente a 1.5 m facendo variare l'illuminazione della stanza. L'illuminamento (intensità luminosa per unità di superficie) è stata misurato in lux con uno smartphone tramite l'app Light Meter di Borce Trajkovski (errore massimo dichiarato del 5%). Si è deciso di iniziare con una condizione di stanza ben illuminata con 2 lampadine a basso consumo da 20 Watt e luce diurna entrante da una porta finestra alle spalle del Kinect, per un totale di 23 lux. Si è quindi chiuso lo scuro della porta finestra rimanendo con 10 lux di illuminamento che è sceso a 5 allo spegnersi della prima lampadina e 0 nel buio completo. In queste quattro registrazioni (soggetto G.S.), è stata utilizzata la solita posa per 30 secondi. Al termine è stata eseguita un'altra registrazione riaccendendo le lampadine (10 lux), per scongiurare l'ipotesi di un miglioramento del tracking dovuto alla sola ripetizione e non alla condizione di luce. Sono quindi riportati i risultati nuovamente analizzati con ANOVA e t-test a coppie.

Condizione di luce:

Total Body = 23 lux (3.3 mm) → F0=14.19, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 0-23, 0-5 lux.
Arto superiore = 23 lux (4.7 mm) → F0=16.0, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 0-23, 0-5 lux.
Tronco = 0 lux (2.7 mm) → F0=42.7, si **rigetta H0**, nessuna differenza significativa limitata alle coppie 0-23, 0-5 lux.
Arto inferiore = 23 lux (1.4 mm) → F0=2.64, si **accetta H0**, nessuna differenza significativa.

Come nelle prove precedenti, anche cambiando l'illuminazione all'interno di un ambiente chiuso non è stato individuato un trend per descrivere un miglioramento da una condizione di luce ad un'altra. Anzi in queste prove paradossalmente le misure con deviazione standard minore sono state le due estreme: buio completo e luce diffusa. Anche provando in un laboratorio con luci al neon ed illuminazione misurata attorno ai 100 lux non è stata registrata una differenza sostanziale nell'entità delle deviazioni standard. Per cui l' unica indicazione che è possibile dare a questo riguardo è di evitare che in qualunque modo arrivino al Kinect dei raggi IR o solari indesiderati.

Capitolo 5
Il filtraggio

Nei capitoli precedenti è stato introdotto il problema del Jittering nel tracking da parte del Kinect. È stata analizzata inoltre la deviazione standard dei dati, che abbiamo visto aggirarsi, nella maggior parte dei casi, attorno ai 5 mm. Dai risultati ottenuti è stato possibile vedere come questa sia scarsamente dipendente dalla distanza dal Kinect, fino a 3 metri dal sensore, ma che nei casi peggiori è potuta arrivare in media anche nell'ordine dei centimetri. A questo punto la necessità di utilizzare il filtraggio viene, non solo dalla volontà di rimuovere i Jitter, ma anche da quella di diminuire il rumore dei dati. La Microsoft stessa per il Kinect per 360 ha realizzato un filtro attivabile a piacimento settando dei parametri in una struttura denominata `TrasformSmoothParameters` e richiamando il metodo `SkeletonStream.Enable(Param)` **[27]**. Questa possibilità è stata tolta nel Kinect One essendo progettato per lavorare anche con più piattaforme allo stesso tempo **[29]**. Per cui dopo numerose richieste è stata rilasciato il codice di una classe, in C++ ed in C# che realizzasse il medesimo filtraggio lasciando però più libertà all'utilizzatore nel richiamarla **[30]**. In realtà in questo modo si rende possibile anche la modifica o l'estensione della stessa per soddisfare qualunque esigenza. In fondo applicazioni diverse o anche Joint diversi possono richiedere tecniche di filtraggio diverse ed in questo modo è possibile fornirgliele creando tanti oggetti istanziati alla classe per quanti sono gli utilizzi necessari.

La classe che implementa il filtro presente nel Kinect V1 è riproposta in Appendice D.

Riguardo all'argomento del filtraggio, la Microsoft è ben conscia del problema della precisione del Kinect, tanto che fornisce lei stessa alcune linee guida su come operare **[28]**. La compagnia afferma che il sensore ha un'ottima accuratezza per quanto riguarda l' algoritmo di tracking e che la media della posizione del Joint nel tempo è praticamente sempre corrispondente alla posizione reale dell'articolazione o della parte anatomica corrispondente. La presenza del rumore tuttavia limita questa accuratezza all'ordine dei centimetri in quanto la posizione subisce variazioni ad ogni frame. Come è stato già detto si possono distinguere due tipi di rumore; un leggero rumore bianco di entità nell'ordine del millimetro, inevitabilmente presente in ogni sensore, ed un rumore composto da picchi o gradini nel segnale. Sulle cause che producono questo effetto indesiderato, la compagnia americana sostiene che siano originati dall'algoritmo di tracking quando il Joint è nascosto alla vista della telecamera di profondità. Se il sistema non ha abbastanza informazioni in uno specifico frame per determinare la posizione esatta del Joint, vuoi perché fuori dalla scena o perché occluso da un oggetto, da una persona o da un' altra parte del corpo dell' utente stesso, è comunque in grado di cercare di dedurla. Il Kinect per questo motivo fornisce ad ogni frame lo stato di ogni Joint come Tracked o Inferred. Secondo Microsoft quando lo stato è dedotto (Inferred) possono presentarsi questi picchi di spostamento del Joint anche di qualche centimetro. Sulla base dell'esperienza fatta tuttavia le cose non sono proprio così. Non è stato raro infatti trovare dei casi in cui analizzando delle registrazioni è stato possibile individuare dei Jitter ed andando poi ad analizzare lo stato del Tracking per il frame ed il Joint corrispondente, questo è risultato essere Tracked. Viceversa è stato possibile notare che in caso di Joint Inferred l'algoritmo si è comportato egregiamente mantenendo la posizione in maniera stabile nel punto corretto. Inoltre non sempre si sono mostrati solo dei picchi, ma anche dei gradini in cui la posizione del Joint dopo essere saltata via è poi rimasta nella nuova posizione.

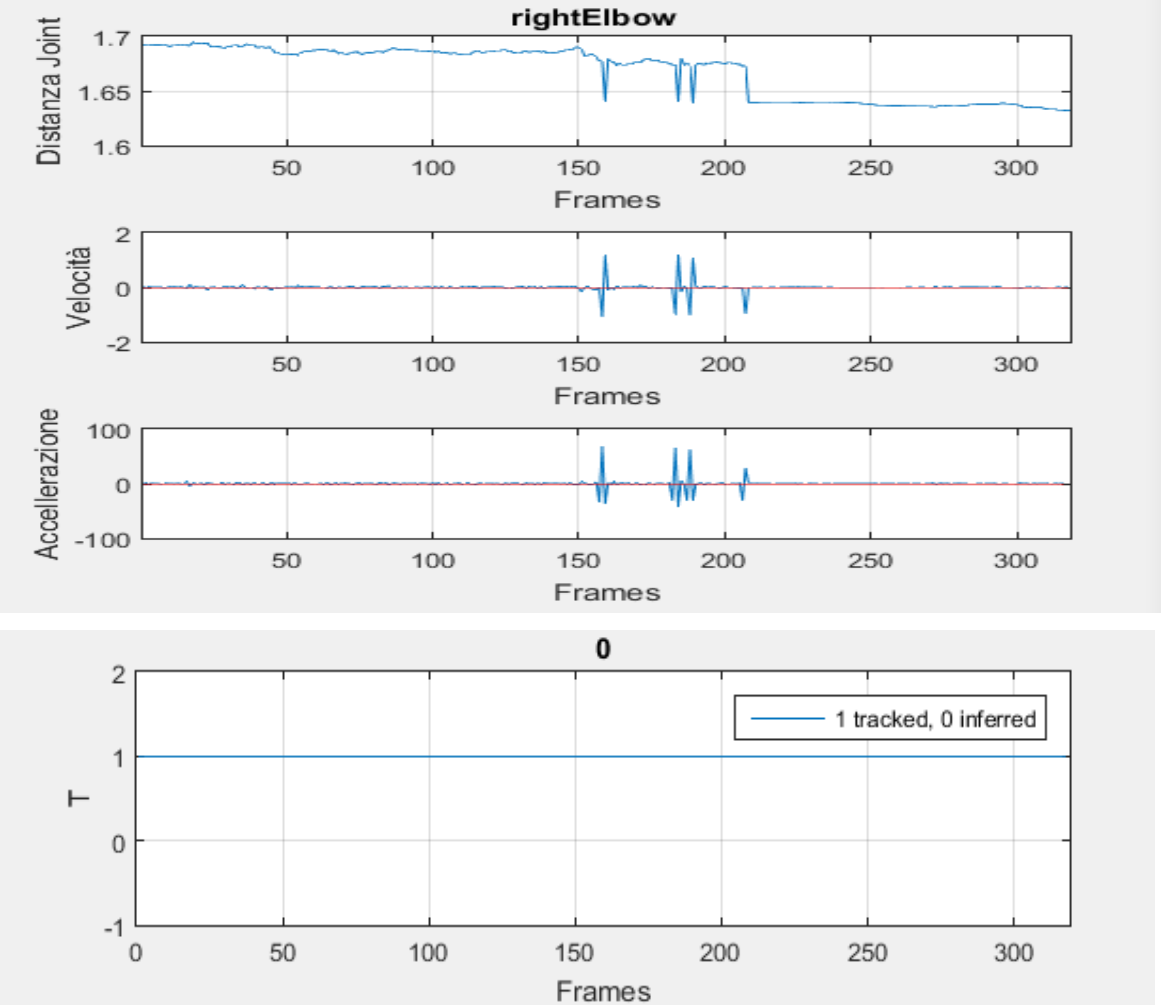


Fig. 17 – in figura è mostrata una registrazione di una posa statica in cui è possibile osservare 4 eventi classificabili come dei Jitter. I primi tre, ai frame 160, 184 e 189 sono visibilmente dei picchi di circa 5 cm, l' ultimo al frame 208 è un gradino di circa la stessa ampiezza. Questo nonostante il Joint sia stato considerato perfettamente traccato dal sensore (0 volte Inferred).

Tornando al filtraggio, esistono numerose tecniche applicabili, naturalmente per ottenere effetti diversi, ma non è raro, anzi è consigliato, dover combinare più tecniche diverse per ottenere i risultati migliori. Un opportuno filtraggio porta grossissimi benefici, ma di contro introduce un fastidioso problema, la latenza. Il problema della latenza nel filtraggio dei Joint rappresenta il tempo necessario all'uscita del filtro per raggiungere la posizione effettiva del Joint a seguito di un movimento. Rappresenta in sostanza un ritardo del segnale filtrato

rispetto a quello non filtrato, ritardo non dovuto al tempo di calcolo della CPU, ma ad una distorsione di fase. In generale l'entità di questo ritardo dipende da quanto velocemente sta cambiando il segnale originale **[28]**. Fanno eccezione i filtri a fase lineare che mostrano indifferentemente lo stesso ritardo per qualunque frequenza di ingresso. È auspicabile che nelle applicazioni in cui si utilizza il filtraggio in tempo reale, la latenza non superi i 100 ms per non risultare fastidiosa **[28]**.
Il metodo di filtraggio implementato nella classe proposta da Microsoft è di fatto composto da un filtro doppio esponenziale, da un Jitter removal ed da un controllore.
Innanzitutto viene applicato un Jitter removal, ovvero un filtro che limita gli spostamenti permessi all'interno di una sfera di raggio definito dall'utente. Chiamando ρ questo raggio, se lo spostamento del Joint al passo n è inferiore al valore di ρ si lascia il valore originale in uscita, altrimenti viene ridotto. In questo caso si riduce con un metodo esponenziale.

$$\widehat{X_n}' = \begin{cases} X_n, & se\ |X_n - \widehat{X_{n-1}}| < \rho \\ \left(\frac{\rho}{d}\right) X_n + (1 - \left(\frac{\rho}{d}\right))\widehat{X_{n-1}}, & altrove \end{cases}$$

Con:

$$d = |X_n - \widehat{X_{n-1}}|$$

L' apostrofo è utilizzato per indicare l'uscita del Jitter removal. La stima del valore filtrato in uscita dal doppio esponenziale avrà il doppio apostrofo mentre l'uscita finale del processo non avrà apostrofi ma solo il cappellino. Il valore grezzo è X_n. Naturalmente stiamo parlando di vettori contenenti la x, la y e la z del joint. Il filtraggio avviene separatamente per le tre componenti.

Poiché all'aumentare di d il rapporto $\frac{\rho}{d}$ diminuisce, tenderà a pesare di più il valore del Joint al frame precedente limitando così il Jitter. P è fornito dall'utente come parametro in metri. In seguito è applicato uno smoothing doppio esponenziale. Questa tecnica ha il vantaggio che se il segnale è caratterizzato da un trend, il filtraggio realizza un eccellente smoothing introducendo meno latenza di qualunque altro filtro digitale **[28]**. L'eliminazione dei jitter è fatta prima proprio per rendere maggiormente lineari i segnali ed evidenziare la presenza di un trend. In sostanza si tratta di un filtro a media mobile pesata con coefficienti decrescenti in maniera esponenziale con la differenza che la media mobile utilizza sempre k campioni e bisogna mantenere in memoria k campioni ad ogni passo, con l'esponenziale vengono utilizzati tutti i campioni passati (tramite un algoritmo ricorsivo), ma è sufficiente mantenere in memoria l'ultimo valore stimato. La formula generale per lo smoothing esponenziale è:

$$s_0 = x_0$$
$$s_t = \alpha x_t + (1-\alpha)s_{t-1},\ t > 0$$

Con α fattore di smoothing che va da 0 ad 1, X_t è il valore grezzo non filtrato mentre s è il valore stimato. I pedici naturalmente rappresentano i passi dell'algoritmo, per cui s_0 è il valore stimato al passo 0 (coincide con il valore grezzo) e s_t è il valore stimato al passo t. Come già detto per iniziare l'algoritmo è necessario almeno un campione per cui il primo valore realmente stimato è s_1. In s_2 si utilizza s_1 per il quale si è utilizzato s_0 e così via seguendo un procedimento ricorsivo fino a s_t.

Il filtro è chiamato "doppio esponenziale" in quanto dopo il primo filtraggio di tipo esponenziale l'uscita è utilizzata per un altro filtraggio esponenziale per trovare il trend. Questo aiuta a ridurre la latenza in quanto il filtro considera i dati in input come se fossero disposti in

maniera lineare, con pendenza data appunto dal valore del trend. Il valore in uscita al filtro è quindi valutato come predizione di valore futuro dalla somma pesata del trend e del valore uscito dal primo filtraggio esponenziale. In sostanza al passo n il filtro immagina che ci sia un trend lineare per i valori dei dati e seguendo questo trend valuta quale sarebbe l'uscita dopo un certo numero di campioni nel futuro. In uscita al filtro al passo n viene data la stima del valore dell'output al passo n+ γ, dove γ è il numero di frame da predire nel futuro. Anche qui l'utente può inserire 3 parametri: α, β e γ per ottenere i risultati più consoni alle proprie esigenze. Di fatto si tratta di un filtro ARMA. L'equazione del filtro doppio esponenziale è usata nella formulazione di Holt **[28]**.

Smoothed output: $\widehat{S_n} = \alpha X_n + (1-\alpha)\left(\widehat{X_{n-1}} + b_{n-1}\right)$

Trend: $b_n = \beta(\widehat{S_n} - \widehat{X_{n-1}}) + (1-\beta)\, b_{n-1}$

Forecasting: $\widehat{X_{n+\gamma|n}}'' = \widehat{S_n} + \gamma b_n$

$\alpha : 0 \leq \alpha < 1$ → parametro di smoothing. incrementando questo valore si realizza uno smoothing più pesante, ma aumenta di conseguenza la latenza.

$\beta : \ 0 \leq \beta < 1$ → parametro di smoothing del trend. Un valore grande corregge maggiormente i dati pesando meno il trend. Aiuta anche a ridurre la latenza.

$\gamma \ : \ \gamma \geq 0$ → parametro di forecasting. Rappresenta il numero di frames da predire nel futuro. Aumentare il valore diminuisce la latenza, ma aumenta anche l'overshooting in caso di cambi bruschi di direzione del segnale.

Per limitare effetti indesiderati come l'overshooting del segnale in presenza di movimenti bruschi è stato inserito anche un controllore

finale che, valutando la differenza tre il valore uscito dal doppio esponenziale e il valore corrispettivo del segnale grezzo, ne limita la distanza in maniera analoga al jitter removal.

$$\widehat{X_n} = \begin{cases} \widehat{X_n''}, & altrove \\ \left(\frac{\delta}{d}\right)\widehat{X_n''} + \left(1 - \left(\frac{\delta}{d}\right)\right) X_n, & se\ |X_n - \widehat{X_{n-1}}| < \delta \end{cases}$$

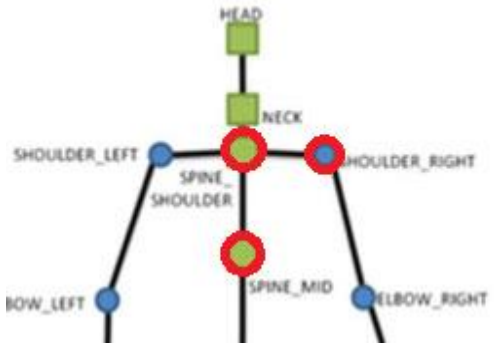
Anche in questo caso tramite un parametro (δ) viene settato il raggio di una sfera centrata sul valore grezzo della posizione del Joint e se la posizione filtrata esce da questa sfera la si riavvicina con l'equazione dello smoothing esponenziale pesando maggiormente il valore grezzo. Niente comunque vieta di utilizzare altre tipologie di filtri anche se è consigliabile rimanere sugli ARMA per questioni di performance.
In questo progetto si è scelto di utilizzare questa classe in quanto ha dimostrato di comportarsi egregiamente ed è stato apprezzato molto il grado di personalizzazione dato dal settaggio a piacere dei 5 parametri. In aggiunta per i Joint che nelle prove a seguire hanno dovuto fungere da riferimento, si è scelto di utilizzare la media cumulativa per ancorarne la posizione e modificarla solo in caso di effettivo spostamento.

$$\widehat{X_n} = \begin{cases} X_n, & se\ \left|X_n - \widehat{X_{n-1}}\right| > 0.15\ m \\ \left(\frac{n}{n+1}\right)\widehat{X_{n-1}} + \left(\frac{1}{n+1}\right) X_n, & altrove \end{cases}$$

$$n = \begin{cases} 1, & se\ \left|X_n - \widehat{X_{n-1}}\right| < \delta \\ n+1, & altrove \end{cases}$$

Valutazione qualitativa sul filtraggio

Per lo scopo è stato realizzato un piccolo gioco in C# in cui seguire con la mano destra o la mano sinistra un cerchio disegnato sullo schermo. Seguendo i consigli forniti da Microsoft **[14]** si è scelto di utilizzare la spalla come Joint di riferimento per il movimento della mano traslando e ruotando il sistema di riferimento. Viene così creata una PHIZ (PHysical Interaction Zone) corrispondente allo spazio di movimento da utilizzare. In questa prova si è deciso di lavorare con un'area rettangolare centrata sul Joint spalla e di dimensioni 80x45 m. Per ottenere la trasformazione delle coordinate del Joint Hand dal sistema globale (Kinect) a quelle locali (spalla) si è calcolata una matrice di rotazione 3x3 utilizzando una nuova terna ortogonale originata dai vettori di posizione di Shoulder, Spine-Shoulder e Mid-Spine. Definendo questi S, Ss e Ms, ed introducendo i nuovi versori i, j e k, le direzioni della nuova terna ortogonale sono ottenute come:



$$\hat{\imath} = (S^G - Ss^G)/|S^G - Ss^G|$$

$$\hat{k} = (Ss^G - Ms^G) \times \hat{\imath}\ /|(Ss^G - Ms^G) \times \hat{\imath}|$$

$$\hat{\jmath} = \hat{k} \times \hat{\imath}$$

Fig. 18 – i Joint Shoulder (right o left), Spine-Shoulder e Spine_mid sono utilizzati per creare un nuovo sistema ortogonale con origine su Shoulder.

Il nuovo asse x si è fatto coincidere quindi con la direzione del segmento spalla-centro_spalla uscente dalla spalla, il prodotto vettoriale tra la direzione del segmento Mid-Spine, Spine-Shoulder e

l'asse x appena calcolato è diventato il nuovo asse z ed infine il prodotto vettoriale tra versore di x e versore di z ha originato il versore dell'asse y. Naturalmente i calcoli sono i medesimi se si vuole utilizzare la parte sinistra del corpo, con solo un nuovo significato per il vettore S. La spalla destra per la mano destra o la spalla sinistra per la mano sinistra sono state "ancorate" nella loro posizione utilizzando la media cumulativa, mentre per la mano corrispondente si è utilizzata la classe con lo smoothing doppio esponenziale.
Si è quindi traslato il sistema di riferimento sulla spalla con la semplice sottrazione vettoriale:

$NP^G = H^G - S^G$ *Vettore mano rispetto alla spalla*

Dove NP^G significa New Point nelle coordinate del sistema di riferimento globale, S è il vettore spalla ed H il vettore mano, come già detto possono essere congiuntamente o destro o sinistro.
Tramite i nuovi versori è stato calcolato il vettore New Point in coordinate locali con origine sul Joint spalla:

$^G R_l = [\,\hat{\imath}\ \hat{\jmath}\ \hat{k}\,]$ *Rotazione da locale a globale*

$^l R_G = {}^G R_l^t$ *Rotazione da globale a locale*

$NP^l = {}^l R_G\, NP^G$ *posizione mano in coordinate locali*

Operando in questo modo si è voluto rendere possibile giocare con lo schermo non necessariamente davanti al Kinect.

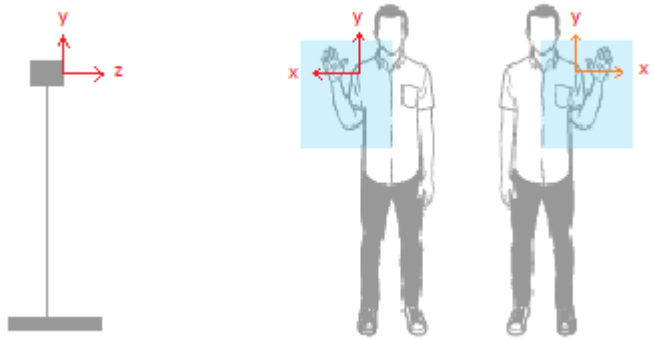


Fig. 19 – La trasformazione del sistema di riferimento da globale a locale permette di effettuare l'esercizio anche se non frontalmente al Kinect. i movimenti che effettua in y e z nelle coordinate globali diventano movimenti in x ed y nelle nuove coordinate locali

La mano è stata raffigurata a schermo come un pallino verde localizzata nella posizione corrispondente alla PHIZ creata in precedenza.

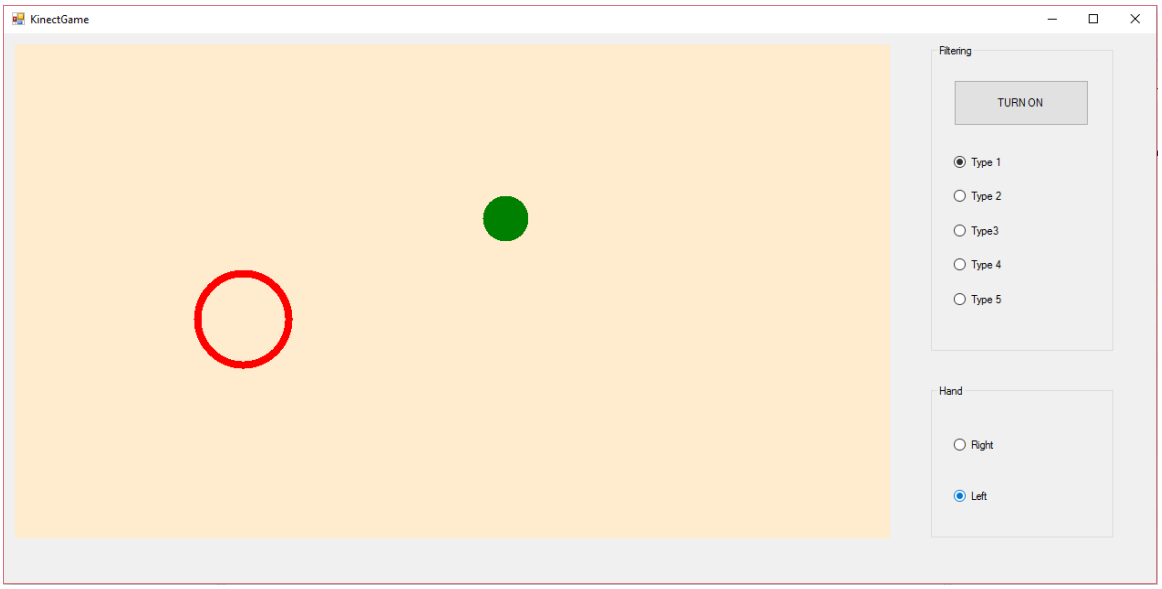


Fig. 20 – Panoramica del Form in cui è stato realizzato il gioco. A destra è possibile selezionare i filtri e quale mano usare. Il movimento della mano è tradotto nello spostamento del tondo verde.

Si è deciso di far cominciare il gioco nel momento in cui il pallino verde comandato dalla mano si trovasse completamente all'interno del cerchio rosso. A questo punto si è deciso di far muovere il cerchio

rosso secondo una funzione matematica sinusoidale per ottenere una figura ad otto rovesciato.

Il gioco è stato implementato in un Windows Form simile a quello descritto nel capitolo 3, il cui codice è visualizzabile in Appendice A. Sono stati tolti gli elementi non utili e la registrazione delle posizioni è stata fatta coincidere con la durata dell' esercizio.
Si è deciso quindi di assegnare un sistema di punteggi che valutasse con quanta fedeltà il giocatore fosse riuscito a seguire il cerchio rosso. In particolare si è fatto muovere l'oggetto con passo temporale di 33 ms dando ad ogni frame 100 punti se perfettamente dentro al cerchio, 50 se vengono toccati i bordi e 0 se si fosse stati completamente fuori. Dopo 30 secondi il gioco termina e viene mostrato il punteggio in percentuale. Si è quindi deciso di effettuare il gioco attivando il filtraggio con diversi set di parametri salvando le posizioni grezze del joint mano e le posizioni filtrate.

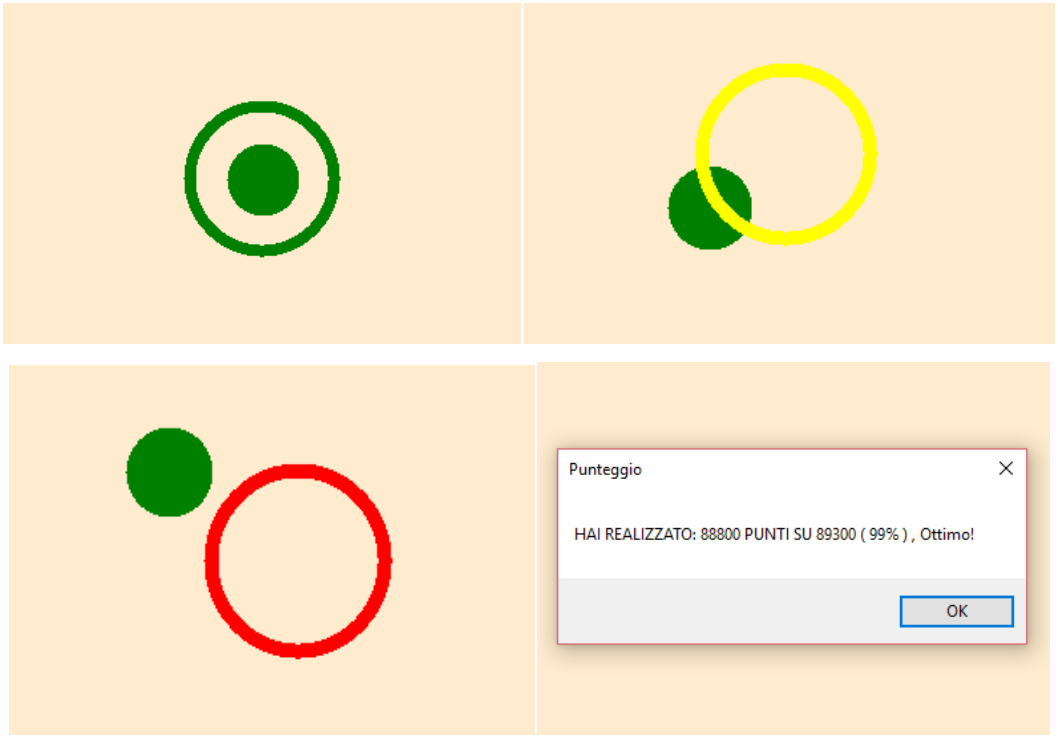
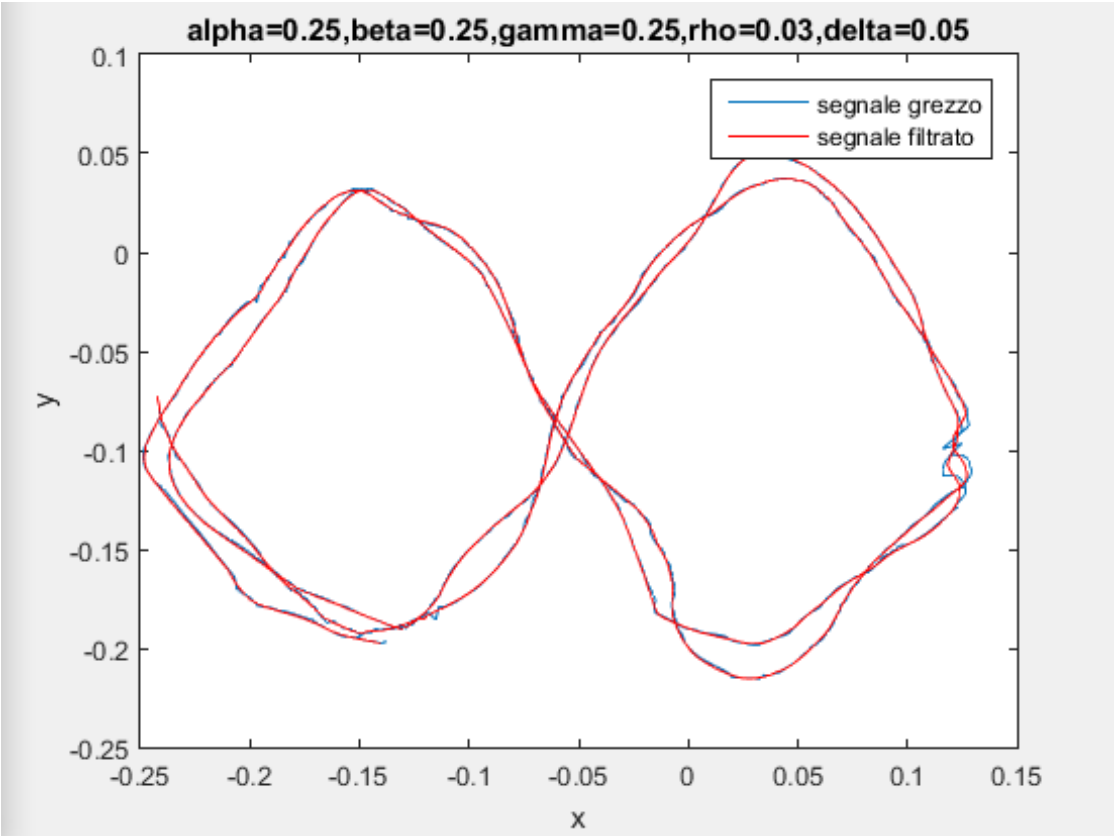
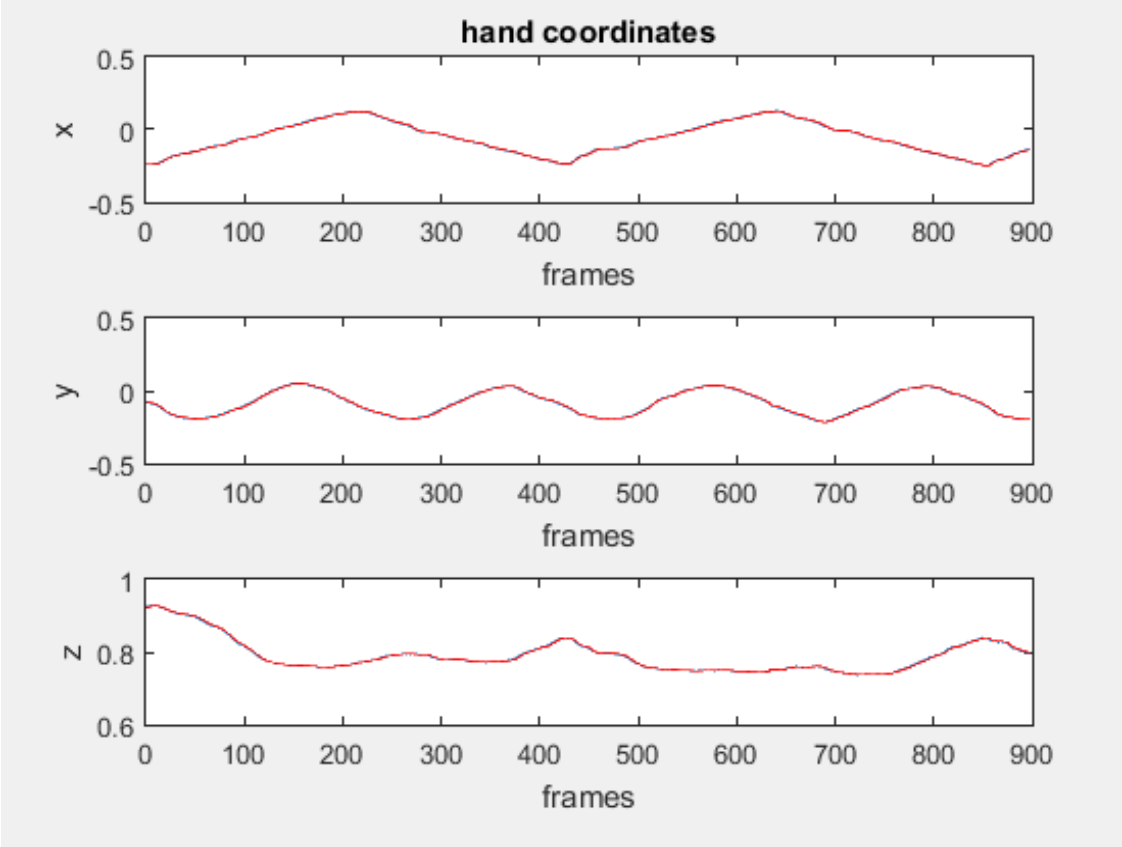


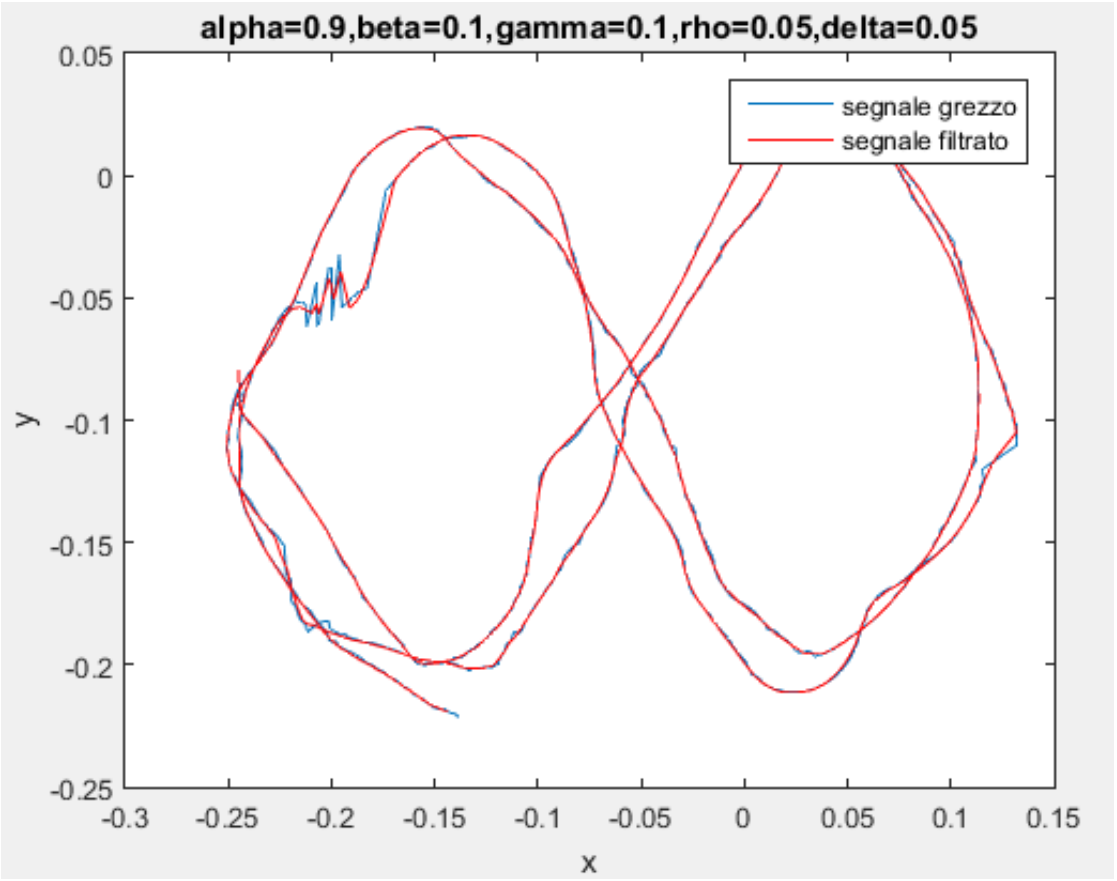
Fig. 21 – Durante il gioco si valutano 3 condizioni: cerchio verde, punteggio massimo, cerchio giallo metà punti e cerchio rosso nessun punto.

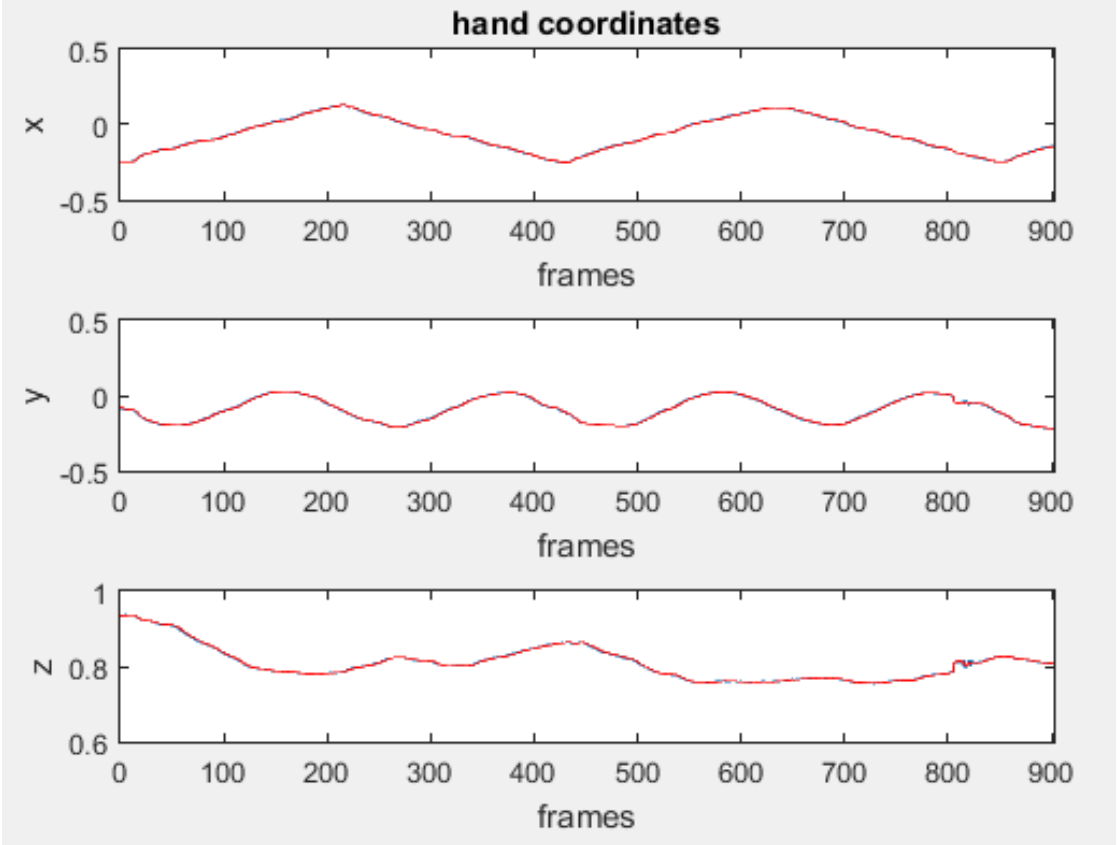
Operando in questo modo tuttavia il Joint spalla è rimasto troppo rumoroso per fare da riferimento tanto da far risultare più difficile da controllare il movimento della mano. Si è deciso quindi di bloccare il punto di riferimento alla posizione della spalla nell'esatto frame in cui inizia il gioco. Sono state fatte diverse registrazioni modificando di volta in volta i parametri per valutare in maniera empirica quali cambiamenti ciò comporta nell'hand tracking.
Ricordiamo che i parametri sono: α per lo smoothing, β per il trend, γ per la predizione, ρ è il raggio per rimuovere i Jitter e δ è il raggio per correggere la massima deviazione dal valore non filtrato.

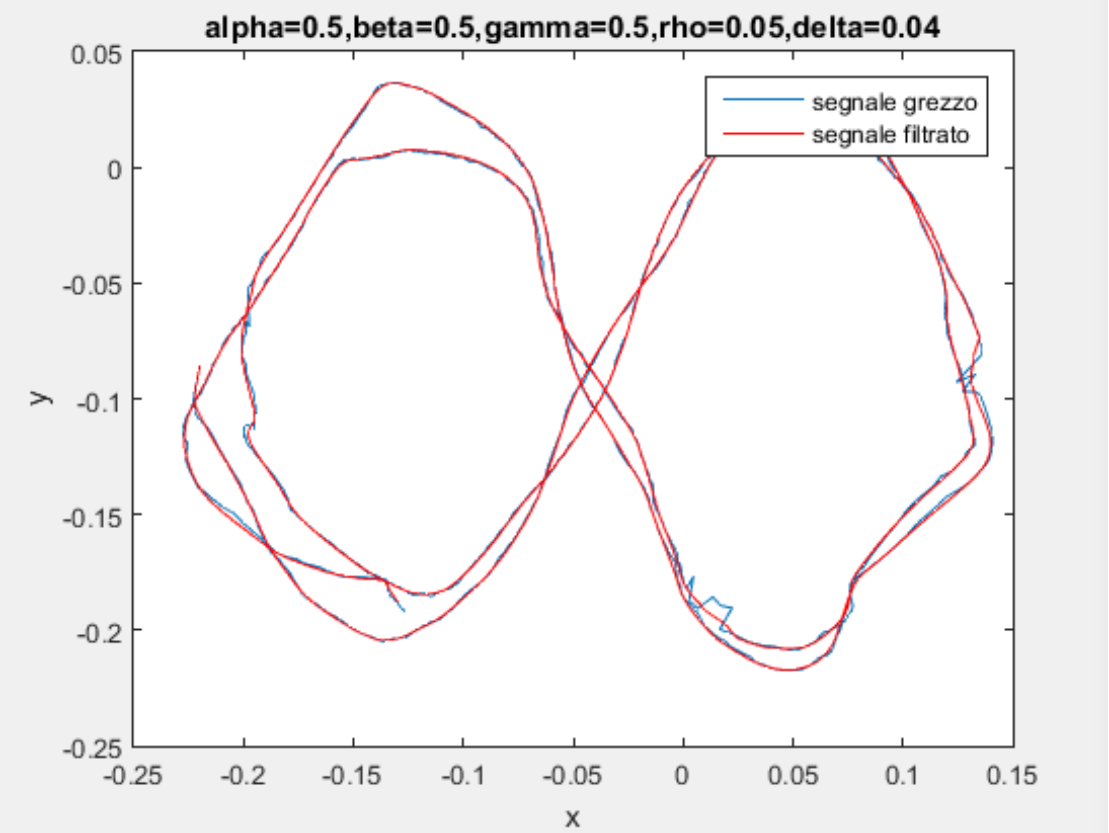
Fig. 22 – Sono mostrate in seguito 6 prove di filtraggio con l'uso di combinazioni diverse di parametri. Sopra è visualizzato il plot in X ed Y, coincidente con quanto visualizzato a schermo. Sotto l'analisi dello stesso tracciato separatamente per X, Y e Z. In rosso sono visualizzate le posizioni filtrate (quelle viste a schermo), mentre in blu è plottato il segnale originale non filtrato.

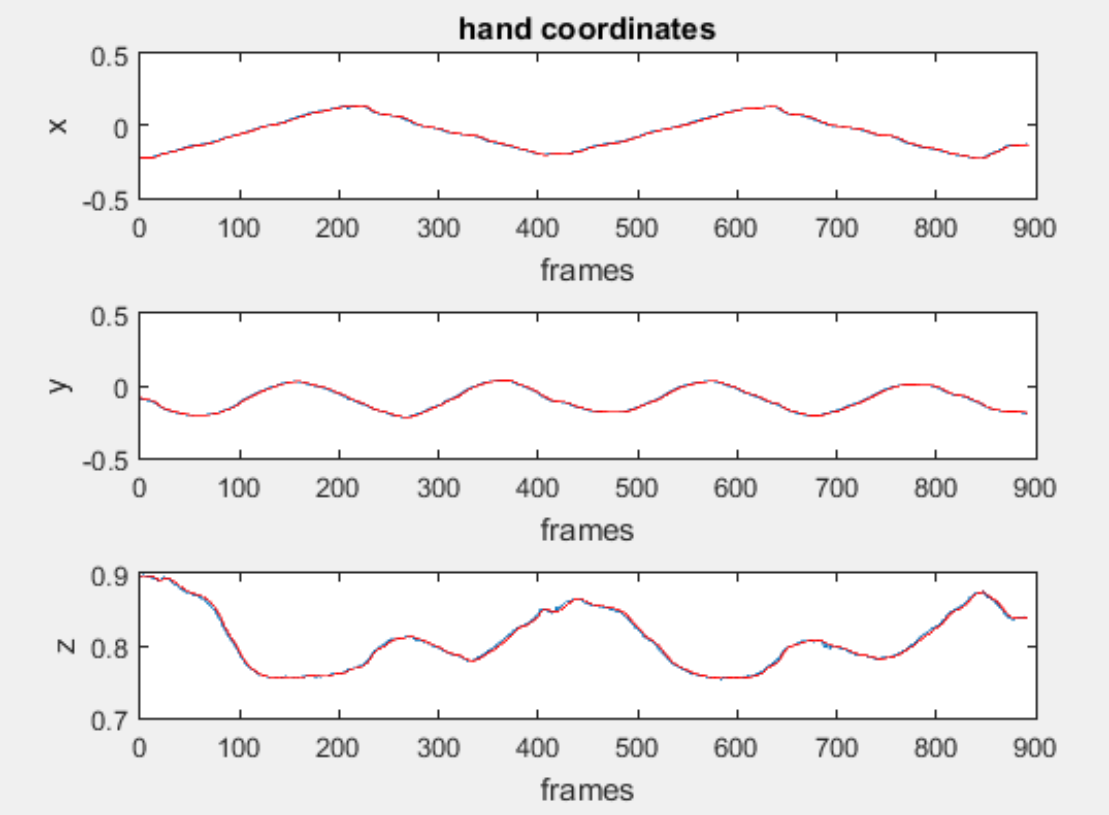

alpha=0.25,beta=0.25,gamma=0.25,rho=0.03,delta=0.05
segnale grezzo
segnale filtrato
y
x
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15

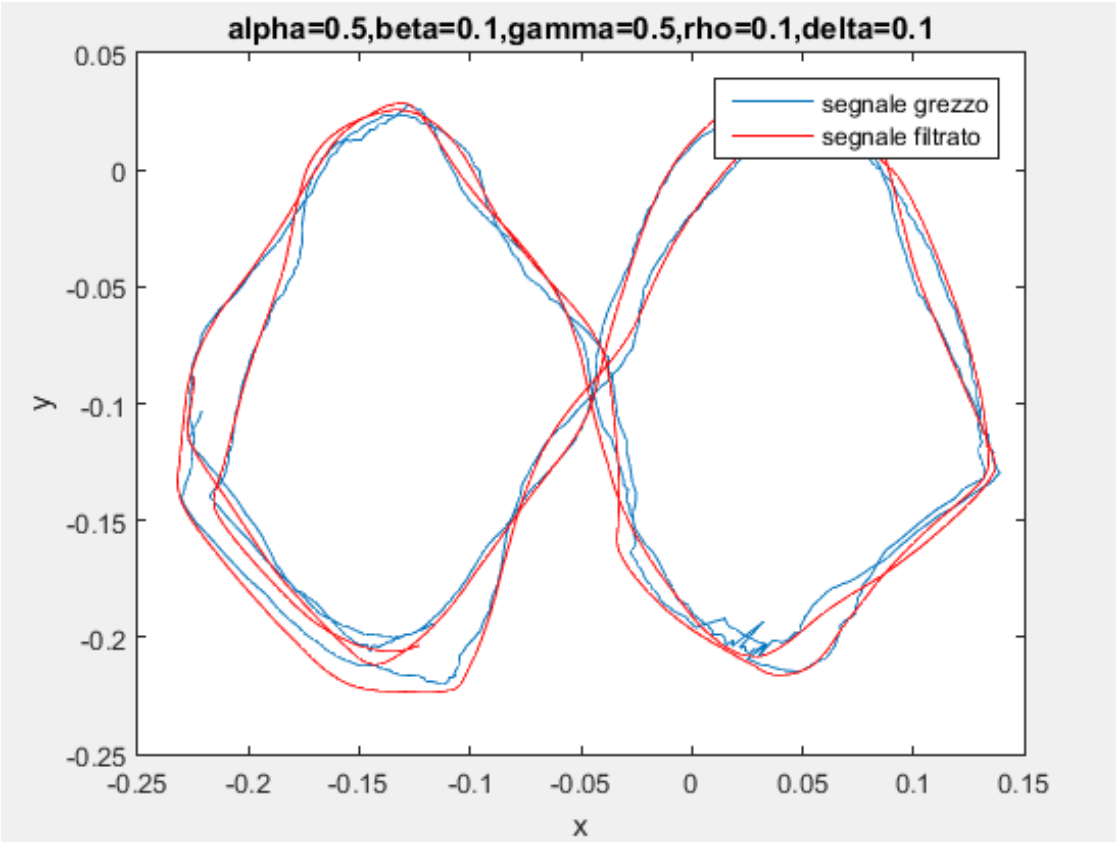

hand coordinates
x
y
z
frames
frames
frames
0.5
0
-0.5
1
0.8
0.6
0
100
200
300
400
500
600
700
800
900

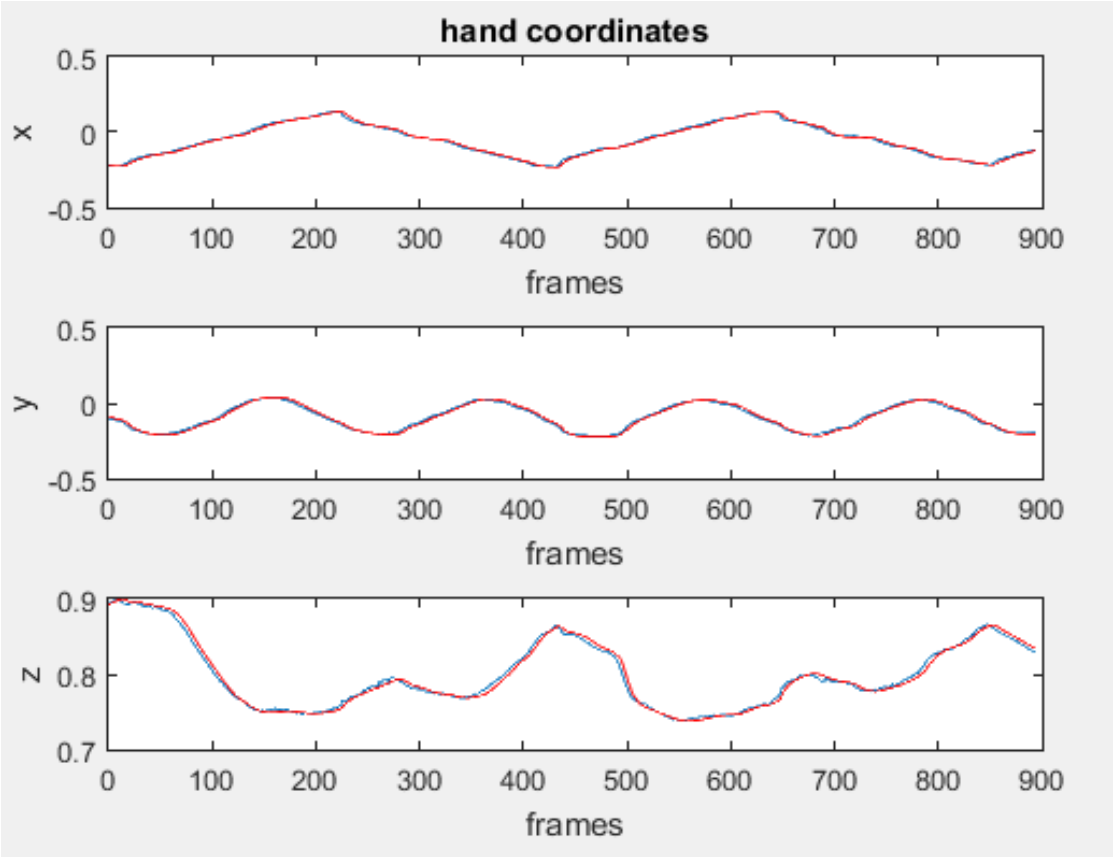

alpha=0.9,beta=0.1,gamma=0.1,rho=0.05,delta=0.05
segnale grezzo
segnale filtrato
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
y
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
x

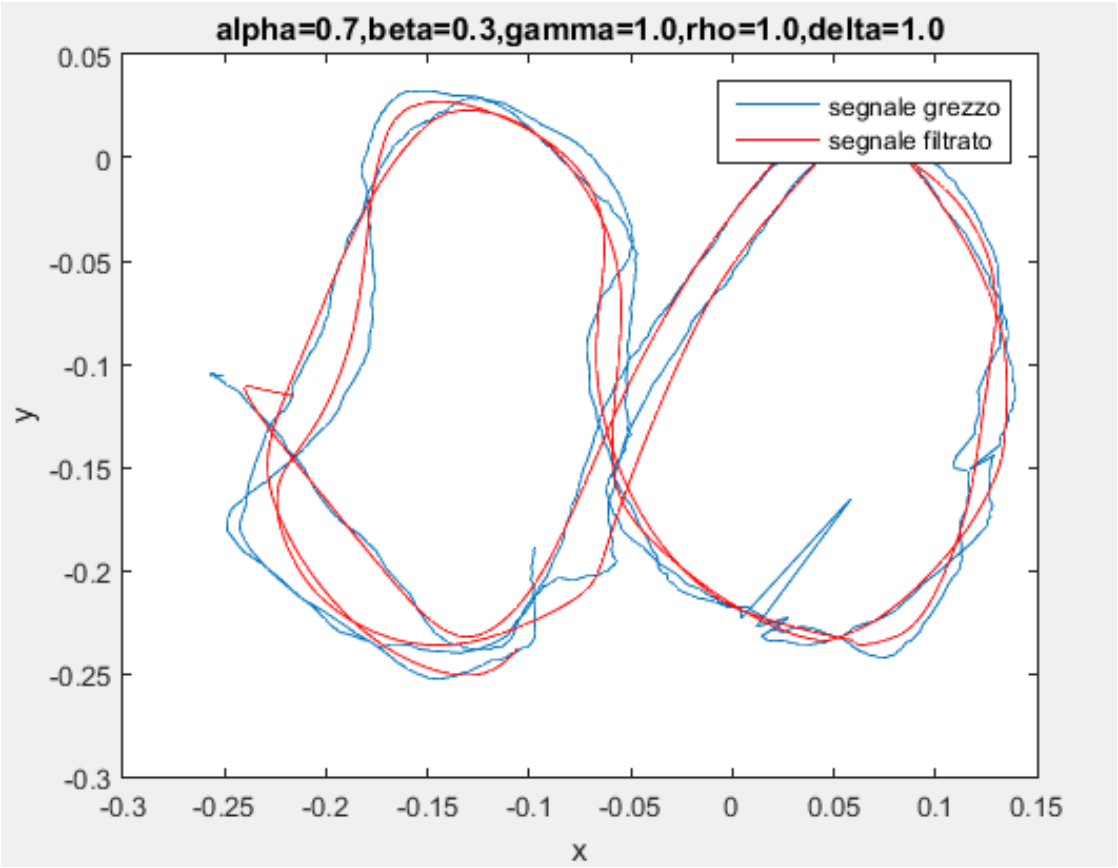

hand coordinates
0.5
0
-0.5
x
0
100
200
300
400
500
600
700
800
900
frames
0.5
0
-0.5
y
0
100
200
300
400
500
600
700
800
900
frames
1
0.8
0.6
z
0
100
200
300
400
500
600
700
800
900
frames

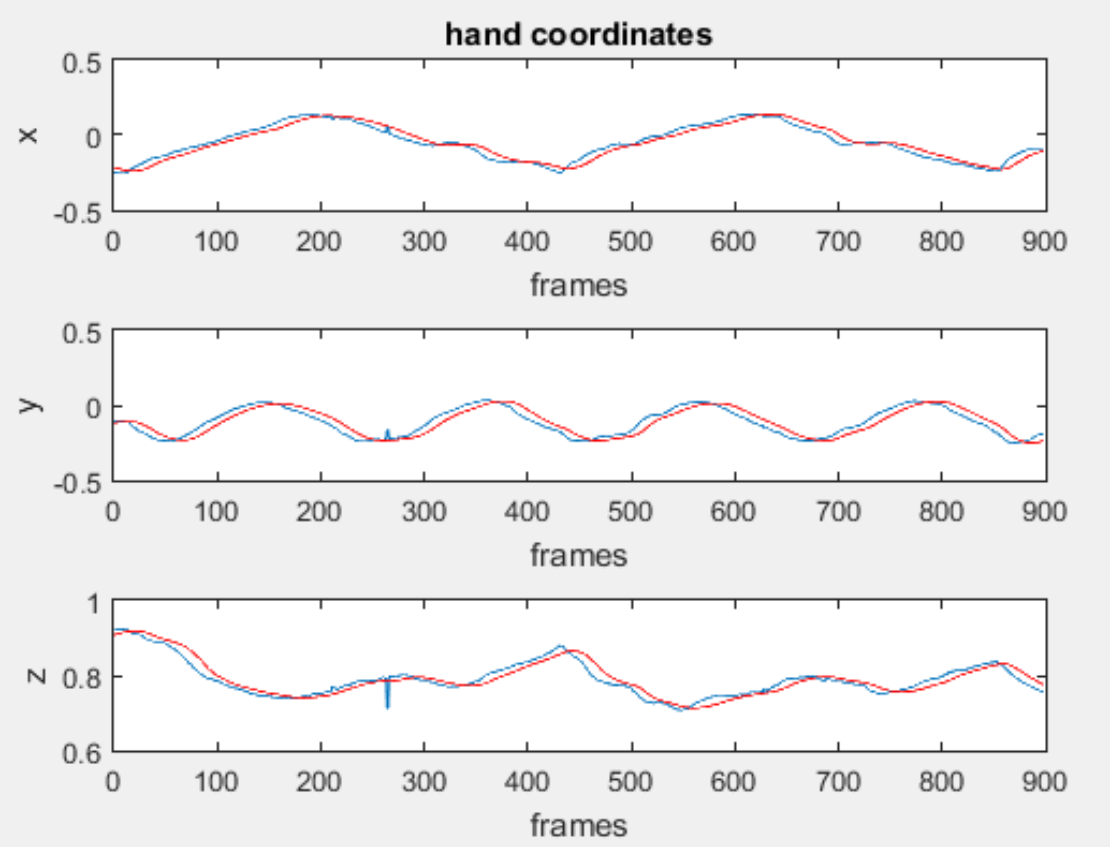

alpha=0.5,beta=0.5,gamma=0.5,rho=0.05,delta=0.04
segnale grezzo
segnale filtrato
y
x
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15

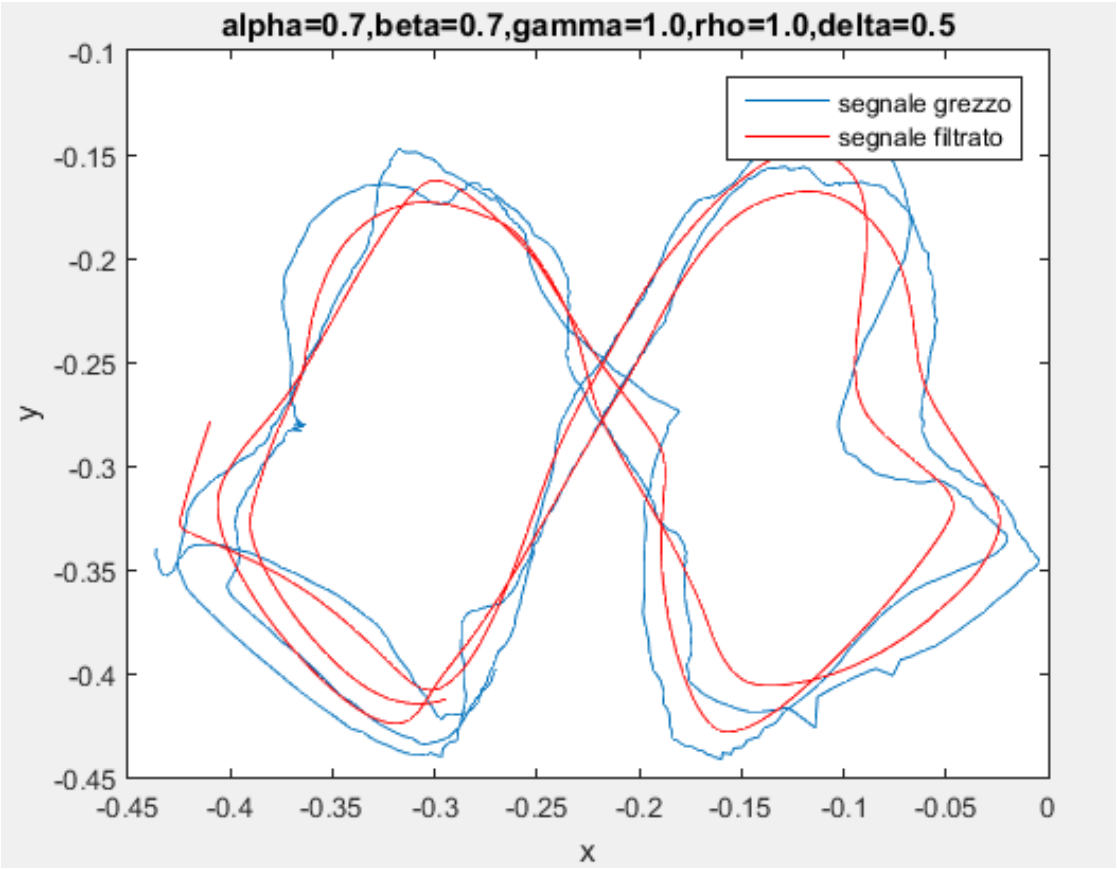

hand coordinates
x
y
z
frames
0.5
0
-0.5
0.9
0.8
0.7
0
100
200
300
400
500
600
700
800
900

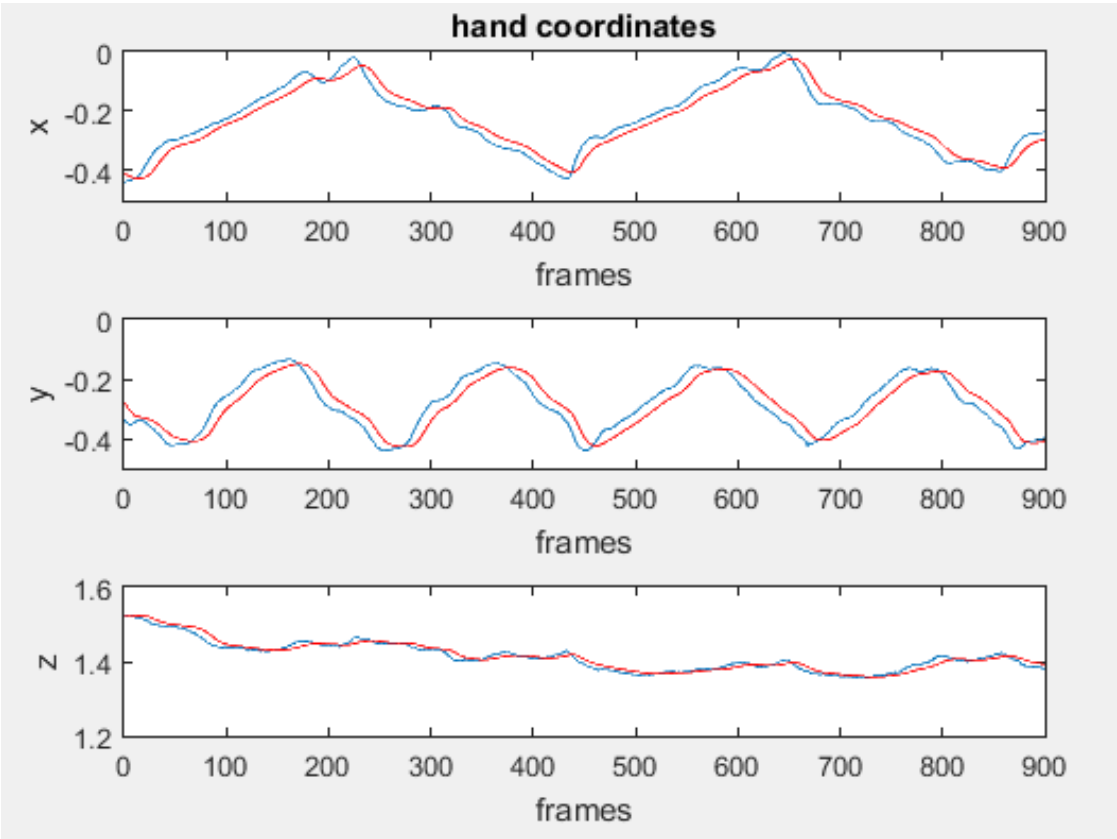

alpha=0.5,beta=0.1,gamma=0.5,rho=0.1,delta=0.1
segnale grezzo
segnale filtrato
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
y
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
x


hand coordinates
0.5
0
-0.5
x
0
100
200
300
400
500
600
700
800
900
frames
0.5
0
-0.5
y
0
100
200
300
400
500
600
700
800
900
frames
0.9
0.8
0.7
z
0
100
200
300
400
500
600
700
800
900
frames


alpha=0.7,beta=0.3,gamma=1.0,rho=1.0,delta=1.0
segnale grezzo
segnale filtrato
y
x
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15


hand coordinates
x
y
z
frames
frames
frames
0.5
0
-0.5
1
0.8
0.6
0
100
200
300
400
500
600
700
800
900


alpha=0.7,beta=0.7,gamma=1.0,rho=1.0,delta=0.5
segnale grezzo
segnale filtrato
y
x
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
-0.45
-0.45
-0.4
-0.35
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0


hand coordinates
x
y
z
frames
0
-0.2
-0.4
1.6
1.4
1.2
0
100
200
300
400
500
600
700
800
900

Al di là dei risultati evidenti di smoothing del segnale è importante dire che la scelta ottimale dei parametri va ponderata per lo scopo d'uso dell'applicazione. Ad esempio per questo gioco introdurre troppa latenza come negli ultimi due casi è risultato molto fastidioso per l'esperienza del giocatore. La sensazione provata è stata una sorta di viscosità dei movimenti che rende difficile seguire l'oggetto. Nelle prime due combinazioni di parametri invece il movimento dell'utente è rimasto molto fluido, ma con la percezione di essere troppo a scatti, con una sensazione meno fastidiosa della precedente, ma comunque non gradita. Le due combinazioni centrali invece sono state molto apprezzate nella pratica in quanto han prodotto una risposta ai movimenti del giocatore molto stabile e precisa. Nella pratica clinica si potrebbe pensare di fare come in figura 20 (pag.60), mettendo a disposizione del paziente 3 o 4 filtri e facendogli provare le diverse combinazioni per permettergli di scegliere quella a lui più congeniale. Diversamente, si potrebbe pensare di ottimizzare il filtro da utilizzare per una specifica applicazione facendola provare ad una popolazione di testing e misurandone la "soddisfazione" di utilizzo tramite questionari compilati "ad hoc". Senza filtraggio è comunque possibile giocare ma i Jitter e il rumore rendono il movimento del cursore troppo nervoso per risultare godibile.

Capitolo 6

Joint Orientation e Avateering

Finora si è parlato meramente di precisione del tracking in termine delle posizioni dei Joint in statica ed in parte in dinamica con l'apportamento di miglioramenti tramite filtraggio. Se l'idea finale è tuttavia di arrivare all'avateering, in altre parole di realizzare un avatar virtuale del paziente, questa forse non è la strada migliore. Il problema fondamentale è che queste posizioni, anche se ottimizzate dall'algoritmo di tracking e dal filtraggio, comunque nei movimenti perdono di stabilità e di conseguenza gli arti non presentano mai la stessa lunghezza per più frame. Il problema così definito è detto model stretching **[31]**. In termini generici è preferibile definire prima la lunghezza dei vari segmenti corporei ed in seguito utilizzare gli angoli articolari per orientarli nello spazio rispetto ad esempio ad una posizione iniziale o ad un determinato sistema di riferimento. Con il Kinect, in entrambe le versioni, è possibile ottenere informazioni sull'orientamento di ciascun Joint sotto forma di quaternioni dalla struttura `JointOrientation` **[37]**. Per capire bene il funzionamento di questo sistema, bisogna introdurre il concetto di "bone" (osso). Definendo bone il segmento che collega un Joint al successivo, il quaternione associato ad un certo Joint fornisce l'orientamento del bone genitore, di cui il Joint è l'estremità distale rispetto al corpo. È un concetto complicato che necessita di un esempio per essere capito bene. Se si considerano i Joint polso e gomito, l'informazione contenuta nel quaternione fornito dal polso si riferisce all'orientamento dell'avambraccio rispetto ad una posizione iniziale ben definita. Questa posizione è stata stimata essere avambraccio in verticale, con gli assi di rotazione coincidenti con la terna ortogonale del Kinect traslata sul

gomito. Per il gomito poi il Kinect fornisce il quaternione con l'orientamento del braccio rispetto alla medesima posizione iniziale, ma con riferimento la spalla e così via, seguendo una gerarchia ben specifica.

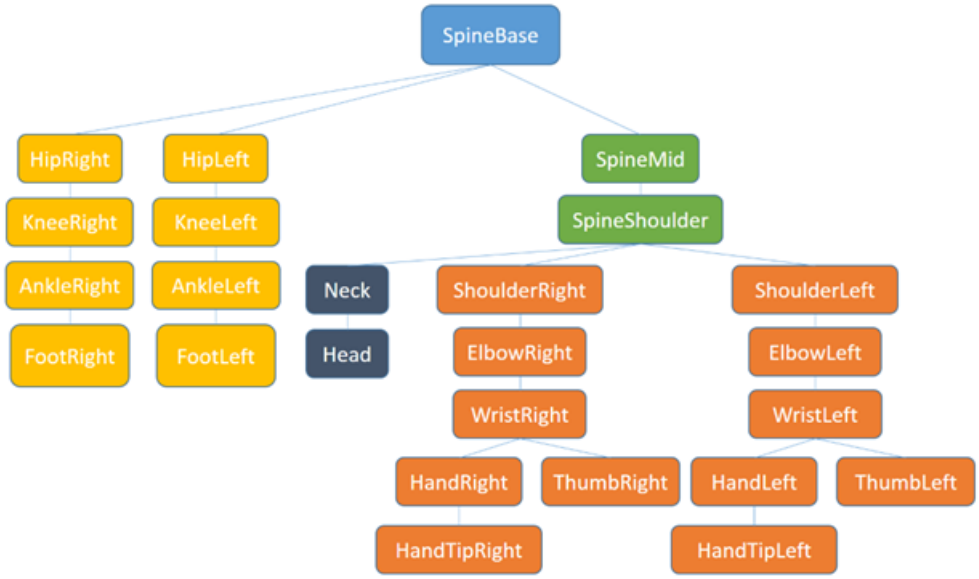


Fig. 23 – Gerarchia dei Joint per l'orientamento dei segmenti Joint-Joint anche detti "bone".

I quaternioni sono un sistema numerico, scoperto da William Rowan Hamilton nel 1843 che estende il piano complesso e rappresentano ad oggi un sistema largamente utilizzato in computer vision, in navigazione ed in aereonautica.
Un generico quaternione è definito dalla formula:

$$q = a + bi + cj + dk$$

Dove q è il quaternione, i, j e k sono numeri ipercomplessi e a, b, c, d sono coefficienti che definiscono la parte reale (a) e la parte immaginaria, o vettoriale (b, c e d) **[32]**.

Tralasciando la storia e le proprietà varie, il quaternione di rotazione può essere espresso anche nella forma alternativa:

$$\mathbf{q} = e^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

In cui si esprime la rotazione di un angolo θ attorno al versore $\hat{u}$.
Per cui tramite i coefficienti è possibile risalire ai coseni direttori e all'angolo di rotazione. Il Kinect fornisce proprio i quattro coefficienti a, b, c, d figurando la rotazione precedentemente descritta, partendo da segmento perfettamente verticale. Utilizzando i versori della terna di riferimento originale, quindi (1,0,0), (0,1,0), (0,0,1), ed associandoli alla condizione iniziale di ogni "bone" è facile ottenere i nuovi orientamenti di questi con la formula:

*p'=qpq**

p' rappresenta la nuova posizione del versore p, q e q* sono il quaternione ed il suo complesso coniugato. Il prodotto non è una semplice moltiplicazione, ma è un prodotto di Hamilton eseguito considerando che p sia un quaternione a parte immaginaria nulla.
Il prodotto di Hamilton pq ad esempio si calcola come:

$$\begin{aligned}
&a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\
&+ (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i \\
&+ (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j \\
&+ (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k.
\end{aligned}$$

In questo modo è possibile ottenere con 3 equazioni il nuovo orientamento dei 3 assi di rotazione sapendo che l'asse Y segue sempre la direzione del bone genitore, Z esprime la normale perpendicolare al bone e X esprime la binormale perpendicolare alla

normale e alla direzione del bone **[35]**. Il quaternione quindi rappresenta in maniera univoca l' orientamento tridimensionale del bone incernierato nel Joint gerarchicamente genitore.

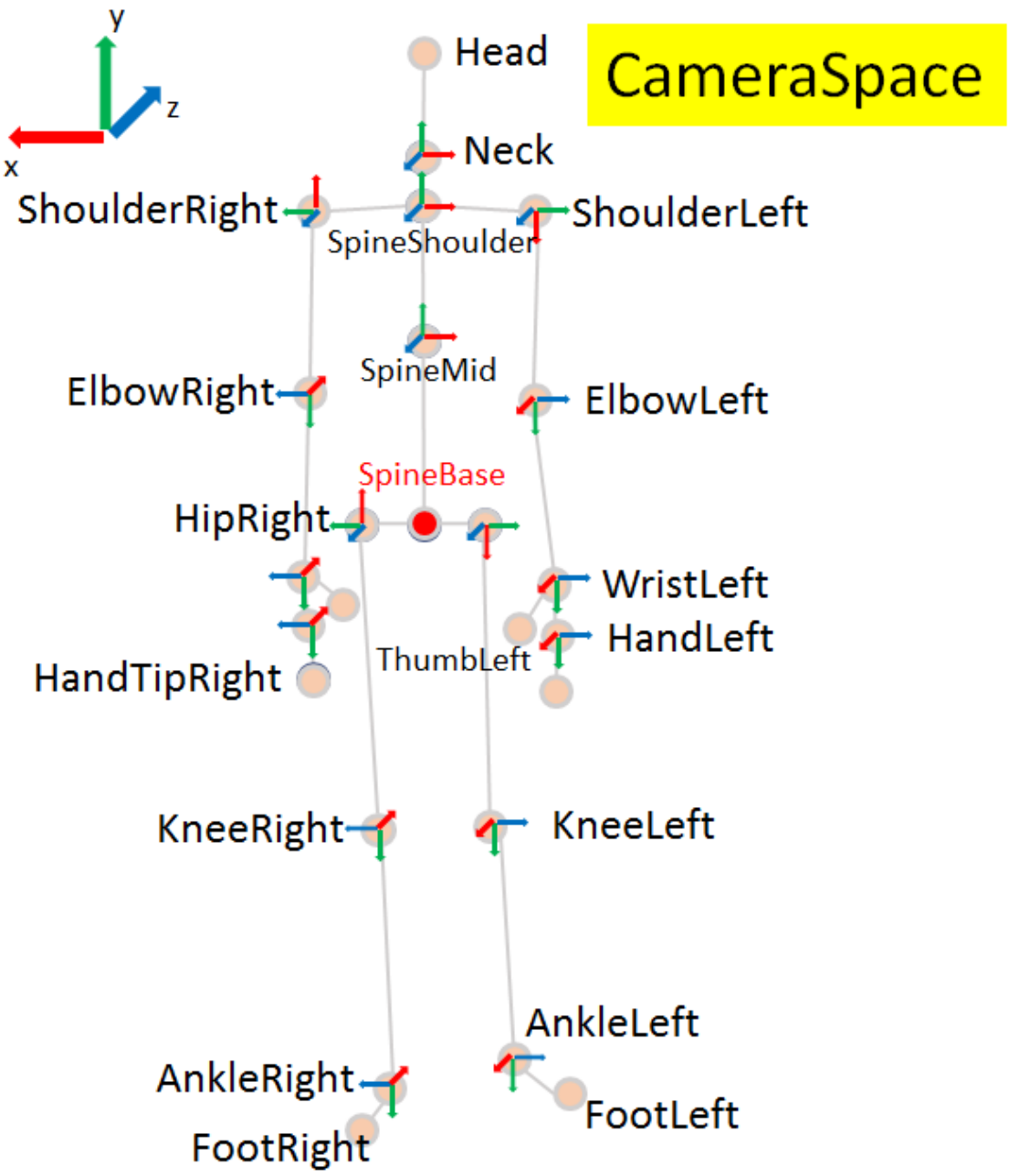


Fig. 24 – Sono rappresentate le nuove terne ottenute dalla formula p'=qpq in cui p è il versore originale rappresentato in alto a sinistra.*

Non si ottengono in questo modo dei gradi, ma dei nuovi versori. Nell' idea di voler stimare gli angoli articolari, si potrebbe pensare di trasformare i quaternioni ad esempio in angoli di Eulero nella variante di Tait-Bryan. Allora l'asse di rotazione X definirebbe lo Yaw (imbardata), l'asse Z definirebbe il Pitch (beccheggio) e l'asse Y il Roll (rollio). Ora gli angoli di Pitch e Yaw possono essere considerati anche di buona affidabilità nella stima rispettivamente di flesso- estensione e ab-adduzione **[33]**, ma per il Roll non è di dominio pubblico sapere come il Kinect possa riconoscere l'orientamento di un segmento corporeo attorno al suo asse **[33]**. Un ipotesi che sembrerebbe essere plausibile è che utilizzi la posizione dei pollici di mano e della posizione dei piedi come riferimento e che valuti il rollio partendo da questi e andando a ritroso. Per questo motivo le misure di roll forniteci dai quaternioni non possono essere considerate affidabili per valutare l'intra-extrarotazione. Se come da ipotesi l' algoritmo di orientamento si basa sulle posizioni dei Joint più distali, questo discorso vale tanto più per zone caratterizzate da Joint instabili, come l' arto superiore, in cui il Thumb (pollice) salta continuamente da pollice a mignolo non riuscendo a discriminarli correttamente. È sufficiente tenere il pugno chiuso infatti per notare che la rotazione del braccio non coincide più con quella figurata dal Kinect. Sempre per l'arto superiore poi non è neppure chiaro come il Kinect distribuisca la rotazione tra gomito e spalla in caso di braccio completamente steso.
A questo punto per ricavare gli angoli di Yaw, Pitch e Roll vale la trasformazione quaternione-angoli di Eulero:

$$\begin{bmatrix}\varphi\\ \vartheta\\ \gamma\end{bmatrix} = \begin{bmatrix}atan2(2(ab+cd), 1-2(b^2+c^2))\\ \text{asin}\,(2ab-da)\\ atan2(2(ad+bc), 1-2(c^2+d^2))\end{bmatrix}$$

La trasformazione in angoli di Eulero potrebbe tuttavia portare al problema del Gimbal lock in quanto si basa su rotational order. All'atto

pratico il problema è dovuto al fatto che viene creata una gerarchia tra Yaw, Pitch e Roll; in ambiente aereonautico ad esempio prima un aereo si orienta a terra con l'imbardata, poi decolla aggiustando il beccheggio ed infine corregge la rotta con il rollio. Associando a questi movimenti degli assi, prima si ha una rotazione attorno a X, poi attorno a Z ed infine attorno ad Y (nel sistema del Kinect). A causa del rotational order però, lo Yaw fa ruotare anche gli assi di Pitch e di Roll, il Pitch fa ruotare anche il Roll ma lascia inalterato lo Yaw mentre il Roll che si trova in fondo alla gerarchia non modifica gli altri assi di rotazione. Il risultato è analogo ad utilizzare 3 matrici di rotazione 3x3 in un ordine ben preciso. Ogni matrice rappresenta la rotazione attorno ad un asse, cambiando l'ordine cambiano i risultati. Il Gimbal lock è prodotto nel momento in cui modificando il secondo della gerarchia si porta l'asse ad essere allineato con quello del primo, perdendo così un grado di libertà. Fotografando quella situazione, ci si accorge che non è più possibile realizzare una delle tre rotazioni (la rotazione attorno al primo asse della gerarchia coincide con quella del terzo). Nell'esempio aereonautico, un beccheggio di 90° porta il suo asse a coincidere con quello di imbardata. Non è più possibile a questo punto imbardare, in quanto cambiamenti all'angolo di Yaw e all'angolo di Roll portano lo stesso tipo di rotazione. Questo concetto può essere spiegato matricialmente come:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Da una rotazione generica attorno agli assi X, Y, Z si decide di applicare una rotazione di 90° attorno ad Y.

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Moltiplicando tra loro le tre matrici e trasformando i risultato con le formule di equivalenza trigonometriche si ottiene:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{bmatrix}$$

In cui si può vedere bene come cambiamenti di α e γ portano agli stessi effetti di rotazione attorno a Z.

Nel movimento di un avatar o nel cercare di ricavare gli angoli articolari questo porta al problema che in alcuni passaggi non è possibile misurare o comandare correttamente lo spostamento dei segmenti corporei. Questo problema purtroppo è inevitabile e causa dei veri e propri salti nella misurazione degli angoli articolari attraverso gli angoli di Eulero.

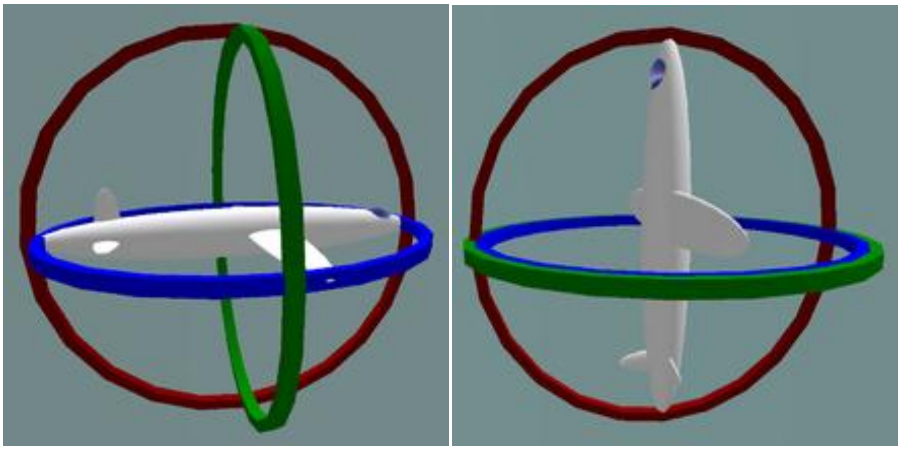


Fig. 25 – L'aereo (assimilabile ad un bone) a sinistra può ruotare secondo gli angoli di imbardata (rosso), di beccheggio (verde) e di rollio (blu). Quando il beccheggio si

allinea al rollio, un cambiamento nel Roll e nello Yaw applica lo stesso movimento. Si perde così un grado di libertà.

È necessario quindi trovare strade alternative. Una naturalmente può essere data dall'uso diretto dei quaternioni.Attraverso misure antropometriche o attraverso una misura iniziale in statica si definiscono delle lunghezze medie per i vari segmenti. Questi sono poi orientati nello spazio attraverso la formula

$$(p2 - p1) = |p2 - p1| * q(0,1,0)q^*$$

Il significato di questa equazione è stato già espresso in precedenza. Si prende la direzione di partenza (0,1,0) e si trova il nuovo versore tramite il prodotto di Hamilton. Questo è poi moltiplicato per la lunghezza del bone per ricavare il nuovo vettore di posizione del Joint distale (in cui è contenuto il quaternione q) **[36]**. Quanto poi questo sia ruotato lo si può vedere ripetendo il calcolo con (1,0,0) o (0,0,1) senza però dover ottenere dei vettori. Nell'idea di realizzare un avatar naturalmente le posizioni iniziali di questo devono coincidere con quelle del sistema di riferimento del Kinect per far si che i quaternioni producano lo stesso effetto **[34]**. Naturalmente anche in questo caso bisogna seguire la gerarchia da Joint genitore a figlio decidendo se utilizzate i quaternioni in forma assoluta (dal sistema di riferimento globale) o in forma relativa, moltiplicando di volta in volta il quaternione del Joint figlio con quello del Joint padre.
Per valutare invece gli angoli articolari è possibile prendere una doppia strada. È necessario innanzitutto definire per ogni Joint la terna anatomica di riferimento.

1. Si può tralasciare l'uso dei quaternioni e decidere di utilizzare le posizioni dei Joint (opportunamente filtrate) per poi calcolare gli angoli di flesso-estensione e di ab-adduzione rispetto alle terne

anatomiche precedentemente definite dall'esigenza clinica o da altre parti anatomiche.

2. Si può utilizzare il versore direzionale ottenuto dai quaternioni e valutare gli stessi angoli con questo. Naturalmente in questo caso bisogna essere consapevoli che la terna originaria di riferimento del quaternione è diversa da quella anatomica. In particolare la differenza varia per ogni articolazione a seconda della terna anatomica di riferimento ed è possibile utilizzare questo metodo solo se dotati in precedenza delle coordinate dei versori delle terne di ogni articolazione.

Visto che però l'instabilità di alcuni Joint rende poco credibile la stima della rotazione di un bone attorno al proprio asse, si potrebbe suggerire di coadiuvare l'uso del Kinect con ad esempio l' uso di sensori inerziali. Un sistema di 3 accelerometri triassiali posto sul dorso della mano potrebbero fornire la rotazione di questa in maniera più precisa di quanto faccia il Kinect rendendo più robusta la misura. Esistono equazioni per trasformare l'uscita in funzione di g in quaternioni senza dover passare dagli angoli di Eulero **[38]**.
Gli accelerometri forniscono una buona base da cui partire per determinare l'orientamento della mano, anche se quando uno dei 3 assi degli accelerometri si allinea a g si perde un grado di libertà **[39]**.
L'uso del giroscopio può colmare questo limite. Considerando inoltre che i Joint Hand e Hand-tip sono piuttosto stabili e che il polso ha 2 gradi di libertà (non permette l'intra-extrarotazione) si può anche valutare di integrare l'informazione sull'orientamento dato dal kinect con eventuali misure di accelerometro (triassiale) e giroscopio (triassiale). In questo modo si vuole sfruttare la maggior precisione di questi sensori inerziali e la versatilità del Kinect per realizzare misure più complete ed affidabili utilizzando anche in supporto strumenti molto potenti come ad esempio un filtro di Kalman. Poiché però utilizzare il filtro di Kalman per ottenere gli angoli di rotazione potrebbe

riportare al fenomeno del Gimbal lock, un diverso modus operandi più completo sarebbe quello di utilizzare Kalman, o un filtro dalle caratteristiche simili, sfruttando direttamente i quaternioni **[32, 40]**.

Capitolo 7

Considerazioni pratiche

Vengono infine presentate alcune considerazioni sull'uso del Kinect, basate sul periodo di utilizzo del sensore e sui test eseguiti. Innanzitutto è bene confermare che il Kinect v2 è uno strumento molto potente rispetto al prezzo molto contenuto, ma che purtroppo soffre anche di alcune debolezze che lo rendono più fragile in alcune applicazioni d'uso. Per quanto riguarda il Body tracking, il sensore riesce a riconoscere una persona solo se questa è di fronte ad esso con viso rivolto alla camera e ben visibile dalla stessa. L'algoritmo di tracking a questo punto però diventa molto robusto tanto da riuscire a seguire la persona anche in movimenti quali la rotazione o il cammino. Anche mettendosi di profilo, ma sempre e solo dopo esser stati inizialmente individuati, il Kinect è in grado di seguire molto bene l'utente. Si potrebbe far tesoro di questa considerazione ad esempio per compiere eventuali studi di fattibilità di Gait Analysis utilizzando il Kinect. In questa condizione si è notata una certa instabilità nel tracking del braccio nascosto alla telecamera, ma la rispettiva gamba è risultata essere molto più fedele al movimento della persona. Il problema più grosso in questo senso è dovuto al campo di visione del Kinect che ovviamente è limitato ai suoi 70 gradi in orizzontale. Si suggerisce inoltre in questa eventuale analisi di tenere il Kinect ad un altezza non superiore al metro. A questo proposito, è stato appurato che il tracking non è tanto influenzato dalla distanza dell'utente, stando entro i 3 metri, quanto più dall'altezza in cui è sistemato il sensore. In particolare tanto più è messo in alto, tanto più deve essere inclinato per vedere l'utente a breve distanza. Questo porta ad un peggioramento della precisione del tracking e ad una maggiore

difficoltà nel riconoscere inizialmente l'utente. Si è visto ad esempio che inclinando oltre i –50 gradi il sensore, questo fatica veramente molto a riconoscere un soggetto umano, tanto che per iniziare il tracking la persona deve piegarsi all'indietro per far vedere bene viso e busto. Nello studio di questa situazione il sensore è stato alzato a circa 2 metri da terra e il soggetto è stato tenuto nell'intorno del metro di distanza. In questi casi si è reso opportuno allontanare l'utente e inclinare di meno il sensore per riottenere un tracking più accurato. Anche nella pessima condizione precedentemente esposta il Kinect, una volta iniziato il tracking, continua a stimare le posizioni dei Joint ad ogni frame senza mai perdere di vista il body. L'accuratezza delle posizioni però ne risente in quanto l'algoritmo sembra non essere ottimizzato per questi casi anomali. In particolare gli arti inferiori tendono ad assumere posizioni diverse da quelle reali, talvolta addirittura assurde. Similmente, tra le configurazioni mal riconosciute dal Kinect troviamo anche quella del paziente sdraiato. Si è provato a far stendere una persona su un piedistallo in legno alto 70 cm e si è tentato di farlo riconoscere al Kinect. Specialmente in questo caso la parte più difficile è risultata essere il far individuare la persona alla camera. Impossibile farlo se questa fosse fatta sdraiare parallelamente all'asse x della camera, mentre è risultata più individuabile se sdraiata perpendicolarmente. La difficoltà in questa seconda opzione risiede nel far entrare tutto il corpo nel campo di visione limitato ai 60° di angolo verticale. Si è provato anche ad utilizzare una visione ad uccello, ma il Kinect avrebbe dovuto essere stato alzato oltre i due metri e mezzo per poter vedere dalla testa ai piedi la persona stesa. Nel tracking in queste pose si è persa totalmente sia la precisione sia l'accuratezza, tanto che è bastato che l'utente si muovesse un po' che i Joint degli arti si posizionassero in punti non veritieri più o meno casuali.

Il tracking di arto superiore è risultato essere in qualunque configurazione sempre migliore di quello di arto inferiore, ad eccezione di una grossa instabilità sulla posizione dei pollici di mano. Se questa

non è tenuta ben aperta con dita serrate e pollice separato la posizione salta continuamente tra pollice e mignolo. Questo naturalmente si traduce in una grossa difficoltà a considerare valide le informazioni sull'orientamento del polso. Per questa articolazione inoltre si è notato che in esercizi di presa o comunque di hand tracking, la posizione del rispettivo Joint è spesso modificata dall'algoritmo del Kinect in quanto la mano spesso nasconde la vista del polso alla camera; analogamente ciò è successo anche per gomito e spalla. Il filtraggio fortunatamente riesce a coprire in buona parte questo difetto.

Per le applicazioni come l'esercizio esposto nel capitolo 5 o nell'utilizzare il Kinect come controller, l' ancoraggio della spalla con la media cumulativa, come suggerito da Microsoft, non ha prodotto la stabilità auspicata. È stato necessario bloccare il punto ad una posizione spaziale fissa per stabilizzare di conseguenza il tracking della mano. Nella realizzazione di piani o di nuove terne ortogonali bisogna fare attenzione anche all'allineamento dei Joint che non è sempre corrispondente a come ci si aspetterebbe. Sempre nell'esercizio proposto nel capitolo 5 il sistema di riferimento locale è stato generato sulla spalla e con asse x lungo la direzione dal Joint Spine_Shoulder, che si trova sulla spina dorsale all'altezza delle spalle, al Joint Shoulder. Ora, solitamente spalla e Spine_Shoulder sono alla stessa quota, ma in alcuni casi la spalla è risultata essere qualche centimetro sotto e questo ha portato alla creazione di una terna con asse x non parallelo al terreno. Questa situazione si è tradotta naturalmente in una difficoltà maggiore nel controllo dell'oggetto a schermo. Per questi motivi ci si sente di consigliare di riservare, se possibile, una parte dello schermo per visualizzare ciò che il Kinect sta effettivamente vedendo, magari in un riquadrino separato. È un dettaglio molto importante nelle applicazioni di realtà virtuale in cui si deve avere il controllo di un oggetto o del mouse. Errori nell'algoritmo di tracking o l'uso di riferimenti in posizioni diverse da quelle previste portano al

caos nel controllo. Solo vedendo ciò che il Kinect vede realmente questi fraintendimenti possono essere individuati e gestiti.
Il tracking con l'utente in piedi è sicuramente la configurazione migliore in cui utilizzare il Kinect, si consiglia a questo proposito di entrare sulla scena da lontano e non da dietro il Kinect. Si è notato infatti che se ci si fa riconoscere completamente dal sensore, facendogli traccare da subito tutti i 25 Joint, aumenta la precisione delle posizioni. In particolare questo si è notato eseguendo i test al capitolo 4 e notando che per il soggetto M.M. che si era fatto entrare in scena da dietro il Kinect, le deviazioni standard siano state più alte delle altre anche di un ordine di grandezza. Arrivando da lontano e facendosi traccare tutto il corpo si permette di avere un tracking stabile e preciso anche se poi ci si avvicina al sensore nascondendogli parte del corpo. Comparendo nel campo visivo del Kinect con già una parte del corpo non visibile invece può portare a stime inverosimili della posizioni dei joint non mostratigli con ripercussioni anche gravi sui Joint visibili. Ovviamente anche in questo caso la visualizzazione a schermo del body tracking permetterebbe di notare la presenza di grossi errori.
Nel Kinect v2 è stata eliminata la Seated mode, il tracking da seduto è comunque piuttosto buono; per l'arto superiore accuratezza e precisione sembrerebbero affidabili quanto le misure in piedi. In particolare i Joint Hand_tip ed Hand, che si localizzano rispettivamente sul dito medio e sul palmo della mano, presentano generalmente una precisione molto buona in X e Y, ma accusano una scarsa precisione in profondità. Questo cattivo trend è però comune per tutti i Joint. In figura sotto vengono presentati due esempi di misure di deviazioni standard in x, y e z (nel sistema di riferimento globale) di tutti i 25 Joint. Le registrazioni sono state effettuate per 2 persone alla stessa distanza ed alla stessa altezza del Kinect. Si nota bene come in genere le deviazioni standard in x ed in y siano minori anche di un ordine di grandezza rispetto a z (profondità).

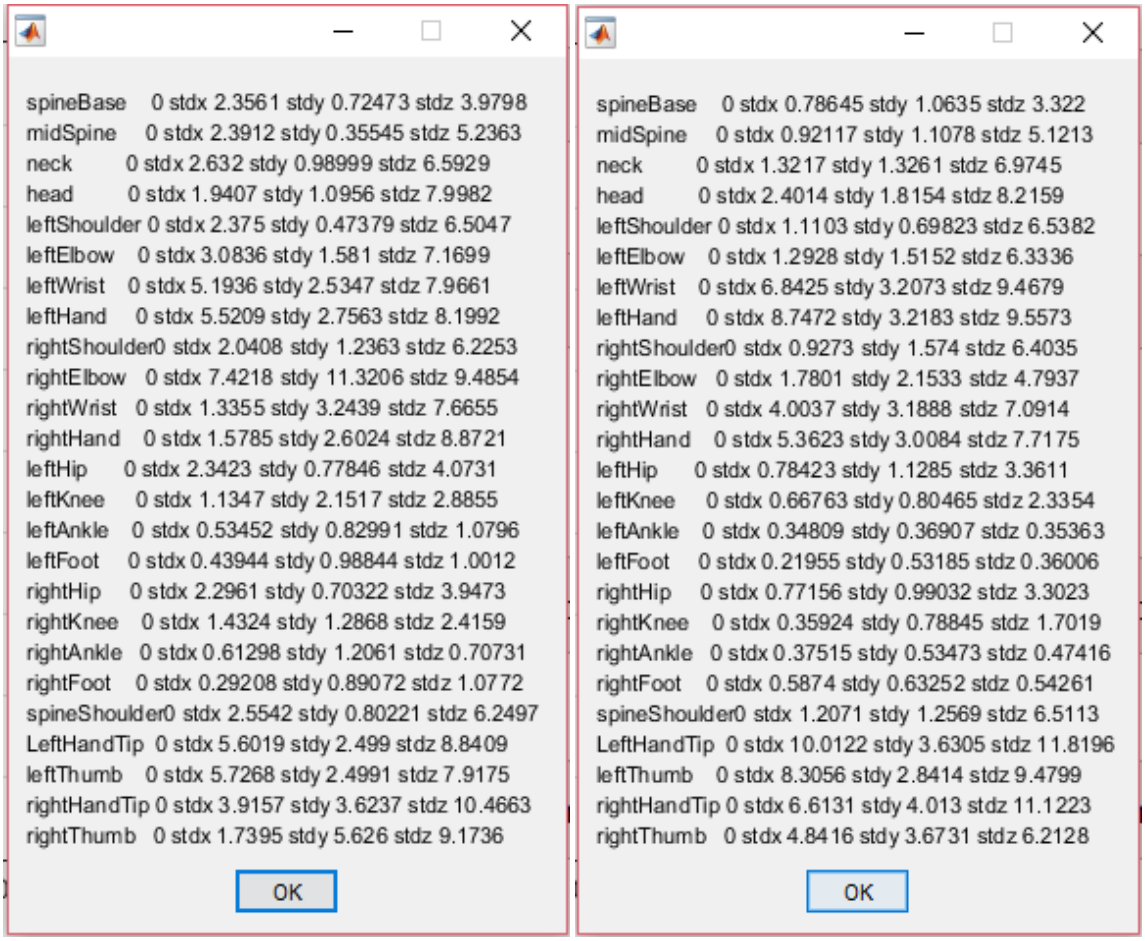


Fig. 26 – schema delle deviazioni standard per i 25 Joint nelle coordinate x, y e z.Le misure sono in mm.

Per eseguire eventuali prove in cui si necessita di valutare la presenza di compenso da parte del paziente, come ad esempio vedere se in un esercizio in cui bisogna muovere solo il braccio il paziente si aiuta con anche il movimento del busto, si dovrebbe tenere in considerazione questa differenza di precisione e valutare se sia il caso di mettere il paziente di profilo.

Per l'arto inferiore si è notato generalmente un calo di accuratezza nelle prove condotte a causa del fatto che generalmente queste vengono eseguite da seduto. La presenza di sedie, tavoli o altri oggetti sulla scena non produce effetti negativi a meno che non siano nelle immediate vicinanze del soggetto. Per cui l'oggetto su cui è seduto il

paziente talvolta può confondere il tracking. Il discorso è valido anche per oggetti a contatto di un utente in piedi, come un attaccapanni troppo vicino. Un'altra fonte di disturbo nell'ottica di utilizzare il Kinect in clinica è dovuta alla presenza di un'altra persona sulla scena. Considerando ad esempio un paziente che deve effettuare un movimento guidato da un fisioterapista, il Kinect non faticherebbe a determinare la presenza di due corpi, avrebbe difficoltà invece a distinguere correttamente le porzioni di corpo di uno e dell' altro nelle zone a contatto. Inevitabilmente un Joint non può essere coperto da nient'altro per essere traccato bene.

Infine abbiamo valutato che fonti di luce artificiale non influenzano la precisione del Kinect. La luce solare invece influisce creando falsi body solo però, se la fonte da cui entra in scena si trova nel campo visivo della telecamera. Questi falsi body possono essere creati anche da riflessi prodotti su oggetti particolarmente lucidi. Naturalmente sono da evitare specchi sulla scena e oggetti che possono influenzare gli infrarossi. Sarebbe stato interessante valutare l'influenza di altri oggetti emettenti infrarossi come un Leap Motion o un altro Kinect, ma non è stato possibile reperirne. Per il Kinect v1 era stata data la possibilità di utilizzare più Kinect nella stessa stanza, con il V2 no, probabilmente per via delle due diverse tecnologie di misurazione della profondità.

Valutando il metodo offerto dal Kinect per stimare l'orientamento dei Joint, si ritiene robusto per realizzare e muovere un avatar, ma migliorabile con il supporto di sensori inerziali. Qui la prova con un Leap Motion per verificare la possibilità di una convivenza dei due sensori sarebbe stata molto interessante. Per la valutazione degli angoli articolari, fermo restando che qualunque errore sul tracking delle posizioni è raddoppiato nella valutazione degli angoli, scegliendo opportunamente i riferimenti è risultato immediato trovare gli orientamenti dei bone nello spazio tramite i quaternioni. La cosa difficile è probabilmente scegliere per ogni Joint o per ogni bone la

configurazione di partenza dal quale far partire la misura degli angoli. Naturalmente non ha molto senso clinico partire dalla condizione di segmento verticale e stimare i 3 angoli di rotazione da questa posizione. Misurando la flesso-estensione del gomito ad esempio bisognerebbe decidere se valutarne l'angolo partendo da un piano passante per spalla e gomito o da un piano verticale passante per quest' ultimo, ma non da avambraccio rivolto al cielo. Per queste decisioni si rimanda poi alla scelta del clinico o alla letteratura.

Per concludere questo capitolo riflessivo è necessario parlare un attimo della programmazione del Kinect. I linguaggi di programmazione permessi sono C++, C# e Visual basic. In questo progetto si è scelto di utilizzare C# ed è stato necessario effettuare uno studio preparativo almeno della base di programmazione in questo linguaggio. A questo riguardo va fatto notare anche che, mentre per il Kinect V1 sul sito Microsoft le librerie sono molto ben spiegate, con esempi pratici in tutti e tre i linguaggi per le varie classi e le varie strutture, per il V2 sono presenti solo le descrizioni lasciando un amaro senso di desolazione. Fortunatamente si è trovato risposta a molte domande sul forum Microsoft e su Blog di collaboratori ufficiali.

Conclusione

In questo percorso è stato studiato, valutato ed utilizzato il Microsoft Kinect One. Partendo nell'ottica di voler verificare se questo strumento, economico e potente fosse in grado di effettuare il motion tracking di un paziente in ambito riabilitativo, dopo le prove più svariate presentate in questo elaborato, si è arrivati alla conclusione che l'algoritmo di tracking implementato nel Firmware del Kinect è sì molto robusto, ma anche poco affidabile se non opportunamente utilizzato. È stato possibile verificare che il tracking una volta individuata la persona segue sempre la figura dell'individuo, purché questa rimanga all'interno del campo di visione almeno con metà del corpo. L'altra faccia della medaglia è che purtroppo basta poco per far si che l'accuratezza delle posizioni degradi molto. Un arto che ne copre un altro, un oggetto sulla scena o una posizione diversa da quella frontale possono portare ad instabilità i Joint stimati con inevitabili ripercussioni nella traduzione di certi movimenti in realtà virtuale. In quest'ottica si è valutato il tracking delle posizioni dei Joint in termine di precisione che si è attestata essere dell'ordine dei millimetri nella maggior parte dei casi. È stato valutato il filtraggio doppio esponenziale implementato nel vecchio Kinect v1 e riprodotto come classe con cui lavorare in C# sviluppando per l' occasione un gioco in cui seguire con la mano un oggetto che si muovesse sullo schermo. Infine sono state date delle valutazioni sull'uso del Kinect in diverse possibili situazioni cliniche.

Si è cercata di rendere la valutazione sul sensore il più obbiettiva possibile attraverso le prove descritte in precedenza. Arrivando alla fine del percorso si ritiene che la validità d'uso Kinect sia relativa alla destinazione di utilizzo. In particolare si ritiene poco consigliabile utilizzarla per quelle applicazioni dove si necessità di un accuratezza minore del centimetro, mentre rappresenta un sistema incredibilmente

vantaggioso nel caso in cui si debbano fare stime total body o comunque di posizione di grosse parti anatomiche. In sostanza potrebbe essere una buona scelta in generale utilizzare una metodologia di tracking molto accurata per le misure di precisione congiuntamente all'uso del Kinect per valutare compensi, piani anatomici e posizione spaziale della persona nell'ambiente di lavoro.

Bibliografia

[1] M.Pirini, R. Stagni - *Ambienti virtuali per la riabilitazione motoria* – slide del corso Biongegneria della riabilitazione, 2015;

[2] http://www.leonardoausili.com/approfondimenti/la-realta-virtuale-nella-riabilitazione-delle-disabilita-motorie;

[3] Ioannis Theoklitos Paraskevopoulos, Emmanuel Tsekleves, Cathy Craig, Caroline Whyatt, John Cosmas - *Design guidelines for developing customised serious games for Parkinson's Disease rehabilitation using bespoke game sensors* - Entertainment Computing 5 (2014), p. 413–424;

[4] Maureen K. Holden - *Virtual Environments for Motor Rehabilitation: Review* – Cyberpsychology & Behavior Volume 8, Number 3, 2005;

[5] Mindy F. Levin, Patrice L. Weiss and Emily A. Keshner - *Emergence of Virtual Reality as a Tool for Upper Limb Rehabilitation: Incorporation of Motor Control and Motor Learning Principles* - Phys Ther. 2015, p. 95:415-425;

[6] Eric B. Larsona, Maia Feigonb, Pablo Gagliardod, Assaf Y. Dvorkina - *Virtual reality and cognitive rehabilitation: A review of current outcome research* - NeuroRehabilitation 34 (2014), p. 759–772;

[7] Frederick H. Raab, Ernest B. Blood, Terry 0. Steiner, Herbert R. Jones - *Magnetic position and orientation tracking system* - Ieee Transactions On Aerospace And Electronic Systems Vol. Aes-15, No. 5 September 1979, p. 709-718;

[8] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak and Denis Fisseler - *Analysis of the Accuracy and Robustness of the Leap Motion Controller* - Sensors 2013, 13(5), p. 6380-6393;

[9] Adso Fernandez Baena, Antonio Susin, Xavier Lligadas - *Biomechanical Validation of Upper body and Lower body Joint Movements of Kinect Motion Capture Data* - Intelligent Networking and Collaborative Systems (INCoS), 2012 4th International Conference on, p. 656 - 661;

[10] Patrice L Weiss, Debbie Rand, Noomi Katz, Rachel Kizony - *Video capture virtual reality as a flexible and effective rehabilitation tool* - Journal of NeuroEngineering and Rehabilitation 2004, p. 1-12;

[11] Nils Hasler, Bodo Rosenhahn, Thorsten Thorm¨ahlen, Michael Wand, Juergen Gall1, Hans Peter Seidel - *Markerless Motion Capture with Unsynchronized Moving Cameras* - Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, p. 224 - 231;

[12] Somphong Phommahavong, Dominik Haas, Jing Yu, Sabine Krüger-Ziolek, Knut Möller, Jörn Kretschmer - *Evaluating the microsoft kinect skeleton joint tracking as a tool for home-based physiotherapy* - Current Directions in Biomedical Engineering 2015; p. 184–187;

[13] HIG 1.8: human interface guideline ver. 1.8 - https://msdn.microsoft.com/en-us/library/jj663791.aspx;

[14] HIG 2.0: human interface guideline ver. 2.0 - http://download.microsoft.com/download/6/7/6/676611B4-1982-47A4-A42E-4CF84E1095A8/KinectHIG.2.0.pdf;

[15] Clemens Amon, Ferdinand Fuhrmann - *Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows*

v1 and v2 - 6th Congress of Alps-Adria Acoustics Assosiation 16.-17. October 2014 Graz, Austria;

[16] Hamed Sarbolandi, Damien Leoch, Andreas Kolb - *Kinect Range Sensing: Structured-Light versus Time-of-Flight* - KinectJournal of Computer Vision and Image understanding May 21, 2015;

[17] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio,Richard Moore, AlexKipman, Andrew Blake - *Real-Time Human Pose Recognition in Parts from Single Depth Images* - Microsoft reasearch;

[18] Loren Arthur Schwarz, Artashes Mkhitaryan, Diana Mateus, Nassir Navab - *Human skeleton tracking from depth data using geodesic distances and optical flow* - Image and Vision Computing 30 (2012) 217–226;

[19] David Webster, Ozkan Celik - *Experimental Evaluation of Microsoft Kinect's Accuracy and Capture Rate for Stroke Rehabilitation Applications* - IEEE Haptics Symposium 2014, 23-26 February, Houston, Tx, USA;

[20] Tilak Dutta - *Evaluation of the Kinect sensor for 3-D kinematic measurement in the workplace* - Applied Ergonomics 43 (2012), p. 645-649;

[21] Xu Xu, Raymond W. McGorry - *The validity of the first and second generation Microsoft Kinect™ for identifying joint center locations during static postures* - Applied Ergonomics 49(2015), p.47-54;

[22] Lin Yang, Longyu Zhang, Student Member, Haiwei Dong, Abdulhameed Alelaiwi, Abdulmotaleb El Saddik, - *Evaluating and*

Improving the Depth Accuracy of Kinect for Windows v2 - IEEE SENSORS JOURNAL, VOL. 15, NO. 8, AUGUST 2015, p. 4275-4285;

[23] https://msdn.microsoft.com/en-us/library/jj130970.aspx - Face Tracking Kinect V1;

[24] https://msdn.microsoft.com/en-us/library/dn782034.aspx - Face Tracking Kinect V2;

[25] Helena M. Reis, Lucas Wiechmann, Simone S. Borges, Isabela Gasparini, Monike Tsutsumi, Alexandre F. Brandão, Fernando A. Vasilceac, Adriana Garcia Gonçalves, Ramílio R. R. Filho, Seiji Isotani - *Rehabilitation Using Kinect and an Outlook on Its Educational Applications: A Review of the State of the Art* - Seiji Isotani, Jul 23, 2015;

[26] Peter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, Roland Siegwart - *Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling* - Advanced Robotics (ICAR), 2015 International Conference on, p. 388 - 394;

[27] https://msdn.microsoft.com/en-us/library/jj131024.aspx - Joint Filtering;

[28] https://msdn.microsoft.com/en-us/library/jj131429.aspx - Skeletal Joint Smoothing White Paper;

[29] https://social.msdn.microsoft.com/Forums/en-US/f3bc6904-d07a-43d5-8657-1ac8cd147f95/how-to-reduce-jitter-in-kinect-v20-sdk?forum=kinectv2sdk – How to reduce jitter in kinect V2 SDK;

[30] https://social.msdn.microsoft.com/Forums/en-US/ffbc8ec7-7551-4462-88aa-2fab69eac38f/joint-smoothing-code-c-errors-in-kinectjointfilter-class?forum=kinectv2sdk - Joint Smoothing code (C#);

[31] https://msdn.microsoft.com/en-us/library/jj131041.aspx – Avateering C# sample;

[32] Nikolas Trawny, Stergios I. Roumeliotis - *Indirect Kalman Filter for 3D Attitude Estimation A Tutorial for Quaternion Algebra* - Multiple Autonomous Robotic Systems Laboratory, TR-2005-002, Rev. 57, march 2005;

[33] https://social.msdn.microsoft.com/Forums/en-US/a87049b5-7842-4c17-b776-3f6f4260c801/how-to-interpret-jointorientation-data?forum=k4wv2devpreview- How to interpret JointOrientation data;

[34] https://social.msdn.microsoft.com/Forums/en-US/3885a629-8e83-413f-b995-ab8e80061d9a/kinect-v2-quaternions?forum=kinectv2sdk - Kinect v2 quaternions;

[35] https://social.msdn.microsoft.com/Forums/en-US/245a3a09-a2e4-4e0e-8c12-b8625102376a/kinect-v2-sdk-joint-orientation?forum=kinectv2sdk – Kinect v2 SDK Joint Orientation;

[36] https://social.msdn.microsoft.com/Forums/en-US/f2e6a544-705c-43ed-a0e1-731ad907b776/meaning-of-rotation-data-of-k4w-v2?forum=k4wv2devpreview – meaning of rotation data of k4w v2;

[37]https://msdn.microsoft.com/en-us/library/windowspreview.kinect.jointorientation.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1

[38] Roberto G. Valenti, Ivan Dryanovski, Jizhong Xiao - *Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs* - Sensors 2015, 15, 19302-19330; doi:10.3390/s150819302;

[39] Mark Pedley - Tilt Sensing Using a Three-Axis Accelerometer - 2007-2009, 2012-2013 Freescale Semiconductor, Inc. All rights reserved;

[40] D. Comotti , M. Ermidoro - Report Of The Course "Progetto Di Microelettronica" – università degli studi di Bergamo;

Appendice A

Viene qui riportato il codice completo del Windows Form:

```
using Microsoft.Kinect;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.IO;


namespace MyKinectForm
{
    public partial class MainForm : Form
    {

        #region Members

        private int countFrame;
        private int width;
        private int height;
        private int stride;
        private int colorIndex;
        private int depthIndex;
        private int infraredIndex;
        private int i;
        private int penIndex;
        private int _x;
        private int _y;
        private int videoGame; //case 1 video, case 2 game
        private int mano;
        private int partialScore;
        private int totalScore;
        private int perc;
        private int n;
        private int Index;
        private float parSmoothing;
        private float parCorrection;
        private float parPrediction;
        private float parJitterRadius;
        private float parMaxDeviationRadius;
        private float posX;
        private float posY;
        private float coordx;
        private float coordy;

        private bool viewBody;
        private bool viewVideo;
        private bool isRecording;
        private bool filterOn;
        private bool gameOn;
        private bool startGame;

        private string stateTrack;
        private string posJoint;
        private string posJointf;
```

```
private string savedFile;
private string savedFile2;
private string typeOfFilter;
private string giudizio;

private System.Text.StringBuilder sb = new System.Text.StringBuilder();
private System.Text.StringBuilder sb2 = new System.Text.StringBuilder();

private byte[] colorData;
private byte intensity;

private KinectSensor KinSensor;
private CoordinateMapper coordinateMapper;
private MultiSourceFrameReader KinReader;
private ColorSpacePoint colorPoint;
private DepthSpacePoint depthPoint;

private JointType jt;
private JointType jt2;
private CameraSpacePoint[] filteredJoints;
private CameraSpacePoint jointPosition;
private CameraSpacePoint filteredJoint;
private CameraSpacePoint jointPos;
private CameraSpacePoint pointBlockMs;
private CameraSpacePoint pointBLockSs;
private CameraSpacePoint pointBlock;
private CameraSpacePoint MidSpine;
private CameraSpacePoint SpineShoulder;
private CameraSpacePoint Shoulder;
private CameraSpacePoint newPoint;
private CameraSpacePoint rawPoint;
private CameraSpacePoint dirX;
private CameraSpacePoint dirY;
private CameraSpacePoint dirZ;

private IList<Body> Bodies;
private List<Pen> bodyColors;
private List<float> Points;
private Pen drawPen;
private Pen penTarget;
private Graphics g;
private IntPtr colorPtr;
private Bitmap kinectVideoBitmap;
private KinectJointFilter filter = new KinectJointFilter();

private Enum_Mode KinMode;
private Body_Mode skelMode;

private ushort minDepth;
private ushort maxDepth;
private ushort ir;
private ushort depth;
private ushort[] pixelData;
private ushort[] frameData;

DateTime startTime;
DateTime currentTime;
TimeSpan diffTime = new TimeSpan();
DateTime startMove;
DateTime currentMove;
TimeSpan diffMove = new TimeSpan();
```

```
        private double initDiffTimeForPreparation;
        private double lengthRecording;


        #endregion

        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public MainForm()
        {
            InitializeComponent();
            videoGame = 1;
            Index = 1;
            n = 1;
            typeOfFilter = "Low Smooth";
            viewBody = false;
            viewVideo = true;
            isRecording = false;
            filterOn = false;
            parSmoothing = 0.25f;
            parCorrection = 0.25f;
            parPrediction = 0.25f;
            parJitterRadius = 0.03f;
            parMaxDeviationRadius = 0.05f;
            initDiffTimeForPreparation = 2;
            lengthRecording = 30;
            g = pictureBox1.CreateGraphics();
            mano = 1;
            partialScore = 0;
            totalScore = 0;
            penTarget = new Pen(Brushes.Red, 8);
        }

        #endregion

        #region Event Handlers

        private void Reader_MultiSourceFrameArrived(object sender,
MultiSourceFrameArrivedEventArgs e)
        {
            try
            {
                var reference = e.FrameReference.AcquireFrame();
                if (viewVideo)
                {

                    #region RGB MANAGEMENT

                    if (KinMode == Enum_Mode.Color)
                    {
                        using (var frame =
reference.ColorFrameReference.AcquireFrame())
                        {

                            if (frame == null)
                            {
                                return;
                            }
```

```
                            width = frame.FrameDescription.Width;
                            height = frame.FrameDescription.Height;


                            if (colorData == null)
                            {
                                colorData = new byte[width * height * ((32 + 7) /
8)];
                            }

                            if (frame.RawColorImageFormat ==
ColorImageFormat.Bgra)
                            {
                                frame.CopyRawFrameDataToArray(colorData);
                            }
                            else
                            {
                                frame.CopyConvertedFrameDataToArray(colorData,
ColorImageFormat.Bgra);
                            }

                            Marshal.FreeHGlobal(colorPtr);
                            colorPtr = Marshal.AllocHGlobal(colorData.Length);
                            Marshal.Copy(colorData, 0, colorPtr,
colorData.Length);
                            PixelFormat format = PixelFormat.Format32bppRgb;

                            stride = width * 32 / 8;


                            kinectVideoBitmap = new Bitmap(width, height, stride,
format, colorPtr);


                            //else
                            //{
                            //    kinectVideoBitmap = new Bitmap(width, height,
format);
                            //}

                            pictureBox1.Image = kinectVideoBitmap;
                        }
                    }
                    #endregion

                    #region DEPTH MANAGEMENT

                    if (KinMode == Enum_Mode.Depth)
                    {
                        using (var frame =
reference.DepthFrameReference.AcquireFrame())
                        {

                            if (frame == null)
                            {
                                return;
                            }

                            width = frame.FrameDescription.Width;
                            height = frame.FrameDescription.Height;
```

```
                            if (colorData == null)
                            {
                                colorData = new byte[width * height * ((32 + 7) /
8)];
                            }

                            minDepth = frame.DepthMinReliableDistance;
                            maxDepth = frame.DepthMaxReliableDistance;

                            pixelData = new ushort[width * height];


                            frame.CopyFrameDataToArray(pixelData);

                            colorIndex = 0;
                            for (depthIndex = 0; depthIndex < pixelData.Length;
++depthIndex)
                            {
                                depth = pixelData[depthIndex];

                                intensity = (byte)(depth >= minDepth && depth <=
maxDepth ? depth : 0);

                                colorData[colorIndex++] = intensity; // Blue
                                colorData[colorIndex++] = intensity; // Green
                                colorData[colorIndex++] = intensity; // Red

                                ++colorIndex;
                            }



                            Marshal.FreeHGlobal(colorPtr);
                            colorPtr = Marshal.AllocHGlobal(colorData.Length);
                            Marshal.Copy(colorData, 0, colorPtr,
colorData.Length);
                            PixelFormat format = PixelFormat.Format32bppRgb;
                            stride = width * 32 / 8;


                                kinectVideoBitmap = new Bitmap(width, height,
stride, format, colorPtr);


                            //else
                            //{
                            //    kinectVideoBitmap = new Bitmap(width, height,
format);
                            //}

                            pictureBox1.Image = kinectVideoBitmap;
                        }
                    }

                    #endregion

                    #region IR MANAGEMENT

                    if (KinMode == Enum_Mode.Infrared)
                    {
                        using (var frame =
reference.InfraredFrameReference.AcquireFrame())
```

```
                        {

                            if (frame == null)
                            {
                                return;
                            }

                            width = frame.FrameDescription.Width;
                            height = frame.FrameDescription.Height;


                            if (colorData == null)
                            {
                                colorData = new byte[width * height * ((32 + 7) /
8)];
                            }

                            frameData = new ushort[width * height];


                            frame.CopyFrameDataToArray(frameData);

                            colorIndex = 0;
                            for (infraredIndex = 0; infraredIndex <
frameData.Length; infraredIndex++)
                            {
                                ir = frameData[infraredIndex];

                                intensity = (byte)(ir >> 7);
                                colorData[colorIndex++] = (byte)(intensity / 1);
// Blue
                                colorData[colorIndex++] = (byte)(intensity / 1);
// Green
                                colorData[colorIndex++] = (byte)(intensity / 1);
// Red


                                colorIndex++;
                            }

                            stride = width * 32 / 8;


                            Marshal.FreeHGlobal(colorPtr);
                            colorPtr = Marshal.AllocHGlobal(colorData.Length);
                            Marshal.Copy(colorData, 0, colorPtr,
colorData.Length);
                            PixelFormat format = PixelFormat.Format32bppRgb;


                                kinectVideoBitmap = new Bitmap(width, height,
stride, format, colorPtr);


                            //else
                            //{
                            //    kinectVideoBitmap = new Bitmap(width, height,
format);
                            //}

                            pictureBox1.Image = kinectVideoBitmap;
                        }
```

```
            }
            #endregion
        }
        # region Body

        using (var frame = reference.BodyFrameReference.AcquireFrame())
            {
                if (frame != null)
                {

                    Bodies = new Body[frame.BodyFrameSource.BodyCount];

                    frame.GetAndRefreshBodyData(Bodies);

                    penIndex = 0;

                    foreach (var body in Bodies)
                    {
                        //filter.Init();
                        drawPen = this.bodyColors[penIndex++];
                        if (body != null)
                        {
                            if (body.IsTracked)
                            {
                                if(gameOn)
                                {
                                    drawGame(body);
                                }
                                else
                                {
                                    drawSkeleton(body);
                                }
                                // Draw skeleton.


                            }
                        }
                    }
                }
            }

        #endregion

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message + ex.StackTrace);
    }

}

private void statusFunction(object sender, IsAvailableChangedEventArgs e)
{
    if (e.IsAvailable)
    {
        labelConnectionStatus.Text = "connesso";
    }
    else
    {
        labelConnectionStatus.Text = "Non Connesso";
    }
}
```

```
        private void Window_Closed(object sender, EventArgs e)
        {
            if (KinReader != null)
            {
                KinReader.Dispose();
            }

            if (KinSensor != null)
            {
                KinSensor.Close();
            }
        }

        private void Window_Loaded(object sender, EventArgs e)
        {
            try
            {
                rbRGB.Checked = true;
                KinMode = Enum_Mode.Color;

                KinSensor = KinectSensor.GetDefault();
                coordinateMapper = KinSensor.CoordinateMapper;

                // populate body colors, one for each BodyIndex
                bodyColors = new List<Pen>();

                bodyColors.Add(new Pen(Brushes.Red, 4));
                bodyColors.Add(new Pen(Brushes.Orange, 4));
                bodyColors.Add(new Pen(Brushes.Green, 4));
                bodyColors.Add(new Pen(Brushes.Blue, 4));
                bodyColors.Add(new Pen(Brushes.Indigo, 4));
                bodyColors.Add(new Pen(Brushes.Violet, 4));

                KinSensor.IsAvailableChanged += new
EventHandler<IsAvailableChangedEventArgs>(statusFunction);

                if (KinSensor == null)
                {
                    return;
                }

                KinSensor.Open();
                KinReader =
KinSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Color |
FrameSourceTypes.Depth | FrameSourceTypes.Infrared | FrameSourceTypes.Body);

                // connect up the video event handler

                KinReader.MultiSourceFrameArrived +=
Reader_MultiSourceFrameArrived;
            }
            catch
            {
                MessageBox.Show("Inizializzazione fallita", "Camera viewer");
                Application.Exit();
            }
        }

        #endregion

        #region View mode
```

```
private void mnRGBView_CheckedChanged(object sender, EventArgs e)
{
    viewVideo = mnVideoView.Checked;
    gameOn = false;
    buttonREC.Text = "START";
    buttonSTOP.Text = "STOP";
    videoGame = 1;
    groupBox1.Text = "Recording:";
}

private void mnBodyView_CheckedChanged(object sender, EventArgs e)
{
    viewBody = mnBodyView.Checked;
    gameOn = false;
    buttonREC.Text = "START";
    buttonSTOP.Text = "STOP";
    groupBox1.Text = "Recording:";
    videoGame = 1;
}

private void rbRGB_CheckedChanged(object sender, EventArgs e)
{
    KinMode = Enum_Mode.Color;
}

private void rbDEPTH_CheckedChanged(object sender, EventArgs e)
{
    KinMode = Enum_Mode.Depth;
}

private void rbINFRARED_CheckedChanged(object sender, EventArgs e)
{
    KinMode = Enum_Mode.Infrared;
}

private void rbFull_CheckedChanged(object sender, EventArgs e)
{
    skelMode = Body_Mode.Full;
}

private void rbSeated_CheckedChanged(object sender, EventArgs e)
{
    skelMode = Body_Mode.Seated;
}

#endregion

#region  drawing

private void drawGame(Body body)
{
    g.Clear(Color.BlanchedAlmond);

    jt = JointType.HandRight;
    jt2 = JointType.ShoulderRight;

    if (mano == 2)
    {
        jt = JointType.HandLeft;
        jt2 = JointType.ShoulderLeft;
    }
```

```
            MidSpine = body.Joints[JointType.SpineMid].Position;
            SpineShoulder = body.Joints[JointType.SpineShoulder].Position;
            Shoulder = body.Joints[jt2].Position;
            jointPos = body.Joints[jt].Position;


            if (filterOn)
            {
                filter.UpdateFilter(body);
                filteredJoint = filter.GetFilteredJoint((int)jt);
                MidSpine = filter.GetFilteredJoint((int)JointType.SpineMid);
                SpineShoulder =
filter.GetFilteredJoint((int)JointType.SpineShoulder);
                Shoulder = filter.GetFilteredJoint((int)jt2);
            }

            if (startGame)
            {
                Shoulder = pointBlock;
                MidSpine = pointBlockMs;
                SpineShoulder = pointBLockSs;
            }


            //depthPoint =
KinSensor.CoordinateMapper.MapCameraPointToDepthSpace(joint.Position);
            newPoint = filter.CSVectorSubtract(Shoulder, jointPos);
            dirX = filter.CSVectorDiv(filter.CSVectorSub(Shoulder, SpineShoulder),
filter.CSVectorLength(filter.CSVectorSub(Shoulder, SpineShoulder)));
            dirZ =
filter.CSVectorDiv(filter.CSVectorCross(filter.CSVectorSub(MidSpine,
SpineShoulder), dirX),
filter.CSVectorLength(filter.CSVectorCross(filter.CSVectorSub(MidSpine,
SpineShoulder), dirX)));
            dirY = filter.CSVectorCross(dirZ, dirX);
            rawPoint = jointPos;

            if (filterOn)
            {
                newPoint = filter.CSVectorSubtract(Shoulder, filteredJoint);
                jointPos = filteredJoint;
            }

            posX = -filter.CSVectorDot(dirX, newPoint);
            posY = filter.CSVectorDot(dirY, newPoint);
            coordx = (int)(((posX * 960) / 0.8f) + (960 / 2));
            coordy = (int)(((posY * 540) / 0.45f) + (540 / 2));

            g.FillEllipse(Brushes.Green, coordx + 25, coordy + 25, 50, 50);
            g.DrawEllipse(penTarget, _x, _y, 100, 100);



            if (Math.Sqrt(Math.Pow((_x - coordx), 2) + Math.Pow((_y - coordy), 2))
< 15 && startGame == false) //distanza minore di tot
            {
                pointBlock = Shoulder;
                pointBlockMs = MidSpine;
                pointBLockSs = SpineShoulder;
                startGame = true;
```

```
                timerMove.Enabled = true;
                sb.Clear();

                startMove = DateTime.Now;
                //inizializzo tutto
            }
            if (startGame)
            {
                gameSetup(_x, _y, coordx, coordy);
                posJoint = rawPoint.X.ToString() + " " + rawPoint.Y.ToString() + "
" + rawPoint.Z.ToString() + " " + jointPos.X.ToString() + " " +
jointPos.Y.ToString() + " " + jointPos.Z.ToString();
                sb.AppendLine(posJoint);
            }
        }



        private void drawSkeleton(Body body)
        {

            Points = new List<float>();
            if (viewBody)
            {


                filter.UpdateFilter(body);

                filteredJoints=filter.GetFilteredJoints();
                i=0;

                foreach (Joint joint in body.Joints.Values)
                {

                    // 3D space point
                    jointPosition = joint.Position;

                    // 2D space point
                    //Point point = new Point();

                    if (KinMode == Enum_Mode.Color)
                    {
                        colorPoint
=KinSensor.CoordinateMapper.MapCameraPointToColorSpace(jointPosition);

                        posX = colorPoint.X;
                        posY = colorPoint.Y;
                        coordx = posX / 2;
                        coordy = posY / 2;
                    }
                    else // Change the Image and Canvas dimensions to 512x424
                    {
                        depthPoint =
KinSensor.CoordinateMapper.MapCameraPointToDepthSpace(jointPosition);

                        posX =depthPoint.X;
                        posY =depthPoint.Y;
                        coordx = (posX * 960) / 512;
                        coordy = (posY * 540) / 424;

                    }
```

```
                    stateTrack =countFrame.ToString()+"
"+joint.JointType.ToString()+"  "+ joint.TrackingState.ToString() + "  non
filtrato";
                    posJoint = joint.Position.X.ToString() + " " +
joint.Position.Y.ToString() + " " + joint.Position.Z.ToString() + "
"+joint.TrackingState.ToString()+" 0 0 0";

                    if (filterOn)
                    {

                        if (KinMode == Enum_Mode.Color)
                        {
                            colorPoint=
KinSensor.CoordinateMapper.MapCameraPointToColorSpace(filteredJoints[i]);

                            posX = colorPoint.X;
                            posY = colorPoint.Y;
                            coordx = posX / 2;
                            coordy = posY / 2;
                        }
                        else // Change the Image and Canvas dimensions to 512x424
                        {
                            depthPoint =
KinSensor.CoordinateMapper.MapCameraPointToDepthSpace(filteredJoints[i]);

                            posX = depthPoint.X;
                            posY = depthPoint.Y;
                            coordx = (posX * 960) / 512;
                            coordy = (posY * 540) / 424;

                        }

                        posJointf = filteredJoints[i].X.ToString() + " " +
filteredJoints[i].Y.ToString() + " " + filteredJoints[i].Z.ToString();
                        posJoint=posJoint.Replace("0 0 0", posJointf);
                        stateTrack = stateTrack.Replace("  non filtrato", "
Filtro di tipo:  " + typeOfFilter);
                        i++;
                    }
                    posJoint = posJoint.Replace("Inferred", "0");
                    posJoint = posJoint.Replace("Tracked", "1");
                    Points.Add(coordx);
                    Points.Add(coordy);
                    if(isRecording)
                    {
                        sb.AppendLine(posJoint);
                        sb2.AppendLine(stateTrack);
                    }

                }
                if (isRecording)
                {
                    countFrame = countFrame + 1;
                }

                g.DrawEllipse(drawPen, Points[4], Points[5], 10, 10);
                g.DrawEllipse(drawPen, Points[6], Points[7], 10, 10);
                g.DrawEllipse(drawPen, Points[8], Points[9], 10, 10);
                g.DrawEllipse(drawPen, Points[10], Points[11], 10, 10);
                g.DrawEllipse(drawPen, Points[12], Points[13], 10, 10);
                g.DrawEllipse(drawPen, Points[14], Points[15], 10, 10);
```

```
                g.DrawEllipse(drawPen, Points[16], Points[17], 10, 10);
                g.DrawEllipse(drawPen, Points[18], Points[19], 10, 10);
                g.DrawEllipse(drawPen, Points[20], Points[21], 10, 10);
                g.DrawEllipse(drawPen, Points[22], Points[23], 10, 10);
                g.DrawEllipse(drawPen, Points[40], Points[41], 10, 10);
                g.DrawEllipse(drawPen, Points[42], Points[43], 10, 10);
                g.DrawEllipse(drawPen, Points[44], Points[45], 10, 10);
                g.DrawEllipse(drawPen, Points[46], Points[47], 10, 10);
                g.DrawEllipse(drawPen, Points[48], Points[49], 10, 10);

                g.DrawLine(drawPen, Points[6], Points[7], Points[4], Points[5]);
                g.DrawLine(drawPen, Points[4], Points[5], Points[40], Points[41]);
                g.DrawLine(drawPen, Points[40], Points[41], Points[8], Points[9]);
                g.DrawLine(drawPen, Points[40], Points[41], Points[16],
Points[17]);
                g.DrawLine(drawPen, Points[8], Points[9], Points[10], Points[11]);
                g.DrawLine(drawPen, Points[16], Points[17], Points[18],
Points[19]);
                g.DrawLine(drawPen, Points[10], Points[11], Points[12],
Points[13]);
                g.DrawLine(drawPen, Points[18], Points[19], Points[20],
Points[21]);
                g.DrawLine(drawPen, Points[12], Points[13], Points[14],
Points[15]);
                g.DrawLine(drawPen, Points[20], Points[21], Points[22],
Points[23]);
                g.DrawLine(drawPen, Points[14], Points[15], Points[42],
Points[43]);
                g.DrawLine(drawPen, Points[22], Points[23], Points[46],
Points[47]);
                g.DrawLine(drawPen, Points[14], Points[15], Points[44],
Points[45]);
                g.DrawLine(drawPen, Points[22], Points[23], Points[48],
Points[49]);

                if (skelMode == Body_Mode.Full)
                {
                    g.DrawEllipse(drawPen, Points[0], Points[1], 10, 10);
                    g.DrawEllipse(drawPen, Points[2], Points[3], 10, 10);
                    g.DrawEllipse(drawPen, Points[24], Points[25], 10, 10);
                    g.DrawEllipse(drawPen, Points[26], Points[27], 10, 10);
                    g.DrawEllipse(drawPen, Points[28], Points[29], 10, 10);
                    g.DrawEllipse(drawPen, Points[30], Points[31], 10, 10);
                    g.DrawEllipse(drawPen, Points[32], Points[33], 10, 10);
                    g.DrawEllipse(drawPen, Points[34], Points[35], 10, 10);
                    g.DrawEllipse(drawPen, Points[36], Points[37], 10, 10);
                    g.DrawEllipse(drawPen, Points[38], Points[39], 10, 10);
                    g.DrawLine(drawPen, Points[40], Points[41], Points[2],
Points[3]);
                    g.DrawLine(drawPen, Points[2], Points[3], Points[0],
Points[1]);
                    g.DrawLine(drawPen, Points[0], Points[1], Points[24],
Points[25]);
                    g.DrawLine(drawPen, Points[0], Points[1], Points[32],
Points[33]);
                    g.DrawLine(drawPen, Points[24], Points[25], Points[26],
Points[27]);
                    g.DrawLine(drawPen, Points[32], Points[33], Points[34],
Points[35]);
                    g.DrawLine(drawPen, Points[26], Points[27], Points[28],
Points[29]);
```

```
                    g.DrawLine(drawPen, Points[34], Points[35], Points[36],
Points[37]);
                    g.DrawLine(drawPen, Points[28], Points[29], Points[30],
Points[31]);
                    g.DrawLine(drawPen, Points[36], Points[37], Points[38],
Points[39]);
                }

            }
        }

        #endregion

        #region Game

        private void buttonGame_Click(object sender, EventArgs e)
        {

            pictureBox1.Image = null;
            mnVideoView.Checked = false;
            viewBody = false;
            mnBodyView.Checked = false;
            viewVideo = false;
            timerMove.Enabled = false;
            buttonREC.Text = "Sinistra";
            buttonSTOP.Text = "Destra";
            groupBox1.Text = "Mano:";
            videoGame = 2;
            partialScore = 0;
            totalScore = 0;
            penTarget = new Pen(Brushes.Red, 8);
            _x = 200;
            _y = 250;
            i = 3;
            g.DrawEllipse(penTarget, _x, _y, 100, 100);
            gameOn = true;
        }

        private void gameSetup(int _x, int _y, float coordx, float coordy)
        {
            if (Math.Sqrt(Math.Pow((_x - coordx), 2) + Math.Pow((_y - coordy), 2))
< 35)
            {
                penTarget = new Pen(Brushes.Green, 8);
                partialScore += 100;

            }
            else if (Math.Sqrt(Math.Pow((_x - coordx), 2) + Math.Pow((_y -
coordy), 2)) > 75)
            {
                penTarget = new Pen(Brushes.Red, 8);

            }
            else
            {
                penTarget = new Pen(Brushes.Yellow, 8);
                partialScore += 50;
            }

            totalScore += 100;

        }
```

```
        private void timerMove_Tick(object sender, EventArgs e)
        {
            currentMove = DateTime.Now;
            diffMove = currentMove - startMove;
            if (diffMove.TotalSeconds < lengthRecording)
            {
                _x += i;


                _y = (int)(250 + Math.Sin(((_x + 250) * Math.PI * 2) / 450) * (400
/ i));

                if (_x >= 650)
                {
                    i = -3;
                }
                if (_x <= 200)
                {
                    i = 3;
                }

                //muovo a destra o sinistra
            }
            else
            {

                timerMove.Enabled = false;
                penTarget = new Pen(Brushes.Red, 8);
                startGame = false;
                _x = 200;
                _y = 250;
                g.DrawEllipse(penTarget, _x, _y, 100, 100);
                perc = (partialScore * 100) / totalScore;
                int range = perc / 25;
                switch (range)
                {
                    case 4:
                        giudizio = "Perfetto!!";
                        break;
                    case 3:
                        giudizio = "Ottimo!";
                        break;
                    case 2:
                        giudizio = "ancora un piccolo sforzo..";
                        break;
                    default:
                        giudizio = "Andrà meglio...";
                        break;

                }

                MessageBox.Show("HAI REALIZZATO: " + partialScore.ToString() + "
PUNTI SU " + totalScore.ToString() + " ( " + perc.ToString() + "% ) , " +
giudizio, "Punteggio");
                SaveRecordedData();
                Index++;
            }
        }

        #endregion
```

```
        #region Recording

        private void buttonREC_Click(object sender, EventArgs e)
        {
            switch (videoGame)
            {
                case 1:
                    sb.Clear();
                    sb2.Clear();
                    tbCent.Text = "00";
                    tbSec.Text = "00";
                    tbMin.Text = "00";
                    startTime = DateTime.Now;
                    timeRec.Enabled = true;
                    countFrame = 1;
                    break;
                case 2:
                    mano = 2;
                    break;
            }

        }

        private void buttonSTOP_Click(object sender, EventArgs e)
        {
            switch (videoGame)
            {
                case 1:
                    isRecording = false;
                    timeRec.Enabled = false;
                    SaveRecordedData();
                    break;
                case 2:
                    mano = 1;
                    break;
            }

        }

        private void SaveRecordedData()
        {
            try
            {
                if (sb.Length != 0)
                {

                    saveFD.InitialDirectory =
System.Environment.GetFolderPath(Environment.SpecialFolder.Personal);
                    saveFD.Title = "Save a Text File";
                    saveFD.FileName = "Nuovo";
                    saveFD.Filter = "Text File:|*.txt|All:|*.*";

                    if (saveFD.ShowDialog() ==
System.Windows.Forms.DialogResult.OK && saveFD.FileName.Length > 0)
                    {
                        savedFile = saveFD.FileName;
                        savedFile2 = saveFD.FileName.Replace(".", "Status.");
                        StreamWriter obj = new StreamWriter(savedFile, true);
                        obj.Write(sb.ToString());
                        obj.Close();
                        StreamWriter obj2 = new StreamWriter(savedFile2);
                        obj2.Write(sb2.ToString());
```

```
                    obj2.Close();

                }

            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message + ex.StackTrace);
        }
    }

    private void timeRec_Tick(object sender, EventArgs e)
    {

        currentTime = DateTime.Now;
        diffTime = currentTime - startTime;
        tbMin.Text = diffTime.Minutes.ToString("00");
        tbSec.Text = diffTime.Seconds.ToString("00");
        tbCent.Text = ((diffTime.Milliseconds)/100.0).ToString("0");
        if (diffTime.TotalSeconds > initDiffTimeForPreparation)
        {

            isRecording = true;

        }
        if (diffTime.TotalSeconds >
(initDiffTimeForPreparation+lengthRecording))
        {
            isRecording = false;
            timeRec.Enabled = false;

            SaveRecordedData();

        }
    }

    #endregion

    #region Filtering

    private void buttonFilter_Click(object sender, EventArgs e)
    {
        if(filterOn==false)
        {
            filterOn = !filterOn;
            buttonFilter.Text="TURN OFF";
            filter = new KinectJointFilter();
        }
        else
        {
            filterOn = !filterOn;
            buttonFilter.Text = "TURN ON";
            filter.Init(parSmoothing, parCorrection, parPrediction,
parJitterRadius, parMaxDeviationRadius);
        }
    }

    private void rbLow_CheckedChanged(object sender, EventArgs e)
    {
        parSmoothing = 0.25f;
        parCorrection = 0.25f;
```

```
            parPrediction = 0.25f;
            parJitterRadius = 0.03f;
            parMaxDeviationRadius = 0.05f;
            typeOfFilter = rbLow.Text;
            filter.Init(parSmoothing, parCorrection, parPrediction,
parJitterRadius, parMaxDeviationRadius);
        }

        private void rbMiddle_CheckedChanged(object sender, EventArgs e)
        {
            parSmoothing = 0.5f;
            parCorrection = 0.5f;
            parPrediction = 0.5f;
            parJitterRadius = 0.05f;
            parMaxDeviationRadius = 0.04f;
            typeOfFilter = rbMiddle.Text;
            filter.Init(parSmoothing, parCorrection, parPrediction,
parJitterRadius, parMaxDeviationRadius);
        }

        private void rbHigh_CheckedChanged(object sender, EventArgs e)
        {
            parSmoothing = 0.5f;
            parCorrection = 0.1f;
            parPrediction = 0.5f;
            parJitterRadius = 0.1f;
            parMaxDeviationRadius = 0.1f;
            typeOfFilter = rbHigh.Text;
            filter.Init(parSmoothing, parCorrection, parPrediction,
parJitterRadius, parMaxDeviationRadius);
        }

        private void rbVeryHigh_CheckedChanged(object sender, EventArgs e)
        {
            parSmoothing = 0.7f;
            parCorrection = 0.3f;
            parPrediction = 1.0f;
            parJitterRadius = 1.0f;
            parMaxDeviationRadius = 1.0f;
            filter.Init(parSmoothing, parCorrection, parPrediction,
parJitterRadius, parMaxDeviationRadius);
            typeOfFilter = rbVeryHigh.Text;
        }


        #endregion


    }
}
```

Per la parte di codice riguardante l'utilizzo del Kinect, delle coordinate Mapper e del body tracking si ringrazia Vangos Pterneas [http://pterneas.com/] che sul suo sito ha gentilmente fornito le linee guida per programmare in c# con il Kinect.

Appendice B

Viene qui riportato il codice completo della GUI di matlab:

```
function varargout = openKinMatlab(varargin)
% OPENKINMATLAB MATLAB code for openKinMatlab.fig
%      OPENKINMATLAB, by itself, creates a new OPENKINMATLAB or raises
the existing
%      singleton*.
%
%      H = OPENKINMATLAB returns the handle to a new OPENKINMATLAB or
the handle to
%      the existing singleton*.
%
%      OPENKINMATLAB('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in OPENKINMATLAB.M with the given input
arguments.
%
%      OPENKINMATLAB('Property','Value',...) creates a new OPENKINMATLAB
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before openKinMatlab_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to openKinMatlab_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help openKinMatlab

% Last Modified by GUIDE v2.5 09-Dec-2015 15:53:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @openKinMatlab_OpeningFcn, ...
                   'gui_OutputFcn',  @openKinMatlab_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
% End initialization code - DO NOT EDIT


% --- Executes just before openKinMatlab is made visible.
function openKinMatlab_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to openKinMatlab (see VARARGIN)

% Choose default command line output for openKinMatlab
handles.output = hObject;

handles.isTracked=[];
handles.x=[];
handles.devx=[];
handles.y=[];
handles.devy=[];
handles.z=[];
handles.devz=[];
handles.s=1;
handles.fullBody=-1;
handles.d=[];
handles.df=[];
handles.v=[];
handles.vf=[];
handles.a=[];
handles.af=[];
handles.Tag=char('spineBase','midSpine','neck','head','leftShoulder','le
ftElbow','leftWrist','leftHand','rightShoulder','rightElbow','rightWrist
','rightHand','leftHip','leftKnee','leftAnkle','leftFoot','rightHip','ri
ghtKnee','rightAnkle','rightFoot','spineShoulder','LeftHandTip','leftThu
mb','rightHandTip','rightThumb');
handles.Joint=0;
set(handles.sliderFrames,'Enable','off');
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes openKinMatlab wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = openKinMatlab_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function sliderFrames_Callback(hObject, eventdata, handles)
% hObject    handle to sliderFrames (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
    i=get(handles.sliderFrames,'Value');
    i=uint16(i);
    x=handles.x;
    y=handles.y;
    z=handles.z;
    xh=[x(:,3) x(:,4) x(:,5) x(:,6) x(:,7) x(:,8) x(:,9) x(:,10) x(:,11)
x(:,12) x(:,21) x(:,22) x(:,23) x(:,24) x(:,25)];
    yh=[y(:,3) y(:,4) y(:,5) y(:,6) y(:,7) y(:,8) y(:,9) y(:,10) y(:,11)
y(:,12) y(:,21) y(:,22) y(:,23) y(:,24) y(:,25)];
    zh=[z(:,3) z(:,4) z(:,5) z(:,6) z(:,7) z(:,8) z(:,9) z(:,10) z(:,11)
z(:,12) z(:,21) z(:,22) z(:,23) z(:,24) z(:,25)];

    switch handles.s
        case 1

            x1=[x(:,22) x(:,8) x(:,7) x(:,6) x(:,5) x(:,21) x(:,9)
x(:,10) x(:,11) x(:,12) x(:,24)];
            x2=[x(:,12) x(:,25)];
            x3=[x(:,8) x(:,23)];
            x4=[x(:,21) x(:,3) x(:,4)];
            y1=[y(:,22) y(:,8) y(:,7) y(:,6) y(:,5) y(:,21) y(:,9)
y(:,10) y(:,11) y(:,12) y(:,24)];
            y2=[y(:,12) y(:,25)];
            y3=[y(:,8) y(:,23)];
            y4=[y(:,21) y(:,3) y(:,4)];

            if(handles.fullBody==1)

                x5=[x(:,21) x(:,2) x(:,1) x(:,13) x(:,14) x(:,15)
x(:,16)];
                x6=[x(:,1) x(:,17) x(:,18) x(:,19) x(:,20)];
                y5=[y(:,21) y(:,2) y(:,1) y(:,13) y(:,14) y(:,15)
y(:,16)];
                y6=[y(:,1) y(:,17) y(:,18) y(:,19) y(:,20)];
                axis(handles.axes1)
                plot(x(i,:),y(i,:),'ro')
                set(handles.editFrames,'String',int2str(i));
                hold on
                xlim([-1 1]);
                ylim([-1 1]);
                xlabel('X')
                ylabel('Y')
                grid on

plot(x1(i,:)',y1(i,:)',x2(i,:)',y2(i,:)',x3(i,:)',y3(i,:)',x4(i,:)',y4(i
,:)',x5(i,:)',y5(i,:)',x6(i,:)',y6(i,:)')
                hold off
            else

                axis(handles.axes1)
                plot(xh(i,:),yh(i,:),'ro')
                set(handles.editFrames,'String',int2str(i));
                hold on
                xlim([-1 1]);
                ylim([-1 1]);
```

```
                xlabel('X')
                ylabel('Y')
                grid on

plot(x1(i,:)',y1(i,:)',x2(i,:)',y2(i,:)',x3(i,:)',y3(i,:)',x4(i,:)',y4(i
,:)')
                hold off

            end

        case 2

            x1=[x(:,22) x(:,8) x(:,7) x(:,6) x(:,5) x(:,21) x(:,9)
x(:,10) x(:,11) x(:,12) x(:,24)];
            x2=[x(:,12) x(:,25)];
            x3=[x(:,8) x(:,23)];
            x4=[x(:,21) x(:,3) x(:,4)];
            y1=[y(:,22) y(:,8) y(:,7) y(:,6) y(:,5) y(:,21) y(:,9)
y(:,10) y(:,11) y(:,12) y(:,24)];
            y2=[y(:,12) y(:,25)];
            y3=[y(:,8) y(:,23)];
            y4=[y(:,21) y(:,3) y(:,4)];
            z1=[z(:,22) z(:,8) z(:,7) z(:,6) z(:,5) z(:,21) z(:,9)
z(:,10) z(:,11) z(:,12) z(:,24)];
            z2=[z(:,12) z(:,25)];
            z3=[z(:,8) z(:,23)];
            z4=[z(:,21) z(:,3) z(:,4)];

            if(handles.fullBody==1)

                x5=[x(:,21) x(:,2) x(:,1) x(:,13) x(:,14) x(:,15)
x(:,16)];
                x6=[x(:,1) x(:,17) x(:,18) x(:,19) x(:,20)];
                y5=[y(:,21) y(:,2) y(:,1) y(:,13) y(:,14) y(:,15)
y(:,16)];
                y6=[y(:,1) y(:,17) y(:,18) y(:,19) y(:,20)];
                z5=[z(:,21) z(:,2) z(:,1) z(:,13) z(:,14) z(:,15)
z(:,16)];
                z6=[z(:,1) z(:,17) z(:,18) z(:,19) z(:,20)];
                axis(handles.axes1)
                plot3(x(i,:),z(i,:),y(i,:),'ro')
                axis equal
                set(handles.editFrames,'String',int2str(i));
                hold on
%                 xlim([-0.8 0.8]);
%                 ylim([0.5 2]);
%                 zlim([-1 1]);
                xlabel('X')
                ylabel('Z')
                zlabel('Y')
                grid on

plot3(x1(i,:)',z1(i,:)',y1(i,:)',x2(i,:)',z2(i,:)',y2(i,:)',x3(i,:)',z3(
i,:)',y3(i,:)',x4(i,:)',z4(i,:)',y4(i,:)',x5(i,:)',z5(i,:)',y5(i,:)',x6(
i,:)',z6(i,:)',y6(i,:)')
                hold off

            else

                axis(handles.axes1)
```

```
                plot3(xh(i,:),zh(i,:),yh(i,:),'ro')
                set(handles.editFrames,'String',int2str(i));
                hold on
%                 xlim([-1 1]);
%                 ylim([0.5 2]);
%                 zlim([-1 1]);
                xlabel('X')
                ylabel('Z')
                zlabel('Y')
                grid on

plot3(x1(i,:)',z1(i,:)',y1(i,:)',x2(i,:)',z2(i,:)',y2(i,:)',x3(i,:)',z3(
i,:)',y3(i,:)',x4(i,:)',z4(i,:)',y4(i,:)')
                hold off

            end
    end

% --- Executes during object creation, after setting all properties.
function sliderFrames_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderFrames (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on button press in pushbutton2D.
function pushbutton2D_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2D (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles.s=1;
    guidata(hObject, handles);

function editFrames_Callback(hObject, eventdata, handles)
% hObject    handle to editFrames (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editFrames as text
%        str2double(get(hObject,'String')) returns contents of
editFrames as a double



% --- Executes during object creation, after setting all properties.
function editFrames_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editFrames (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbuttonViewJ.
function pushbuttonViewJ_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonViewJ (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%viewer=get(get(handles.buttongroupJoint,'SelectedObject'),'String');
    i=handles.Joint;
    if i~=0
        d=handles.d;
        df=handles.df;
        v=handles.v;
        vf=handles.vf;
        a=handles.a;
        af=handles.af;
        Tag=handles.Tag;
        isTracked=handles.isTracked;

        Frames=[1:size(d,1)];
        Frames2=[1:size(v,1)];
        Frames3=[1:size(a,1)];

        figure
        subplot(311)
        plot(Frames,d(:,i),Frames,df(:,i),'r')
        grid on
        xlim([1 size(d,1)])
        ylabel('Distanza Joint')
        xlabel('Frames')
        title(Tag(i,:))

        subplot(312)
        plot(Frames2,v(:,i),Frames2,vf(:,i),'r')
        ylabel('Velocità')
        xlabel('Frames')
        grid on
        xlim([1 size(d,1)])

        subplot(313)
        plot(Frames3,a(:,i),Frames3,af(:,i),'r')
        ylabel('Accellerazione')
        xlabel('Frames')
        grid on
        xlim([1 size(d,1)])

        figure
        plot(Frames,isTracked(:,i))
        grid on
        xlim([0 length(Frames)+1])
        ylim([-1 2])
        ylabel('T')
        xlabel('Frames')
        legend('1 tracked, 0 inferred')
        count=length(isTracked(:,i))-sum(isTracked(:,i));
        count=num2str(count);
```

```
        title(count)

    end
  guidata(hObject, handles);

% --- Executes on button press in pushbutton3D.
function pushbutton3D_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3D (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.s=2;
guidata(hObject, handles);

% --- Executes on button press in pushbuttonRef.
function pushbuttonRef_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonRef (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axis(handles.axes1)
imshow('bodyref.png')

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axes1
axis(hObject)
imshow('bodyref.png')


function uipushtool2_ClickedCallback(hObject, eventdata, handles)
% hObject    handle to uipushtool2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    [FileName,PathName]=uigetfile('*.txt','Browse...');
    file=fullfile(PathName,FileName);


    A = txt2mat(file);
    l=25;
    Tc=1/30;
    for i=1:l

        isTracked(:,i)=A(i:25:end-25+i,4);
        x(:,i)=A(i:25:end-25+i,1);
        y(:,i)=A(i:25:end-25+i,2);
        z(:,i)=A(i:25:end-25+i,3);
        xf(:,i)=A(i:25:end-25+i,5);
        yf(:,i)=A(i:25:end-25+i,6);
        zf(:,i)=A(i:25:end-25+i,7);
        d(:,i)=sqrt(x(:,i).^2+y(:,i).^2+z(:,i).^2);
        df(:,i)=sqrt(xf(:,i).^2+yf(:,i).^2+zf(:,i).^2);


    end
    devx=std(x,0,1);
```

```
    devy=std(y,0,1);
    devz=std(z,0,1);
    v=(d(2:end,:)-d(1:end-1,:))/Tc;
    vf=(df(2:end,:)-df(1:end-1,:))/Tc;
    af=(vf(2:end,:)-vf(1:end-1,:))/Tc;
    a=(v(2:end,:)-v(1:end-1,:))/Tc;
    handles.x=x;
    handles.y=y;
    handles.z=z;
    handles.devx=devx;
    handles.devy=devy;
    handles.devz=devz;
    handles.d=d;
    handles.df=df;
    handles.v=v;
    handles.vf=vf;
    handles.a=a;
    handles.af=af;
    handles.Joint=4;
    handles.isTracked=isTracked;
    lunghezza=size(d,1);
    set(handles.sliderFrames,'Value',1.0);
    set(handles.sliderFrames,'max',lunghezza);
    set(handles.sliderFrames,'min',1.0);
    set(handles.sliderFrames, 'SliderStep', [1/(lunghezza-1) ,
10/(lunghezza-1) ]);
    set(handles.sliderFrames,'Enable','on');
    guidata(hObject, handles);


% --- Executes on button press in radiobuttonHead.
function radiobuttonHead_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonHead (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=4;
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of radiobuttonHead


% --- Executes on button press in radiobuttonNeck.
function radiobuttonNeck_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonNeck (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=3;
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of radiobuttonNeck


% --- Executes on button press in radiobuttonSS.
function radiobuttonSS_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonSS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=21;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonSS
guidata(hObject, handles)

% --- Executes on button press in radiobuttonMS.
```

```
function radiobuttonMS_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonMS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=2;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonMS
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLS.
function radiobuttonLS_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=5;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLS
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRS.
function radiobuttonRS_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=9;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRS
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLE.
function radiobuttonLE_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=6;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLE
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRE.
function radiobuttonRE_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=10;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRE
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLW.
function radiobuttonLW_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=7;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLW
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRW.
function radiobuttonRW_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=11;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRW
```

```
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLHand.
function radiobuttonLHand_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLHand (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=8;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLHand
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRHand.
function radiobuttonRHand_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRHand (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=12;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRHand
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLHT.
function radiobuttonLHT_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLHT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=22;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLHT
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRHT.
function radiobuttonRHT_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRHT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=24;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRHT
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLT.
function radiobuttonLT_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=23;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLT
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRT.
function radiobuttonRT_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=25;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRT
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLHip.
function radiobuttonLHip_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLHip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=13;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLHip
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRHip.
function radiobuttonRHip_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRHip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=17;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRHip
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLK.
function radiobuttonLK_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLK (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=14;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLK
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRK.
function radiobuttonRK_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRK (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=18;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRK
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLA.
function radiobuttonLA_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=15;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLA
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRA.
function radiobuttonRA_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=19;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRA
guidata(hObject, handles)

% --- Executes on button press in radiobuttonLF.
function radiobuttonLF_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonLF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=16;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonLF
guidata(hObject, handles)

% --- Executes on button press in radiobuttonRF.
```

```
function radiobuttonRF_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonRF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=20;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonRF
guidata(hObject, handles)

% --- Executes on button press in radiobuttonSB.
function radiobuttonSB_Callback(hObject, eventdata, handles)
% hObject    handle to radiobuttonSB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Joint=1;
% Hint: get(hObject,'Value') returns toggle state of radiobuttonSB
guidata(hObject, handles)

% --- Executes during object creation, after setting all properties.
function radiobuttonHead_CreateFcn(hObject, eventdata, handles)
% hObject    handle to radiobuttonSS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% --- Executes during object creation, after setting all properties.
function radiobuttonSS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to radiobuttonSS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to radiobuttonSS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.fullBody=handles.fullBody*-1;
% Hint: get(hObject,'Value') returns toggle state of checkbox1
guidata(hObject, handles)


% --- Executes on button press in pushbuttonTrack.
function pushbuttonTrack_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonTrack (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Tracked=handles.isTracked;
tag=handles.Tag;
devx=handles.devx;
devx=devx*1000;
devy=handles.devy;
devy=devy*1000;
```

```
devz=handles.devz;
devz=devz*1000;
if(~isempty(Tracked))

   for i=1:25
      count(i)=length(Tracked(:,i))-sum(Tracked(:,i));

   end
    msgbox({[tag(1,:) num2str(count(1)),' stdx ',num2str(devx(1)),' stdy
',num2str(devy(1)),' stdz ',num2str(devz(1))]
        [tag(2,:)  num2str(count(2)),' stdx ',num2str(devx(2)),' stdy
',num2str(devy(2)),' stdz ',num2str(devz(2))]
        [tag(3,:)  num2str(count(3)),' stdx ',num2str(devx(3)),' stdy
',num2str(devy(3)),' stdz ',num2str(devz(3))]
        [tag(4,:)  num2str(count(4)),' stdx ',num2str(devx(4)),' stdy
',num2str(devy(4)),' stdz ',num2str(devz(4))]
        [tag(5,:)  num2str(count(5)),' stdx ',num2str(devx(5)),' stdy
',num2str(devy(5)),' stdz ',num2str(devz(5))]
        [tag(6,:)  num2str(count(6)),' stdx ',num2str(devx(6)),' stdy
',num2str(devy(6)),' stdz ',num2str(devz(6))]
        [tag(7,:)  num2str(count(7)),' stdx ',num2str(devx(7)),' stdy
',num2str(devy(7)),' stdz ',num2str(devz(7))]
        [tag(8,:)  num2str(count(8)),' stdx ',num2str(devx(8)),' stdy
',num2str(devy(8)),' stdz ',num2str(devz(8))]
        [tag(9,:)  num2str(count(9)),' stdx ',num2str(devx(9)),' stdy
',num2str(devy(9)),' stdz ',num2str(devz(9))]
        [tag(10,:)  num2str(count(10)),' stdx ',num2str(devx(10)),' stdy
',num2str(devy(10)),' stdz ',num2str(devz(10))]
        [tag(11,:)  num2str(count(11)),' stdx ',num2str(devx(11)),' stdy
',num2str(devy(11)),' stdz ',num2str(devz(11))]
        [tag(12,:)  num2str(count(12)),' stdx ',num2str(devx(12)),' stdy
',num2str(devy(12)),' stdz ',num2str(devz(12))]
        [tag(13,:)  num2str(count(13)),' stdx ',num2str(devx(13)),' stdy
',num2str(devy(13)),' stdz ',num2str(devz(13))]
        [tag(14,:)  num2str(count(14)),' stdx ',num2str(devx(14)),' stdy
',num2str(devy(14)),' stdz ',num2str(devz(14))]
        [tag(15,:)  num2str(count(15)),' stdx ',num2str(devx(15)),' stdy
',num2str(devy(15)),' stdz ',num2str(devz(15))]
        [tag(16,:)  num2str(count(16)),' stdx ',num2str(devx(16)),' stdy
',num2str(devy(16)),' stdz ',num2str(devz(16))]
        [tag(17,:)  num2str(count(17)),' stdx ',num2str(devx(17)),' stdy
',num2str(devy(17)),' stdz ',num2str(devz(17))]
        [tag(18,:)  num2str(count(18)),' stdx ',num2str(devx(18)),' stdy
',num2str(devy(18)),' stdz ',num2str(devz(18))]
        [tag(19,:)  num2str(count(19)),' stdx ',num2str(devx(19)),' stdy
',num2str(devy(19)),' stdz ',num2str(devz(19))]
        [tag(20,:)  num2str(count(20)),' stdx ',num2str(devx(20)),' stdy
',num2str(devy(20)),' stdz ',num2str(devz(20))]
        [tag(21,:)  num2str(count(21)),' stdx ',num2str(devx(21)),' stdy
',num2str(devy(21)),' stdz ',num2str(devz(21))]
        [tag(22,:)  num2str(count(22)),' stdx ',num2str(devx(22)),' stdy
',num2str(devy(22)),' stdz ',num2str(devz(22))]
        [tag(23,:)  num2str(count(23)),' stdx ',num2str(devx(23)),' stdy
',num2str(devy(23)),' stdz ',num2str(devz(23))]
        [tag(24,:)  num2str(count(24)),' stdx ',num2str(devx(24)),' stdy
',num2str(devy(24)),' stdz ',num2str(devz(24))]
        [tag(25,:)  num2str(count(25)),' stdx ',num2str(devx(25)),' stdy
',num2str(devy(25)),' stdz ',num2str(devz(25))]})
end
```

Appendice C

Viene qui riportato il codice completo dell' m-File Matlab per visualizzare i trends, i test ANOVA ed i t-test per le deviazioni standard di più prove all' interno della stessa cartella.

```
Clear;close all;clc


file_directory = uigetdir();
if isempty(file_directory)
    file_directory = uigetdir();
end
STD_X=zeros(25,1);
lista = dir(file_directory);
for i = 1:length(lista)-2
    filename = lista(i+2).name;
    full_path = fullfile(file_directory, filename);

    A = txt2mat(full_path);
    [n,m]=size(A);
    x=zeros(n/25,25);
    y=zeros(n/25,25);
    z=zeros(n/25,25);
    d=zeros(n/25,25);
    for k=1:25


        x(:,k)=A(k:25:end-25+k,1);
        y(:,k)=A(k:25:end-25+k,2);
        z(:,k)=A(k:25:end-25+k,3);
        d(:,k)=sqrt(x(:,k).^2+y(:,k).^2+z(:,k).^2);

    end
    devx=std(x,0,1);
    devy=std(y,0,1);
    devz=std(z,0,1);
    devd=std(d,0,1);
    if(i ==1)
        STD_X=devx';
        STD_Y=devy';
        STD_Z=devz';
        STD_D=devd';
    else
        STD_X=[STD_X devx'];
        STD_Y=[STD_Y devy'];
        STD_Z=[STD_Z devz'];
        STD_D=[STD_D devd'];
    end
end

Tag={'spineBase','midSpine','neck','head','leftShoulder','leftElbow','le
ftWrist','leftHand','rightShoulder','rightElbow','rightWrist','rightHand
','leftHip','leftKnee','leftAnkle','leftFoot','rightHip','rightKnee','ri
ghtAnkle','rightFoot','spineShoulder','LeftHandTip','leftThumb','rightHa
ndTip','rightThumb'};
```

```
for i=1:5
    j=i*5;
    figure(i)
    subplot(5,1,1)
    plot(STD_D(j-4,:))
    xlabel(Tag{j-4})
    subplot(5,1,2)
    plot(STD_D(j-3,:))
    xlabel(Tag{j-3})
    subplot(5,1,3)
    plot(STD_D(j-2,:))
    xlabel(Tag{j-2})
    subplot(5,1,4)
    plot(STD_D(j-1,:))
    xlabel(Tag{j-1})
    subplot(5,1,5)
    plot(STD_D(j,:))
    xlabel(Tag{j})
end


Media=mean(STD_D)
[p,tbl,stats]=anova1(STD_D);
c=multcompare(stats,'Alpha',0.01,'CType','lsd');

HIGH=[STD_D(5:12,:);STD_D(22:25,:)];
BODY=[STD_D(1:4,:);STD_D(21,:)];
LOW=[STD_D(13:20,:)];

mean_high=mean(HIGH)
std_high=std(HIGH)
mean_body=mean(BODY)
std_body=std(BODY)
mean_low=mean(LOW)
std_low=std(LOW)

[p1,tbl1,stats1]=anova1(HIGH);
c1=multcompare(stats1,'Alpha',0.01,'CType','lsd');
[p2,tbl2,stats2]=anova1(BODY);
c2=multcompare(stats2,'Alpha',0.01,'CType','lsd');
[p3,tbl3,stats3]=anova1(LOW);
c3=multcompare(stats3,'Alpha',0.01,'CType','lsd');
```

Appendice D

Viene qui riportata la classe C# che implementa il filtraggio dei Joint. Creando un istanza della classe e passandole l'oggetto Body, questa restituisce una struttura di dati 3D (CameraSpacePoint[]) con le posizioni filtrate dei 25 Joint.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Kinect;
using System.Collections;

namespace MyKinectForm
{
    public class KinectJointFilter
    {
        public struct TRANSFORM_SMOOTH_PARAMETERS
        {
            public float fSmoothing;            // [0..1], lower values closer to
raw data
            public float fCorrection;           // [0..1], lower values slower to
correct towards the raw data
            public float fPrediction;           // [0..n], the number of frames
to predict into the future
            public float fJitterRadius;         // The radius in meters for
jitter reduction
            public float fMaxDeviationRadius;   // The maximum radius in meters
that filtered positions are allowed to deviate from raw data
        }



        public class FilterDoubleExponentialData
        {
            public CameraSpacePoint m_vRawPosition;
            public CameraSpacePoint m_vFilteredPosition;
            public CameraSpacePoint m_vTrend;
            public int m_dwFrameCount;
        }

        // Holt Double Exponential Smoothing filter
        CameraSpacePoint[] m_pFilteredJoints;
        FilterDoubleExponentialData[] m_pHistory;
        float m_fSmoothing;
        float m_fCorrection;
        float m_fPrediction;
        float m_fJitterRadius;
        float m_fMaxDeviationRadius;

        public KinectJointFilter()
        {
            m_pFilteredJoints = new CameraSpacePoint[Body.JointCount];
            m_pHistory = new FilterDoubleExponentialData[Body.JointCount];
            for (int i = 0; i < Body.JointCount; i++)
            {
```

```
                m_pHistory[i] = new FilterDoubleExponentialData();
            }

            Init(0.25f, 0.25f, 0.25f, 0.03f, 0.05f);
        }



        ~KinectJointFilter()
        {
            Shutdown();
        }

        public void Init(float fSmoothing, float fCorrection, float fPrediction,
float fJitterRadius, float fMaxDeviationRadius)
        {

            Reset(fSmoothing, fCorrection, fPrediction, fJitterRadius,
fMaxDeviationRadius);
        }

        public void Shutdown()
        {
        }

        public void Reset(float fSmoothing, float fCorrection, float fPrediction,
float fJitterRadius, float fMaxDeviationRadius)
        {
            if (m_pFilteredJoints == null || m_pHistory == null)
            {
                return;
            }

            m_fMaxDeviationRadius = fMaxDeviationRadius; // Size of the max
prediction radius Can snap back to noisy data when too high
            m_fSmoothing = fSmoothing;                   // How much smothing will
occur.  Will lag when too high
            m_fCorrection = fCorrection;                 // How much to correct
back from prediction.  Can make things springy
            m_fPrediction = fPrediction;                 // Amount of prediction
into the future to use. Can over shoot when too high
            m_fJitterRadius = fJitterRadius;             // Size of the radius
where jitter is removed. Can do too much smoothing when too high

            for (int i = 0; i < Body.JointCount; i++)
            {
                m_pFilteredJoints[i].X = 0.0f;
                m_pFilteredJoints[i].Y = 0.0f;
                m_pFilteredJoints[i].Z = 0.0f;

                m_pHistory[i].m_vFilteredPosition.X = 0.0f;
                m_pHistory[i].m_vFilteredPosition.Y = 0.0f;
                m_pHistory[i].m_vFilteredPosition.Z = 0.0f;

                m_pHistory[i].m_vRawPosition.X = 0.0f;
                m_pHistory[i].m_vRawPosition.Y = 0.0f;
                m_pHistory[i].m_vRawPosition.Z = 0.0f;

                m_pHistory[i].m_vTrend.X = 0.0f;
                m_pHistory[i].m_vTrend.Y = 0.0f;
                m_pHistory[i].m_vTrend.Z = 0.0f;
```

```
                m_pHistory[i].m_dwFrameCount = 0;
            }
        }

        //--------------------------------------------------------------------
--------------
        // Implementation of a Holt Double Exponential Smoothing filter. The
double exponential
        // smooths the curve and predicts.  There is also noise jitter removal.
And maximum
        // prediction bounds.  The paramaters are commented in the init function.
        //--------------------------------------------------------------------
--------------
        public void UpdateFilter(Body pBody)
        {
            if (pBody == null)
            {
                return;
            }

            // Check for divide by zero. Use an epsilon of a 10th of a millimeter
            m_fJitterRadius = Math.Max(0.0001f, m_fJitterRadius);

            TRANSFORM_SMOOTH_PARAMETERS SmoothingParams;

            for (JointType jt = JointType.SpineBase; jt <= JointType.ThumbRight;
jt++)
            {
                SmoothingParams.fSmoothing = m_fSmoothing;
                SmoothingParams.fCorrection = m_fCorrection;
                SmoothingParams.fPrediction = m_fPrediction;
                SmoothingParams.fJitterRadius = m_fJitterRadius;
                SmoothingParams.fMaxDeviationRadius = m_fMaxDeviationRadius;

                // If inferred, we smooth a bit more by using a bigger jitter
radius
                Microsoft.Kinect.Joint joint = pBody.Joints[jt];
                if (joint.TrackingState == TrackingState.Inferred)
                {
                    SmoothingParams.fJitterRadius *= 2.0f;
                    SmoothingParams.fMaxDeviationRadius *= 2.0f;
                }

                UpdateJoint(pBody, jt, SmoothingParams);
            }
        }

        //--------------------------------------------------------------------
--------------
        // if joint is 0 it is not valid.
        //--------------------------------------------------------------------
--------------
        bool JointPositionIsValid(CameraSpacePoint vJointPosition)
        {
            return (vJointPosition.X != 0.0f ||
                    vJointPosition.Y != 0.0f ||
                    vJointPosition.Z != 0.0f);
        }

        public CameraSpacePoint[] GetFilteredJoints()
        {
            return m_pFilteredJoints;
```

```
        }

        CameraSpacePoint CSVectorSet(float x, float y, float z)
        {
            CameraSpacePoint point = new CameraSpacePoint();

            point.X = x;
            point.Y = y;
            point.Z = z;

            return point;
        }

        CameraSpacePoint CSVectorZero()
        {
            CameraSpacePoint point = new CameraSpacePoint();

            point.X = 0.0f;
            point.Y = 0.0f;
            point.Z = 0.0f;

            return point;
        }

        public CameraSpacePoint CSVectorAdd(CameraSpacePoint p1, CameraSpacePoint
p2)
        {
            CameraSpacePoint sum = new CameraSpacePoint();

            sum.X = p1.X + p2.X;
            sum.Y = p1.Y + p2.Y;
            sum.Z = p1.Z + p2.Z;

            return sum;
        }

        public CameraSpacePoint CSVectorSub(CameraSpacePoint p1, CameraSpacePoint
p2)
        {
            CameraSpacePoint sub = new CameraSpacePoint();

            sub.X = p1.X - p2.X;
            sub.Y = p1.Y - p2.Y;
            sub.Z = p1.Z - p2.Z;

            return sub;
        }

        public CameraSpacePoint CSVectorCross(CameraSpacePoint p1,
CameraSpacePoint p2)
        {
            CameraSpacePoint Cross = new CameraSpacePoint();

            Cross.X = p1.Y * p2.Z - p2.Y * p1.Z;
            Cross.Y = (p1.X * p2.Z - p2.X * p1.Z) * -1;
            Cross.Z = p1.X * p2.Y - p2.X * p1.Y;

            return Cross;
        }

        public float CSVectorDot(CameraSpacePoint p1, CameraSpacePoint p2)
        {
```

```
            float dot;

            dot = p1.X * p2.X + p1.Y * p2.Y + p1.Z * p2.Z;

            return dot;
        }

        public CameraSpacePoint CSVectorScale(CameraSpacePoint p, float scale)
        {
            CameraSpacePoint point = new CameraSpacePoint();

            point.X = p.X * scale;
            point.Y = p.Y * scale;
            point.Z = p.Z * scale;

            return point;
        }

        public CameraSpacePoint CSVectorDiv(CameraSpacePoint p, float scale)
        {
            CameraSpacePoint div = new CameraSpacePoint();

            div.X = p.X / scale;
            div.Y = p.Y / scale;
            div.Z = p.Z / scale;

            return div;
        }

        public CameraSpacePoint CSVectorSubtract(CameraSpacePoint p1,
CameraSpacePoint p2)
        {
            CameraSpacePoint diff = new CameraSpacePoint();

            diff.X = p1.X - p2.X;
            diff.Y = p1.Y - p2.Y;
            diff.Z = p1.Z - p2.Z;

            return diff;
        }

        public float CSVectorLength(CameraSpacePoint p)
        {
            return Convert.ToSingle(Math.Sqrt(p.X * p.X + p.Y * p.Y + p.Z * p.Z));
        }

        void UpdateJoint(Body body, JointType jt, TRANSFORM_SMOOTH_PARAMETERS
smoothingParams)
        {
            CameraSpacePoint vPrevRawPosition;
            CameraSpacePoint vPrevFilteredPosition;
            CameraSpacePoint vPrevTrend;
            CameraSpacePoint vRawPosition;
            CameraSpacePoint vFilteredPosition;
            CameraSpacePoint vPredictedPosition;
            CameraSpacePoint vDiff;
            CameraSpacePoint vTrend;
            float fDiff;
            bool bJointIsValid;

            Microsoft.Kinect.Joint joint = body.Joints[jt];
```

```
            vRawPosition = joint.Position;
            vPrevFilteredPosition = m_pHistory[(int)jt].m_vFilteredPosition;
            vPrevTrend = m_pHistory[(int)jt].m_vTrend;
            vPrevRawPosition = m_pHistory[(int)jt].m_vRawPosition;
            bJointIsValid = JointPositionIsValid(vRawPosition);

            // If joint is invalid, reset the filter
            if (!bJointIsValid)
            {
                m_pHistory[(int)jt].m_dwFrameCount = 0;
            }

            // Initial start values
            if (m_pHistory[(int)jt].m_dwFrameCount == 0)
            {
                vFilteredPosition = vRawPosition;
                vTrend = CSVectorZero();
                m_pHistory[(int)jt].m_dwFrameCount++;
            }
            else if (m_pHistory[(int)jt].m_dwFrameCount == 1)
            {
                vFilteredPosition = CSVectorScale(CSVectorAdd(vRawPosition,
vPrevRawPosition), 0.5f);
                vDiff = CSVectorSubtract(vFilteredPosition,
vPrevFilteredPosition);
                vTrend = CSVectorAdd(CSVectorScale(vDiff,
smoothingParams.fCorrection), CSVectorScale(vPrevTrend, 1.0f -
smoothingParams.fCorrection));
                m_pHistory[(int)jt].m_dwFrameCount++;
            }
            else
            {
                // First apply jitter filter
                vDiff = CSVectorSubtract(vRawPosition, vPrevFilteredPosition);
                fDiff = CSVectorLength(vDiff);

                //if (fDiff > smoothingParams.fJitterRadius)
                //{
                //    vFilteredPosition = CSVectorAdd(CSVectorScale(vRawPosition,
(smoothingParams.fJitterRadius / fDiff)),
                //        CSVectorScale(vPrevFilteredPosition, 1.0f -
(smoothingParams.fJitterRadius / fDiff)));
                //}
                //else
                //{
                //    vFilteredPosition = vRawPosition;
                //}

                if (fDiff <= smoothingParams.fJitterRadius)
                {
                    vFilteredPosition = CSVectorAdd(CSVectorScale(vRawPosition,
fDiff / smoothingParams.fJitterRadius),
                        CSVectorScale(vPrevFilteredPosition, 1.0f - fDiff /
smoothingParams.fJitterRadius));
                }
                else
                {
                    vFilteredPosition = vRawPosition;
                }

                // Now the double exponential smoothing filter
```

```
                vFilteredPosition = CSVectorAdd(CSVectorScale(vFilteredPosition,
1.0f - smoothingParams.fSmoothing),
                    CSVectorScale(CSVectorAdd(vPrevFilteredPosition, vPrevTrend),
smoothingParams.fSmoothing));


                vDiff = CSVectorSubtract(vFilteredPosition,
vPrevFilteredPosition);
                vTrend = CSVectorAdd(CSVectorScale(vDiff,
smoothingParams.fCorrection), CSVectorScale(vPrevTrend, 1.0f -
smoothingParams.fCorrection));
            }

            // Predict into the future to reduce latency
            vPredictedPosition = CSVectorAdd(vFilteredPosition,
CSVectorScale(vTrend, smoothingParams.fPrediction));

            // Check that we are not too far away from raw data
            vDiff = CSVectorSubtract(vPredictedPosition, vRawPosition);
            fDiff = CSVectorLength(vDiff);

            if (fDiff > smoothingParams.fMaxDeviationRadius)
            {
                vPredictedPosition = CSVectorAdd(CSVectorScale(vPredictedPosition,
smoothingParams.fMaxDeviationRadius / fDiff),
                    CSVectorScale(vRawPosition, 1.0f -
(smoothingParams.fMaxDeviationRadius / fDiff)));
            }

            // Save the data from this frame
            m_pHistory[(int)jt].m_vRawPosition = vRawPosition;
            m_pHistory[(int)jt].m_vFilteredPosition = vFilteredPosition;
            m_pHistory[(int)jt].m_vTrend = vTrend;

            // Output the data
            m_pFilteredJoints[(int)jt] = vPredictedPosition;
        }

        public CameraSpacePoint GetFilteredJoint(int jt)
        {
            return m_pFilteredJoints[jt];
        }

    }
}
```