Università di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di laurea specialistica in Informatica

Studio ed implementazione di un sistema basato sulle reti neurali per il calcolo della percezione attesa di flusso ottico

Nino Cauli

Realtore:
Cecilia Laschi

8 Ottobre 2010

Indice

Sommario

L'uomo, per rispondere agli stimoli sensoriali, non usa un semplice paradigma reattivo nel quale prima analizza i dati sensoriali e poi esegue l'azione corrispondente. Nell'attività di percezione l'essere umano anticipa le conseguenze dell'azione, realizzando una simulazione interna dell'azione stessa. Questo avviene perché, se il cervello basasse i suoi controlli motori esclusivamente sui feedback sensoriali, sarebbe sempre soggetto ad un ritardo intrinseco rispetto all'ambiente che lo circonda. Tutto ciò è vero anche nella robotica, soprattutto quella umanoide, dove spesso il loop percezione-azione può risultare molto lento. Per questo motivo sono stati proposti, per il controllo, diversi modelli interni basati su percezione, come ad esempio l'Expected Perception (EP) scheme [11, 10] . Questo concetto può essere applicato a qualsiasi processo di percezione-azione a partire dal grasping, passando per il controllo dei movimenti oculari, fino ad arrivare alla locomozione. E' proprio su quest'ultimo aspetto che si concentra il seguente lavoro di tesi. Si descrive lo studio e l'implementazione di un modello per il controllo della locomozione di un robot bipede umanoide, sfruttando il calcolo della percezione attesa. Dopo un'attenta analisi dello stato dell'arte è stata costruita un'architettura basata su di una particolare implementazione [6] dell'EP scheme di Edoardo Datteri. Questo schema sfrutta la predizione per alleggerire il loop di controllo della locomozione in un robot. Grazie al calcolo della percezione attesa delle immagini delle camere e il confronto di quest'ultima con le immagini reali, si riesce ad evitare di ripetere ogni passo i calcoli di visual processing (collo di bottiglia del sistema). In questa tesi si è modellato uno schema di controllo che calcola la percezione attesa a partire dal flusso ottico per mezzo di una rete neurale ricorrente. E' stato, successivamente, implementato il modulo atto alla generazione dell'EP (estrazione del flusso ottico e calcolo di quello predetto), sono stati effettuati i test e ne sono stati analizzati i risultati che dimostrano la validità dell'architettura proposta.

Capitolo 1

Il ruolo della predizione della percezione nel controllo robotico

Negli esseri umani la percezione non è solo un'interpretazione dei segnali sensoriali, ma una predizione delle conseguenze delle azioni. L'attività di percezione non è confinata all'interpretazione delle informazioni sensoriali, ma anticipa le conseguenze dell'azione, come una simulazione interna dell'azione stessa [8]. L'insieme dei sensori che sono implicati nell'analisi del movimento e dello spazio ("Sense of Moviment") sono particolarmente importanti per questo scopo. Il nostro cervello non basa i controlli motori sui feedback sensoriali, che sono troppo lenti, ma sull'input sensoriale predetto. Nei robot, il comportamento predittivo, unito in modo adeguato con quello reattivo, può migliorare notevolmente l'efficienza del ciclo percezione-azione.

1.1 Introduzione

Inizialmente lo studio della robotica e lo sviluppo dell'intelligenza artificiale erano fortemente collegati al concetto di robot industriale. Il robot stesso veniva visto come un agente inserito all'interno di un ambiente strutturato noto a priori. I task da compiere erano semplici e ripetitivi. Per questo motivo si sviluppò un approccio nel quale la rappresentazione del mondo sul robot veniva totalmente

scollegata dalla percezione. La realizzazione della cognizione umana era espressa attraverso alcuni insiemi di simboli, usati per pianificare come agire sul mondo.

L'evoluzione della robotica da allora fino ai giorni nostri ha però cambiato radicalmente questa idea di robot. Ormai la robotica non si limita più all'impiego in ambienti industriali, ma spazia da quello medico al militare, dal settore dell'assistenza a quello ludico fino ad arrivare al mercato di largo consumo. Questo porta l'agente robotico a non doversi più interfacciare in uno spazio di azione costruito ad hoc e ben noto, ma bensì in un ambiente vario e sconosciuto. Se si aggiungono anche la presenza all'interno del ambiente di persone, animali e tutti i numerosi stimoli imprevisti che possono essere presenti in spazi come case, ospedali, strade, etc. ci si rende conto come i precedenti approcci (Gerarchico, Reattivo, Ibrido (paragrafo 1.2)) non siano più adeguati.

Partendo dal presupposto che i robot dovessero ormai essere inseriti all'interno di ambienti reali nei quali animali ed esseri umani si sono evoluti o addirittura che siano stati creati appositamente per la vita umana, si è cominciato a studiare più approfonditamente il comportamento biologico per lo sviluppo di architetture robotiche. Da questi studi si è sviluppato l'idea secondo la quale in natura percezione, pianificazione ed azione siano strettamente legati alla forma fisica del sistema biologico. Così il concetto di architettura di controllo totalmente indipendente dalla piattaforma su cui viene implementata è andato via via sparendo.

Queste ed altre considerazioni hanno portato alla nascita dell'idea di architetture basate su modelli interni [5, 23](paragrafo 1.3) del sistema da controllare. In essi gioca un ruolo fondamentale la predizione. Grazie al modello interno è possibile anticipare la percezione sensoriale che scaturisce dai cambiamenti nell'ambiente dati dall'attuale comportamento del sistema robotico (o viceversa fornire il comportamento del sistema necessario per raggiungere un determinato stato percettivo). In questo modo si riescono a risolvere i problemi dati dal ritardo intrinseco nel processare dati provenienti da un ambiente complesso e caotico, rendendo allo stesso tempo inscindibili architettura di controllo e piattaforma.

Una delle prime implementazioni robotiche di modello interno è stato l'Expected Perception Scheme [11, 10](paragrafo 1.4). Questa tesi si focalizza sull'implementazione di questo modello per la realizzazione di un sistema di controllo locomotorio di un robot umanoide.

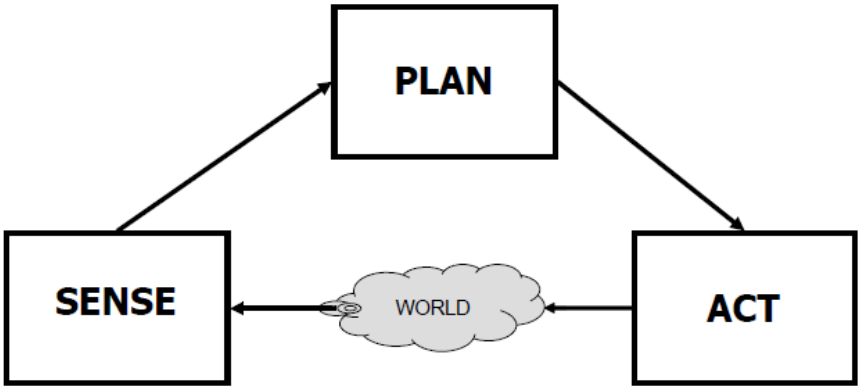

Figura 1.1: Il Paradigma Gerarchico presenta una struttura feedforward in cui si parte dal blocco SENSE, si passa dal PLAN e si arriva all'ACT.

1.2 Paradigmi di IA nei sistemi robotici

Come già detto i paradigmi classici di IA per il controllo dei sistemi robotici risultano inadeguati per gestire la complessità dei robot moderni e degli ambienti in cui sono inseriti. Si analizzeranno adesso più nel dettaglio per capire meglio quali siano i loro difetti e come ci si sia spostati verso l'idea di modelli interni.

Per prima cosa tutte le architetture classiche di controllo robotico si basano sulla correlazione di tre primitive di base: SENSE, PLAN, ACT [33]. La prima è formata da dei blocchi che prendono in input i dati sensoriali e li trasformano in informazioni utili per gli altri blocchi (es. i sensori del robot). La primitiva PLAN elabora le informazioni provenienti dai blocchi sensoriali e quelle cognitive, pianifica le azioni da compiere ed invia le direttive ai blocchi ACT. Questi ultimi tramutano le direttive (o i direttamente i dati sensoriali a seconda del paradigma usato) in comandi motori per gli attuatori.

Il *Paradigma Gerarchico* è stato il primo ad essere inventato. Proposto verso la fine degli anni 60, ha continuato ad essere l'architettura più utilizzata per vent'anni. Il concetto su cui si basa è molto semplice: l'essere umano, quando deve svolgere un compito, prima riceve i dati sensoriali dal mondo, poi gli elabora ed in fine esegue le azioni adeguate. Questo ha portato ad un'architettura strettamente top-down in cui per prima cosa i blocchi SENSE estrapolano i dati sensoriali dal mondo. Questi dati vengono poi analizzati dal blocco PLAN che invierà i comandi motori necessari per svolgere il task ai blocchi ACT (Figura

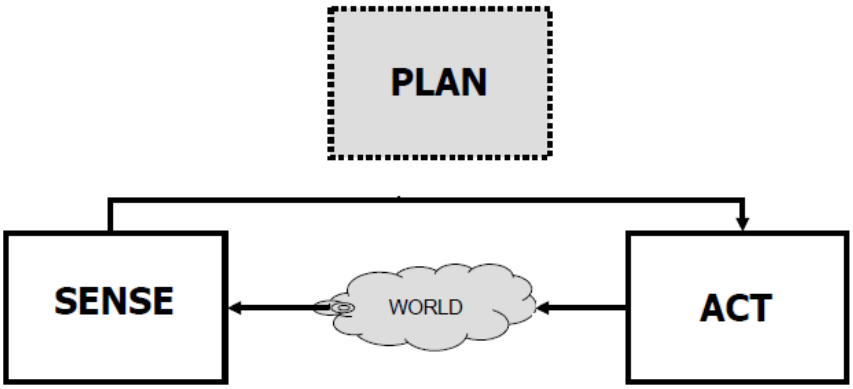

Figura 1.2: Il Paradigma Reattivo presenta solo i blocchi SENSE e ACT eliminando il PLAN.

1.1). Nel paradigma Gerarchico i dati sensoriali servono a formare un modello globale del mondo. Questo permette al robot di lavorare solamente all'interno di un ambiente strutturato. In caso contrario la complessità computazionale derivante dall'analisi dei dati sensoriali rallenterebbe eccessivamente i tempi di azione del sistema. Il robot non riuscirebbe più a lavorare in real time.

Dallo studio del comportamento di sistemi biologici più semplici, si è sviluppata, tra la fine degli anni 80 e l'inizio dei 90, un'architettura che risolve il problema della lentezza del paradigma Gerarchico: il *Paradigma Reattivo*. I ricercatori hanno notato come animali semplici (es. insetti) riuscissero a compiere azioni complesse senza alcuna rappresentazione simbolica dell'ambiente attraverso un sistema di controllo parallelo e distribuito. Da questo nasce quindi l'idea di eliminare il blocco PLAN e connettere direttamente la percezione (SENSE) ai comandi motori (ACT) (Figura 1.2). Per realizzare un'architettura del genere l'intero sistema viene diviso in più moduli SENSE-ACT (chiamati comportamenti) che lavorano in modo parallelo ed indipendente. In questo modo una combinazione dei vari comportamenti porta all'esecuzione di compiti complessi. Il paradigma Reattivo permette di ottenere un sistema rapido e leggero, sebbene non riesca ad agire bene in ambienti eccessivamente complessi.

La completa eliminazione della pianificazione (PLAN) non permette di rispondere ad informazioni memorizzate sull'ambiente. Questo problema ha portato allo sviluppo negli anni 90 del *Paradigma Ibrido*. Nei sistemi ibridi ritorna un blocco di pianificazione che, però, viene affiancato ad un sistema reattivo (Figu-

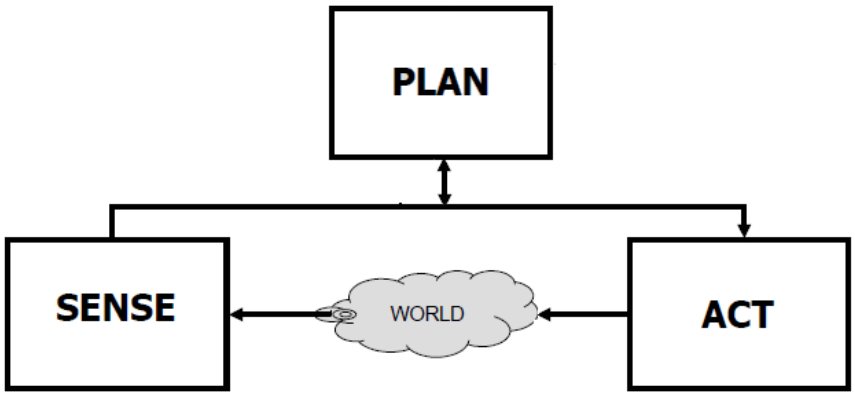

Figura 1.3: Il Paradigma Ibrido controlla il robot in modo reattivo attraverso la supervisione di un pianificatore.

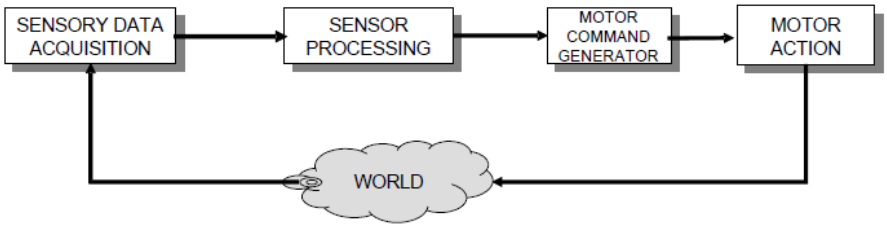

Figura 1.4: Sistema feedback. notare il loop che parte dal mondo, passa per l'acquisizione ed elaborazione dei dati sensoriali, la generazione dei comandi motori, per poi tornare al mondo attraverso l'attuazione degli stessi.

ra 1.3). Il PLAN ha il compito di suddividere il task in sotto-task e di scegliere quali comportamenti eseguire per risolverlo. I dati sensoriali saranno disponibili sia per i comportamenti che ne avranno bisogno e sia per il PLAN. Il vantaggio di separare il PLAN dal sistema SENSE-ACT è quello di poter eseguire i due in modo asincrono con frequenza differente. In questo modo il sistema può funzionare in maniera reattiva mentre viene elaborata la pianificazione.

1.3 Modelli interni

Fino ad ora si è mostrato quali siano state le soluzioni proposte durante la fine del secolo scorso per il problema dell'aumento di complessità dell'ambiente entro

il quale interagisce il robot. Per gestire al meglio l'aumento della lentezza di elaborazione dei sistemi gerarchici si è distribuito il controllo nei piccoli comportamenti paralleli dei sistemi reattivi. Per essere coordinati al meglio i moduli reattivi sono stati affiancati da un pianificatore centrale nei sistemi ibridi. Visti in un ottica più generale, tutti i paradigmi presentati precedentemente possono essere raggruppati in un unica classe: la classe dei *sistemi di tipo feedback* (Figura 1.4). In questo tipo di sistemi si parte da un'estrazione dei dati sensoriali dal mondo, si continua con la loro elaborazione, si generano i comandi motori, ed in fine si produce l'azione influenzando il mondo. I sistemi feedback presentano, però, un grosso problema di fondo. Anche andando ad analizzare sistemi molto semplici, in cui il semplice paradigma reattivo è sufficiente, si nota come debba passare del tempo perché i sensori acquisiscano i dati e gli attuatori eseguano il comando motorio richiesto. In questo lasso di tempo l'ambiente si evolverà e l'azione eseguita dal robot potrebbe non essere più adeguata a svolgere il task richiesto. È come il gatto che insegue la propria coda. La situazione peggiora ancor di più nei sistemi più evoluti in cui sono presenti i ritardi dati dal pianificatore ad alto livello. Nell'uomo, ad esempio, i ritardi dati dalla cattura dello stimolo sensoriale, la conduzione del segnale nervoso, il processamento centrale, il sistema decisionale e l'attuazione del comando motorio portano a ritardi dell'ordine di 200-300 ms per una semplice risposta visiva [30]. Queste considerazioni ci fanno capire come sia indispensabile una predizione degli eventi per poter rispondere in modo tempestivo ad essi.

Sono stati fatti diversi studi biologici su sistemi che eseguono l'anticipazione attraverso la predizione ottenuta dagli eventi passati. Un neuroscienziato attivo in questo campo di ricerca è Alain Berthoz . Quest'ultimo ritiene che la memoria giochi un ruolo fondamentale nell'anticipazione delle conseguenze delle proprie azioni [8]. Per riuscire a gestire il concetto di predizione, essendo i sistemi feedback inadeguati, sono stati sviluppati *Modelli Interni* (MI) anticipatori [30, 24, 22]. I MI modellizzano lo spazio percettivo, del mondo o dei comandi motori e permettono una sua anticipazione a partire dalla stato corrente del sistema o del mondo. Esistono tre tipi di modelli interni [31].

Il modello anticipativo (Figura 1.5) è il primo di essi. Questo modello prende in ingresso lo stato percettivo attuale e i comandi motori emanati fornendo in risposta lo stato percettivo futuro predetto. Un sistema di controllo motorio basato su questo modello può così basare le proprie azioni non più solo sulle percezioni attuali ma anche su quelle predette, trovandosi così sempre sincrono con il mondo e pronto a reagire agli imprevisti.

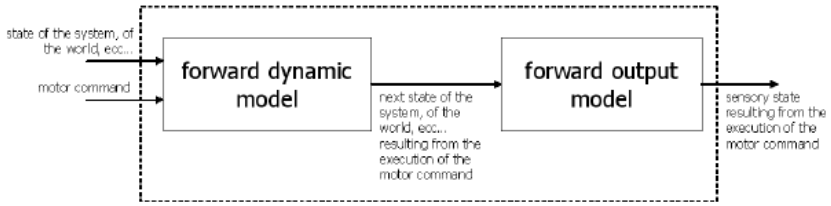

Figura 1.5: Il modello anticipativo a partire dallo stato attuale e dal comando motorio fornisce lo stato risultante l'esecuzione del comando.

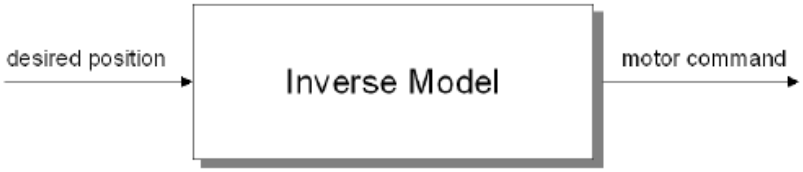

Figura 1.6: Il modello inverso fornisce i comandi motori che servono per raggiungere una posizione desiderata.

La seconda categoria assomiglia molto a quella dei modelli anticipativi con la differenza che essa modella esclusivamente il comportamento dell'ambiente esterno (ad esempio viene modellato il comportamento di oggetti all'interno dell'ambiente esterno e ne viene predetta la posizione futura).

Per finire ci sono i modelli inversi [24](Figura 1.6), chiamati così perché si muovono nella direzione opposta rispetto ai modelli anticipativi. Questi, infatti, prendono in ingresso lo stato attuale del sistema e quello che si vuole raggiungere e danno in risposta i comandi motori necessari per passare da uno all'altro.

1.4 Expected Perception scheme

Tra le varie soluzioni proposte di MI molto interessante è il predittore di Smith [30]. Quest'ultimo inserisce all'interno di un sistema feedback classico il modello anticipativo. Nel predittore il comando motorio viene passato sia agli attuatori che al blocco di predizione, il quale riceve anche l'input sensoriale. A questo punto lo stato stimato viene confrontato con quello desiderato e l'errore viene utilizzato per calcolare il comando motorio successivo. Nello schema proposto da Johansson [22] il MI viene usato per predire l'immagine tattile di un oggetto

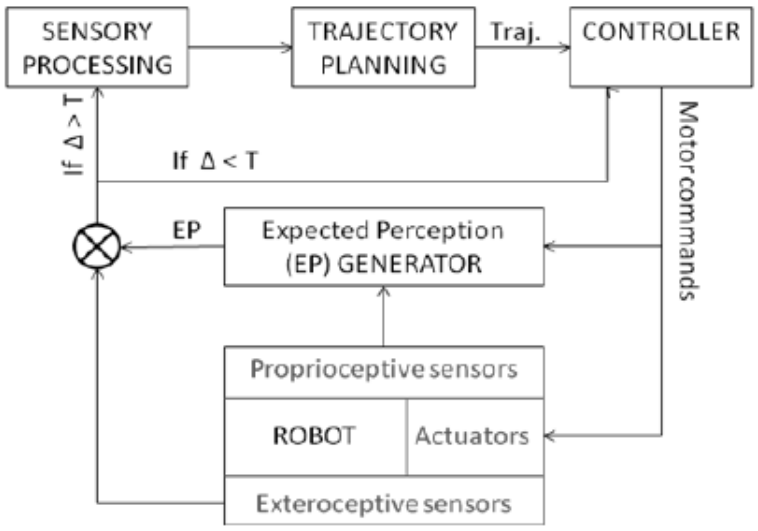

Figura 1.7: L'EP Scheme inserisce il concetto di modello anticipativo all'interno di un sistema di controllo basato sulla visione.

durante il grasping. La predizione viene usata per calibrare la presa e le forze di carico. Se la calibrazione, una volta che la mano ha raggiunto l'oggetto, risulta errata viene aggiornata, altrimenti si utilizza quella predetta.

Una delle prime implementazioni di modello anticipativo è stato realizzato nell'ARTS Lab della Scuola Superiore Sant'Anna. Prendendo spunto dai modelli esposti precedentemente, Datteri ha ideato un sistema di controllo per un braccio meccanico basato sulla visione chiamato *Expected Perception scheme* [11, 10](Figura 1.7). Lo schema del sistema di controllo è interamente basato sul suo blocco centrale: l'Expected Perception (EP) Generator. Questo blocco ha la funzione di predittore. Infatti prende in ingresso i dati percettivi sensoriali ed i comandi motori e fornisce la predizione delle informazioni sensoriali risultanti dall'esecuzione di quei comandi (EP). L'EP Generator è inserito all'interno di un sistema feedback basato sulla visione. In questo tipo di sistemi il collo di bottiglia è proprio l'elaborazione visiva. È impensabile ottenere un sistema del genere in tempo reale dove l'elaborazione visiva viene eseguita in ogni ciclo. La percezione attesa prodotta dal generatore viene così utilizzata per evitare questo. In ogni ciclo l'EP calcolato precedentemente viene confrontato con i dati sensoriali appena estratti dall'ambiente. Se la differenza supera una certa soglia vuol dire che un evento imprevisto è avvenuto e verrà eseguita l'e-

laborazione visiva per gestire il problema. Altrimenti si prenderà per buona la predizione e si continuerà basandosi su di essa saltando il blocco di elaborazione.

Questo schema di controllo è stato in seguito utilizzato per risolvere task più specifici come il grasping di oggetti [26]. In questa implementazione la percezione attesa consiste nell'immagine tattile predetta.

Come dimostrano diversi studi neuroscientifici come quelli portati avanti da Berthoz [8], la predizione è un processo intrinseco in ogni sottosistema di controllo dell'essere umano. Per questo motivo l'EP scheme può essere facilmente inserito nella progettazione di qualsiasi loop di controllo basato sulla visione (movimenti oculari, grasping, locomozione). Un percorso di ricerca seguito dalla Scuola recentemente è proprio l'applicazione del concetto di percezione attesa all'interno di un sistema di locomozione basato sulla visione. Questi studi si inseriscono in un progetto più ampio (RoboSoM project) nel quale si cerca di realizzare una piattaforma robotica che si muova utilizzando i concetti di predizione e di *human Sense of Movement* elaborati da Alain Berthoz [8]. Questa tesi si concentra sulla progettazione ed implementazione dell'EP Generator all'interno di un sistema di questo tipo.

Capitolo 2

Percezione Attesa nel controllo della locomozione di un robot

Si è visto dal capitolo precedente l'importanza che detiene la predizione nel controllo di sistemi basati sul concetto percezione-azione. L'anticipazione delle conseguenze delle azioni nello spazio sensoriale risulta indispensabile per poter svolgere determinati task e rimanere in tempo reale con il mondo. Sebbene la predizione risulti importante in qualsiasi sistema di controllo basato su input sensoriale, diventa fondamentale nei task in cui l'input sensoriale è molto complesso e la sua elaborazione necessita un grosso sforzo computazionale. A questa categoria appartengono tutti i sistemi di controllo basati sulla visione: coordinazione mano occhio, manipolazione di oggetti, locomozione, etc... Sono stati fatti numerosi studi sull'applicazione di modelli interni anticipativi nei sistemi di controllo con input sensoriale visivo, sia da un punto di vista biologico che da quello robotico. Questo vale anche per il controllo della locomozione, anche se in tale campo risultano poche le implementazioni di sistemi anticipativi su robot bipedi umanoidi [18, 20]. In questa tesi si sviluppa un modello anticipativo del tipo EP scheme basandosi sul lavoro effettuato nel progetto RoboSoM realizzato all'interno dell'ARTS Lab della Scuola Superiore Sant'Anna [6].

2.1 Stato dell'arte

Come già detto sono stati fatti diversi studi sull'anticipazione sensoriale nel controllo. Si presenteranno ora alcuni di questi lavori focalizzati maggiormente sul problema del controllo della locomozione.

Tra le varie ricerche neuroscientifiche, oltre al lavoro portato avanti da Berthoz di cui si è già parlato, hanno molta importanza gli studi effettuati da Grasso e al. [16, 17]. In uno dei loro lavori [17] hanno investigato l'orientamento di testa e occhi durante una camminata lungo traiettorie che girano attorno ad un angolo di 90 gradi. I test sono stati fatti su sei volontari ai quali è stata fatta percorrere la traiettoria curva in avanti e all'indietro con occhi aperti e bendati. I risultati ottenuti mostrano come lo sguardo si muova in modo anticipatorio verso la direzione che il soggetto ha intenzione di prendere sia con occhi aperti che chiusi. Tutto ciò non accade invece durante i movimenti all'indietro. Questo e precedenti studi di Grasso [16] evidenziano l'importanza della predizione nei sistemi di controllo durante la locomozione.

Dal punto di vista dell'implementazione robotica è interessante lo studio discusso da Hoffmann [20]. L'autore presenta un sistema di anticipazione visuomotoria per l'apprendimento della disposizione spaziale degli ostacoli nell'ambiente. La piattaforma su cui viene implementato il modello è un robot mobile guidato da visione. Per effettuare la predizione percettiva il robot rimane fermo, osserva l'immagine della camera omnidirezionale e da questa calcola la sequenza di immagini predette utilizzando la sequenza di comandi motori e le immagini predette man mano. Grazie a questa predizione il sistema stima la distanza di ostacoli e valuta la posizione di passaggi e vicoli ciechi. Per l'apprendimento viene usata una rete Multilayer Perceptron addestrata attraverso movimenti random del robot all'interno dell'ambiente con varie disposizioni degli ostacoli al suo interno.

Un'altra valida implementazione di un modello anticipativo in un sistema di navigazione è quello presentato da Gross e al. in [18]. Nell'articolo si fornisce un'architettura di controllo neurale utilizzata per controllare localmente una piccola piattaforma robotica mobile per un task di navigazione. L'ambiente in cui lavora il robot è ricco di ostacoli, corridoi e vicoli ciechi. Il sistema di anticipazione predice la sequenza delle possibili risposte sensoriali date da tutte le ipotetiche azioni possibili ed alla fine effettua la migliore di esse. Come informazione sensoriale il sistema utilizza il flusso ottico estratto dalle immagini fornite dal sistema visivo del robot. Il flusso ottico predetto in ogni passo è

ottenuto dal precedente e dall'azione eseguita e grazie ad esso il sistema può anticipare le conseguenze sensoriali di un azione. Il confronto tra il sistema proposto da Gross e uno reattivo mostra come, grazie all'anticipazione dell'input sensoriale, l'architettura proposta da Gross riesca ad evitare gli ostacoli (obstacle advoidance) più velocemente (anticipando il movimento).

Per quanto riguarda i modelli interni e l'EP sono da citare il lavoro di Smith [30] sui modelli anticipativi e quello di Johansson [22] sull'anticipazione nel grasping. Sempre per i modelli anticipativi molto importante è il lavoro di Datteri [11, 10] sull'EP, concetto sul quale si fonda tutto il lavoro presentato in questa tesi. Si basa sul concetto di EP riconducendosi al lavoro di Johansson l'implementazione del sistema di grasping tramite percezione attesa realizzato da Laschi e al. [26]. Di tutti queste implementazioni è stata fatta un esauriente descrizione nello scorso capitolo, quindi non ci si soffermerà oltre.

Interessante è il recente lavoro di Alejandra Barrera e Cecilia Laschi [6] in cui viene presentata un'architettura basata sull'EP Scheme di un sistema di controllo anticipatorio per la locomozione di un robot. Essendo questo lavoro il punto di partenza per gli studi svolti nella presente tesi, lo si analizzerà ora in modo più dettagliato.

2.2 AVP Scheme

L'*Anticipatory Visual Perception (AVP) Scheme [6]* è un'architettura ad alto livello basata sulla predizione dell'input sensoriale visivo utile per il controllo di un robot umanoide durante la locomozione. La modellazione di questo schema è avvenuta all'interno di un progetto più ampio chiamato RoboSoM (A Robotic Sense of Movement). L'obiettivo principale del progetto è quello di investigare e realizzare nuovi approcci nella progettazione e sviluppo di robot umanoidi con capacità avanzate di percezione ed azione. Tutto ciò deve portare ad un comportamento del robot robusto, adattivo, predittivo ed efficace all'interno del mondo reale. L'intero lavoro si basa sul concetto di *Sense of Moviment* ideato da Berthoz [8]. L'AVP scheme cerca di inserire la predizione all'interno di questo sistema di controllo.

Per come è strutturata l'architettura basata sull'AVP (Figura 2.1), il controllo della locomozione viene gestito dalla combinazione di due cicli Percezione-Azione: uno classico reattivo e l'altro basato sull'AVP. Si procederà adesso con l'analisi di entrambi.

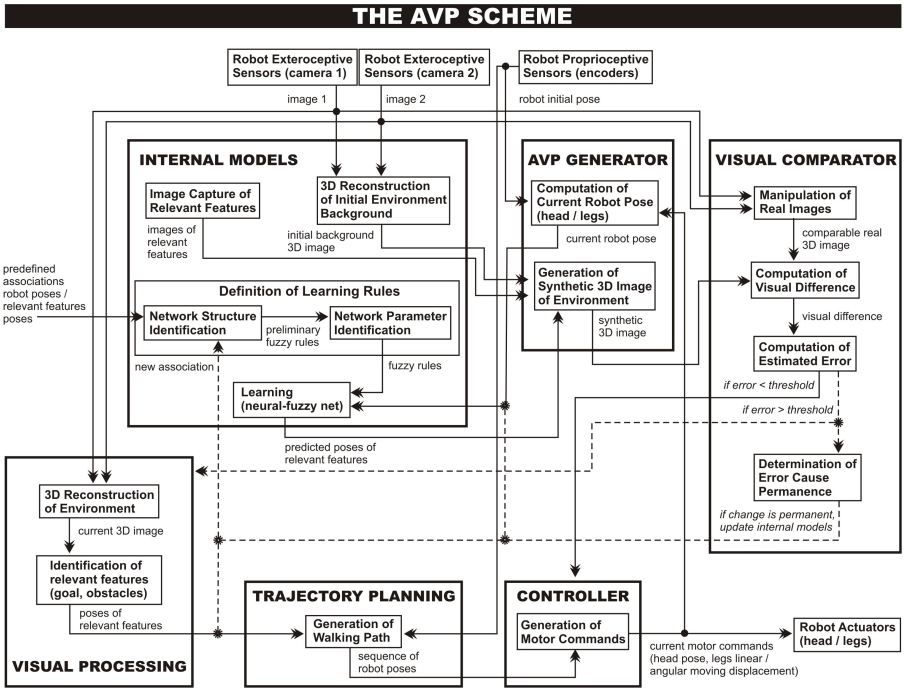

Figura 2.1: L'Anticipatory Visual Perception Scheme, una particolare applicazione dell'EP Scheme nel campo del controllo della locomozione di robot per mezzo della visione.

Il ciclo reattivo è formato da 3 moduli: Visual Processing, Trajectory Planning e Controller. Il primo prende in ingresso le immagini dalle due camere del robot e fornisce in uscita posizione ed orientamento delle feature rilevanti. Questo viene fatto in due passi. In primo luogo viene costruita un immagine 3D dell'ambiente, e poi da questa vengono ricavate le informazioni sulle feature. A questo punto orientamento e posizione delle feature rilevanti sono utilizzati dal modulo Trajectory Planning per generare il percorso che il robot dovrà percorrere per raggiungere il suo obbiettivo espresso tramite una sequenza di posture del robot. Per ultimo il Controller prende la prima posa (orientamento e posizione) dalla sequenza fornita dal Trajectory Planning e, grazie ad essa, produce i corrispondenti comandi motori per la testa e le gambe. Questo ciclo continua fino al raggiungimento dell'obbiettivo.

Si vedrà ora come è strutturato il ciclo basato sull'AVP. Ad ogni passo all'interno del ciclo vengono calcolate un'immagine 3D a partire dai dati provenienti dalle camere e una predizione sintetica dell'immagine 3D del passo successivo (AVP). Fatto questo l'immagine 3D viene comparata con l'AVP calcolato durante il passo precedente. Se l'errore rimane al di sotto di una certa soglia verranno saltati i moduli di Visual Processing e Trajectory Planning e si faranno calcolare direttamente i comandi motori al Controller a partire dalla sequenza di pose precedentemente calcolata. Se invece l'errore supera la soglia il ciclo reattivo verrà eseguito normalmente e verrà ricalcolata la traiettoria. Inoltre se l'errore è dovuto a cambiamenti permanenti dell'ambiente il modello interno del sistema verrà aggiornato (struttura e parametri della rete fuzzy). Questa architettura si basa sul fatto che finché non avvengono cambiamenti imprevisti del mondo o della posizione del robot l'immagine predetta risulterà simile a quella effettiva. In questo caso il robot può continuare lungo la traiettoria precedentemente calcolata senza problemi. Se invece avvengono degli imprevisti, questi risulteranno in un errore nella predizione e quindi sarà necessario ridefinire la traiettoria e, nel caso fosse necessario, nel modello. Anche il ciclo di basato sull'AVP è formato da tre grandi moduli: Internal Models, AVP Generator e Visual Comparator.

Il primo modulo, come dice il nome stesso definisce il modello interno del sistema, consistente sia nell'immagine 3D dello sfondo e le immagini delle feature rilevanti e sia principalmente dal posizionamento predetto di queste ultime. La ricostruzione 3D dello sfondo e le immagini delle feature vengono calcolate dalle immagini delle camere e memorizzate una tantum all'inizio del processo sensomotorio. Durante questo processo si effettua anche il calcolo delle relazioni tra

pose del robot e posizionamento delle feature. Tali relazioni sono utilizzate per definire la struttura della rete fuzzy adoperata per predire, della quale verranno anche settati i parametri di apprendimento. Tale rete verrà poi usata in ogni ciclo per calcolare il posizionamento nel passo successivo delle feature ambientali.

L'AVP generator è il modulo che ha il compito di generare l'immagine 3D sintetica predetta a partire dalla predizione di posizione ed orientamento delle feature, dallo sfondo 3D iniziale e dalle immagini delle feature ambientali. Per fare questo, per prima cosa, utilizza le informazioni sensoriali propriocettive e il comando motorio attuale per calcolare la posa del robot corrente. Quest'ultima viene poi data in ingresso alla rete fuzzy per il calcolo della posa futura delle feature ambientali che verranno utilizzate per ottenere l'immagine sintetica.

L'ultimo modulo è il Visual Comparator che ha il compito ad ogni passo di confrontare l'immagine 3D sintetica e quella estratta dalle camere e decidere se andare avanti con la traiettoria inizialmente predetta, ricalcolarla o addirittura aggiornare i modelli interni. Il suo primo sotto modulo costruisce un'immagine 3D da quelle ottenute dalle 2 camere. Confrontando quest'ultima con l'immagine sintetica 3D predetta calcola la loro differenza dalla quale estrae un errore. Dall'analisi dell'errore e il confronto con una soglia il Visual Comparator deciderà se ricalcolare o meno traiettoria o, nella peggiore delle ipotesi, struttura e parametri della rete.

2.3 Nuovo modello per il calcolo dell'EP

Prendendo spunto dall'architettura presentata da Barrera e Laschi [6], nelle pagine successive si cercherà di progettare, implementare e testare un'architettura per il controllo della locomozione di un robot attraverso un modello interno di tipo EP Scheme [11, 10]. Il motivo di fondo da cui nascono le modifiche apportate allo schema precedente è il fatto di voler evitare di lavorare con camere stereo e dati nelle tre dimensioni, ma si preferirebbe alleggerire i calcoli portando i dati nello spazio 2D. Nel nostro sistema sarà quindi presente una sola camera e l'intero ciclo dell'AVP verrà rivisitato. Prima di entrare in dettaglio nella presentazione dell'architettura è necessario introdurre il concetto di flusso ottico e rete neurale ricorrente. Il motivo di questa digressione è dato dal fatto che la comprensione di tali strumenti è fondamentale per capire struttura e funzionamento del modello.

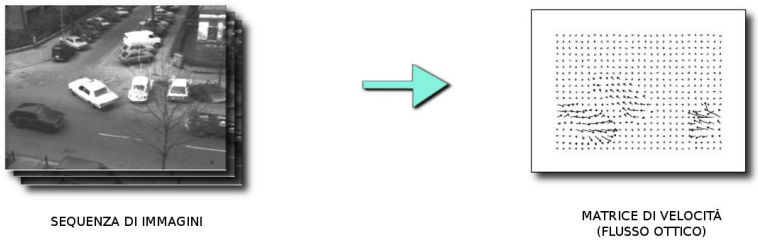

Figura 2.2: Il flusso ottico estratto da una sequenza di immagini

2.3.1 Il Flusso Ottico

Un'informazione riguardante 2 o più immagini consecutive di una sequenza video che può risultare molto utile è la quantità di moto della camera e del mondo che è avvenuta durante la loro cattura. Questo perché spesso il moto stesso indica che sta avvenendo qualcosa di interessante. Quando non si possiede nessuna conoscenza a priori riguardo al contenuto dell'immagine, un valida misura del movimento che avviene nelle immagini è il Flusso Ottico [9].

Il flusso ottico è formato dall'insieme di un qualche tipo di velocità (o una stima dello spostamento) associato ad alcuni precisi punti delle immagini da analizzare. Un modo per calcolare il flusso ottico è quello di tener traccia (tracking) dei punti nelle immagini e calcolane la quantità di spostamento. Quando i punti usati sono tutti i pixel dell'immagine si parla di *flusso ottico denso*, mentre quando vengono scelte solo alcune feature nell'immagine si ottiene il *flusso ottico sparso*.

Nel caso denso il flusso ottico consiste nel campo di velocità (matrice dei vettori) di ogni punto dell'immagine (Figura 2.2). Il problema principale di questo tipo di algoritmi è il fatto che molti pixel non sono facili da riconoscere nelle varie immagini. Ad esempio tutti i pixel appartenenti ad un muro bianco o quelli che fanno parte di uno spigolo parallelo al movimento (problema della finestra). Per risolvere questo problema i metodi densi devono compiere del lavoro di interpolazione tra i punti in postprocessing per risolvere le ambiguità presenti. Tutto ciò ha come risultato l'elevato costo computazionale di questo

genere di flussi ottici.

Per questo e altri motivi spesso si ricorre all'utilizzo dei metodi sparsi per il calcolo del flusso ottico. Gli algoritmi di questo tipo effettuano una fase preprocessing sulle immagini per scegliere i punti più adatti per essere traccati. Sono diverse le caratteristiche scelte per la valutazione dei punti, ad esempio un buon modo di procedere può essere quello di scegliere gli *angoli* (punti di incontro tra gli spigoli delle immagini). Per molte applicazioni pratiche gli algoritmi sparsi risultano essere notevolmente più veloci rispetto a quelli densi.

2.3.2 Le Reti Neurali Artificiali

Il Machine Learning [32] è un campo dell'informatica in cui si cerca di approssimare un funzione obiettivo attraverso la modifica dei parametri liberi di particolari modelli matematici che rappresentano degli spazi di funzioni. Attraverso la modifica di questi parametri gli algoritmi di apprendimento cercano la funzione, all'interno dello spazio definito dal modello, che più si avvicina a quella obiettivo.

Gli algoritmi di apprendimento si dividono in tre gruppi: apprendimento supervisionato, non supervisionati e di rinforzo.

Nei primi durante la fase di apprendimento ogni elemento del dataset presenta i dati di input e la relativa risposta del modello. Nel secondo ogni elemento possiede solamente i dati di input. Questo tipo di algoritmi vengono solitamente utilizzati per fare clustering di dati, feature extraction o riduzione di dimensione dell'input. Nel Reinforcement Learning vi è un sistema di premiazione (o punizione) per ogni determinata azione senza fornire nessuna informazione dettagliata sulla migliore scelta da compiere.

Le Reti Neurali Artificiali (ANN) [19] sono un vasto insieme di modelli di ML la cui particolarità comune è quella di essere formate da una rete di unità di calcolo più piccole ispirate (più o meno a seconda del modello) ai neuroni biologici. Tra le varie distinzioni che si possono fare tra i modelli di reti neurali una è quella tra reti feedforward e reti ricorrenti.

Per quanto riguarda le prime uno dei modelli più utilizzati e versatili sono le MLP (Multilayer Perceptron) [35] che hanno il vantaggio di essere approssimatori universali. Purtroppo le loro limitazioni quali problema del minimo locale e la grossa dipendenza dalle conoscenze a priori per la scelta della topologia della rete, non le rendono la scelta ottimale per diversi task. Le RBF (Radial Basis Function) network [34] risolvono il problema dei minimi locali, ma peccano in

generalizzazione. Le reti costruttive quali la CC (Cascade Correlation) [14] variano la loro topologia durante la fase di apprendimento. Tra le reti feedforward che utilizzano algoritmi di apprendimento non supervisionati vi sono invece le SOM (Self-Organizing Map) [25] e le GNGs (Growing Neural Gas) [15] le prime a topologia fissa, mentre le seconde a topologia e dimensione variabili.

Le reti ricorrenti (RNN) sono particolari modelli di NN che presentano dei loop al loro interno. Questa particolare topologia permette loro di avere come input non solo i dati correnti, ma anche informazioni relative agli stati passati. Questo fatto le rende particolarmente indicate per l'analisi di sequenze (ad esempio la sequenza temporale di immagini di una camera). Oltre le versioni ricorrenti della maggior parte dei modelli precedentemente enunciati, un recente sottogruppo di RNN sono le Reservoir network (ad esempio le Echo State Network (ESN) [21] o le Liquid State Machine (LSM) [29]). Queste reti presentano una rete di neuroni con connessioni sparse e cicliche chiamata Reservoir. Nella loro forma più semplice il Reservoir è totalmente connesso ad un layer di input ed uno di output e gli unici pesi modificabili durante l'apprendimento sono quelli che vanno verso il layer di uscita. La struttura di quest'architettura si basa fortemente sulla proprietà delle RNN di apprendere a priori. Il vantaggio principale di questo tipo di reti è la loro velocità di apprendimento ed esecuzione.

2.3.3 L'architettura e le differenze rispetto all'AVP Scheme

Si inizierà ad analizzare ora l'architettura progettata ed implementata in questa tesi. Il suo scopo è il controllo di un sistema di locomozione basato su di un sistema di visione a camera singola. Le differenze principali che si possono trovare tra l'AVP Scheme e l'architettura presentata in questo lavoro sono varie.

In primis la rappresentazione della percezione attesa. Mentre nell'AVP Scheme l'EP (chiamata in quel caso AVP) consiste in un immagine sintetica 3D del mondo, nell'architettura che ci si accinge a presentare consiste, più semplicemente, in un immagine 2D rappresentante le feature ambientali nella loro posizione futura. Il motivo di questo cambiamento è dato dal desiderio di alleggerire i calcoli effettuati in ogni ciclo, evitando di effettuare un preprocessing eccessivamente pesante sulle immagini stereo delle camere per ricavare un immagine sintetica 3D comparabile con l'EP. Nella soluzione proposta, durante il confronto vengono analizzate l'EP e l'immagine della camera senza alcuna fase di preprocessing, facendo una differenza tra le zone dell'immagine sintetica dove sono presenti le feature e le zone equivalenti dell'immagine della camera.

Questo è, tra l'altro, uno dei motivi principali per il quale si è scelto di portare il numero delle camere da 2 a 1.

La seconda grossa differenza tra i due approcci è il tipo di informazioni dalle quali verrà poi ricavata la percezione attesa. Mentre nello schema di Barrera e Laschi viene utilizzata la predizione delle pose delle feature calcolata a partire da i dati sensoriali propriocettivi e dal comando motorio, nell'architettura di questa tesi viene utilizzato il flusso ottico predetto. La scelta di ricavare l'EP attraverso il flusso ottico è data dal fatto che il flusso ottico è l'informazione sul moto per eccellenza ottenuta da una sequenza di immagini. Siccome la predizione dell'immagine sintetica futura dipende fortemente dal movimento del robot e dell'ambiente che lo circonda, il flusso ottico sembra essere un dato appropriato. In più il flusso ottico, oltre ad essere ricco di informazione riguardo a movimento e posizione, è un campo della computer vision che è stato ampiamente studiato [7]. Esistono al giorno d'oggi un infinità di algoritmi che permettono di calcolarlo, veloci ed adatti a risolvere molteplici task.

L'ultimo cambiamento, anche se forse risulta essere il più importante, riguarda il tipo di modello di sistema di apprendimento usato per effettuare la predizione. Diversamente rispetto a quanto avviene nell'AVP Scheme, dove viene usata una rete neurale fuzzy per predire le pose delle feature rilevanti, nell'approccio che si sta presentando in queste pagine si è scelto di utilizzare le reti neurali ricorrenti (RNN). Quello che rende le RNN adatte a questo problema è la loro capacità di analizzare sequenze, derivante dalla loro struttura ricorrente. Come già detto precedentemente, questa loro qualità è data dal fatto che la loro uscita non dipende solamente dall'input precedente, ma dalla sequenza di input passati. Nel caso della predizione di un immagine sintetica durante il controllo della locomozione di un robot questa caratteristica è molto importante. Basarsi semplicemente sulle informazioni sensoriali esterocettive e propriocettive attuali per il calcolo dell'EP rischia di portarci a situazioni ambigue. Si prenda in considerazione il caso in cui il robot sia arrivato durante due sequenze di spostamento diverse in un punto in cui posizione e orientamento del robot siano identiche. Applicando al robot uno stesso comando motorio nei due differenti casi si otterranno risposte diverse a seconda della sequenza di posizioni che è stata necessaria per raggiungere quel determinato punto (velocità a cui arriva il robot, etc). Lo stesso concetto vale maggiormente per il calcolo del flusso ottico predetto. Per quanto riguarda la fase di apprendimento della rete essa può avvenire sia offline che online a seconda della rete e dell'algoritmo di apprendimento scelti. Nel primo caso essa verrà effettuata utilizzando un

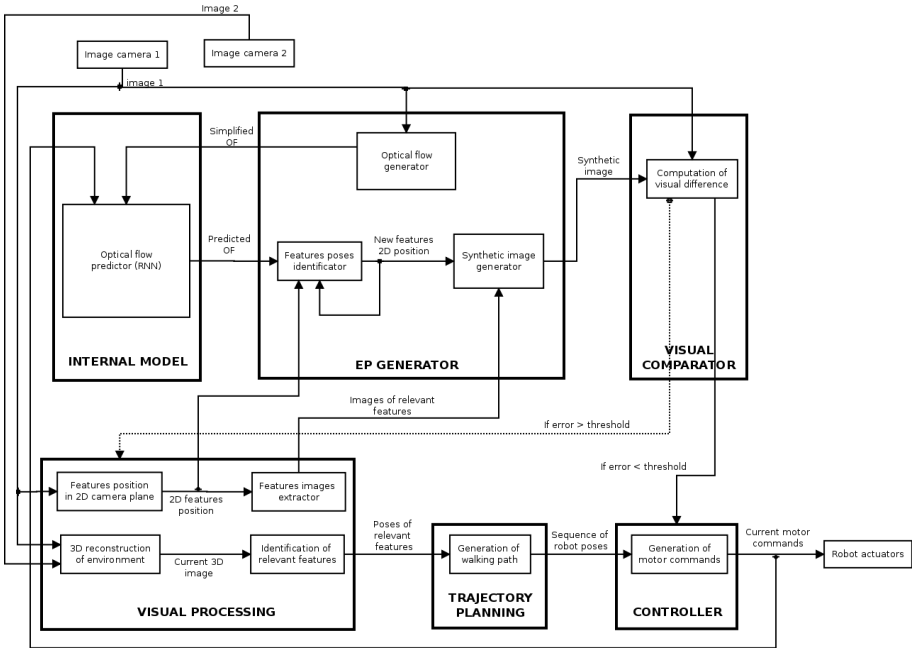

Figura 2.3: Il nuovo EP Scheme per un sistema di controllo della locomozione basato sulla visione

training set che sia rappresentativo di tutte le situazioni in cui il robot si possa trovare e verrà eseguita una sola volta prima di utilizzare il robot. Nel secondo caso la fase di learning potrà essere ripetuta durante il funzionamento della rete. Essendo stato scelto il sistema di apprendimento offline d'ora in avanti si continuerà assumendo questo tipo di soluzione.

Si andrà ad analizzare adesso lo schema che rappresenta questa nuova architettura (Figura 2.3). Come si può vedere il ciclo reattivo è rimasto praticamente invariato, le uniche differenze consistono nel modulo di visual processing. Sono infatti da aggiungere il calcolo della posizione delle feature nel piano 2D dell'immagine e l'aggiornamento delle immagini delle feature visibili. Il metodo con cui viene calcolata la traiettoria non è stato analizzato in questo lavoro e quindi si è deciso di lasciarlo inalterato anche se andrebbe bene qualunque algoritmo che renda una sequenza di comandi motori relativi ad una determinata traiettoria.

Nel ciclo dell'AVP le modifiche sono invece molteplici. L'Internal Model è formato da una sola rete neurale ricorrente. Ad ogni passo tale rete, a partire dal flusso ottico attuale e dal comando motorio, calcola il flusso ottico predetto

(a seconda del metodo scelto per il calcolo della traiettoria, durante una fase iniziale di calibrazione si potrebbe far creare al robot una mappa dell'ambiente).

L'EP Generator presenta tre sotto moduli che lavorano in ogni passo del ciclo. L'OF Generator estrae il flusso ottico dall'immagine della camera e lo passa alla rete. Il Feature Poses Identificator calcola la nuova posizione e dimensione delle feature nell'immagine a partire da quella precedente e dal flusso ottico predetto ed in fine il Synthetic Image Generator posiziona su di un'immagine bianca le immagini delle feature con posizione e dimensione appropriate.

Il Visual Comparator prende in ingresso l'immagine dalle camere e quella sintetica predetta il passo precedente, confronta le zone in cui sono presenti le feature e dalla differenza calcola un errore. A seconda che questo errore sia più piccolo o meno di una soglia il sistema decide se proseguire con la traiettoria precalcolata ed il ciclo di predizione oppure se effettuare un passo del ciclo reattivo.

Ricapitolando il sistema, una volta avvenuta la fase di learning della rete, inizialmente esegue almeno un passo del ciclo reattivo e calcola l'immagine sintetica predetta. Ogni passo successivo predice l'immagine sintetica del passo successivo, se l'immagine predetta nel passo precedente è differente da quella della camera, significa che l'ambiente è stato modificato o il robot non è dove dovrebbe essere e quindi viene rieseguito il ciclo reattivo (Visual Processing e Trajectory Planning). Se, al contrario, le immagini sono simili allora si prosegue con la traiettoria calcolata e si genera l'immagine predetta a partire da flusso ottico predetto e posizione delle feature nell'immagine.

Capitolo 3

Implementazione del modello

Nelle seguenti pagine sarà descritta una particolare implementazione dei moduli *Internal Model* ed *EP Generator* appartenenti all'architettura proposta nel precedente capitolo. Sono stati per prima cosa scelti gli algoritmi e i modelli specifici da utilizzare. Una volta decisi si è passato alla loro implementazione in parte in matlab e in parte in C++. Per effettuare i vari test è stato utilizzato il simulatore di un robot umanoide chiamato iCub. Si analizzerà adesso quali sono queste scelte, il perché sono state fatte e il modo in cui si è deciso di implementarle.

3.1 Scelta dei dettagli implementativi

Per quanto riguarda l'implementazione svolta del modello studiato in questa tesi si è deciso di concentrarsi esclusivamente nella parte dello schema dell'EP Generator e dell'Internal Model. Questo è dovuto al fatto che una ponderata scelta del modello di apprendimento e della metodologia del calcolo del flusso ottico è indispensabile per un funzionamento corretto ed efficace dell'intero sistema. Inoltre le maggiori modifiche apportate allo schema guida dell'AVP si trovano proprio all'interno di questi moduli.

Si cercherà quindi di analizzare meglio i due moduli in questione. Dallo schema presente in queste pagine si possono notare alcune importanti scelte implementative. Prima di tutto la divisione dell'OF Generator in due sotto moduli: uno per il calcolo del flusso ottico dalle immagini della camera e l'altro per fornirne una versione semplificata. Come algoritmo di flusso ottico è stato

scelto il *Pyramid Lucas-Kanade* e si sono disposti i vettori di velocità ottenuti in una matrice delle dimensioni dell'immagine. Questa matrice viene in seguito semplificata dividendola in un numero adeguato di zone e calcolando per ognuna di esse il vettore media delle velocità presenti al suo interno. Il numero delle zone necessarie è stato deciso in relazione ai parametri degli algoritmi scelti e verrà analizzato nel dettaglio nel sotto-paragrafo 3.3.3.

I vettori di flusso ottico medio, più i comandi motori forniti al robot (per i test effettuati in questa sede sono stati utilizzati i comandi in velocità dei giunti del collo), sono i dati di input della rete neurale ricorrente. Essa fornirà in uscita i vettori del flusso ottico predetti del passo successivo. Il tipo di architettura di rete ricorrente scelta è la *Echo State Network* (Sotto-paragrafo 3.1.2). Per quanto riguarda l'apprendimento si è deciso di addestrare la rete a priori tramite un metodo *batch offline*.

Prima di procedere con il motivare le scelte fatte è bene descrivere meglio gli algoritmi e le architetture utilizzate. In questo modo sarà più facile comprendere il perché si è fatta una scelta piuttosto che un'altra.

3.1.1 Lucas-Kanade piramidale

L'algoritmo di *Lucas-Kanade* (LK) [27, 9] è stato un algoritmo denso proposto nel 1981 dai due autori da cui prende il nome. Basandosi, però, su di un metodo locale per calcolare la velocità di ogni pixel (derivandola attraverso una piccola finestra attorno al pixel), è presto stato utilizzato come algoritmo sparso. Il problema principale proveniente dal fatto di ottenere il flusso attraverso piccole finestre locali è dato dall'impossibilità di riconoscere spostamenti ampi che portano il pixel al di fuori della finestra. Per risolvere questa debolezza è stata sviluppata la versione piramidale dell'algoritmo. Nel *Pyramidal Lucas-Kanade* (PLK) si forma una serie di immagini di risoluzione via via più bassa a partire da quella originale (per questo motivo piramidale). L'algoritmo iniza ad analizzare l'immagine con risoluzione più bassa per poi reiterare l'algoritmo con quelle sempre più dettagliate. In questo modo anche gli spostamenti più ampi possono venire riconosciuti, dato che a risoluzione più bassa la finestra contiene una porzione maggiore dell'immagine. Ovviamente tutto ciò avviene a discapito di un maggiore dispendio computazionale. Se, comunque, si utilizza l'algoritmo in modo sparso, il limitato numero di feature consente di ridurre notevolmente il suo tempo di esecuzione.

Si entrerà adesso più nel dettaglio descrivendo i concetti e la teoria che stanno alla base dell'algoritmo.

È possibile affermare che l'intera metodologia per il calcolo del flusso ottico proposta da Lucas e Kanade si basa su tre assunzioni fondamentali:

1. La CONSERVAZIONE DELLA LUMINOSITÀ (brightness constancy) di un pixel rappresentante un punto da un'immagine a quella sucessiva.

2. Il fatto che avvengano PICCOLI SPOSTAMENTI (small movements) tra un'immagine e l'altra. Questo risulta vero fatta l'assunzione che il campionamento sia sufficentemente veloce rispetto alla quantita di movimeno all'interno dell'immagine.

3. La COERENZA SPAZIALE (spatial coherence) tra punti vicini. Si suppone che punti vicini appartengano ad uno stessa superficie e quindi si spostino con movimento simile gli uni con gli altri.

Per capire al meglio il funzionamento dell'algoritmo è utile analizzarlo nel caso di una singola dimensione per poi in seguito estenderlo al 2D dell'immagine. Per iniziare definiamo la seguente equivalenza:

$$f(x,t) \equiv I(x(t),t) \tag{3.1}$$

dove entrambe le funzioni rendono il valore dell'intensità del pixel di cui si intende tener traccia in funzione della sua x (immagini monodimensionali) e dell'istante di tempo t, sebbene in$I()$ anche l'input rappresentante la posizione del punto di cui si esegue il tracking dipenda dal tempo.

Assumendo la *brightness constancy* si può affermare che l'intensità del pixel di cui si tiene traccia non varia nel tempo. Questo può essere espresso ugualiando a 0 la derivata parziale di $I(x(t),t)$ sul tempo:

$$\frac{\partial I(x(t),t)}{\partial t} = 0 \tag{3.2}$$

Svolgendo i calcoli si ottiene:

$$\underbrace{\frac{\partial I}{\partial x}}_{I_x}\underbrace{\frac{dx}{dt}}_{v} + \underbrace{\frac{\partial I}{\partial t}}_{I_t} = 0 \tag{3.3}$$

dove I_x rappresenta la derivata spaziale lungo la prima immagine, v la velocità e I_t la derivata nel tempo tra le immagini. A questo punto si è finalmente arrivati all'equazione per il calcolo della velocità in un punto dell'immagine:

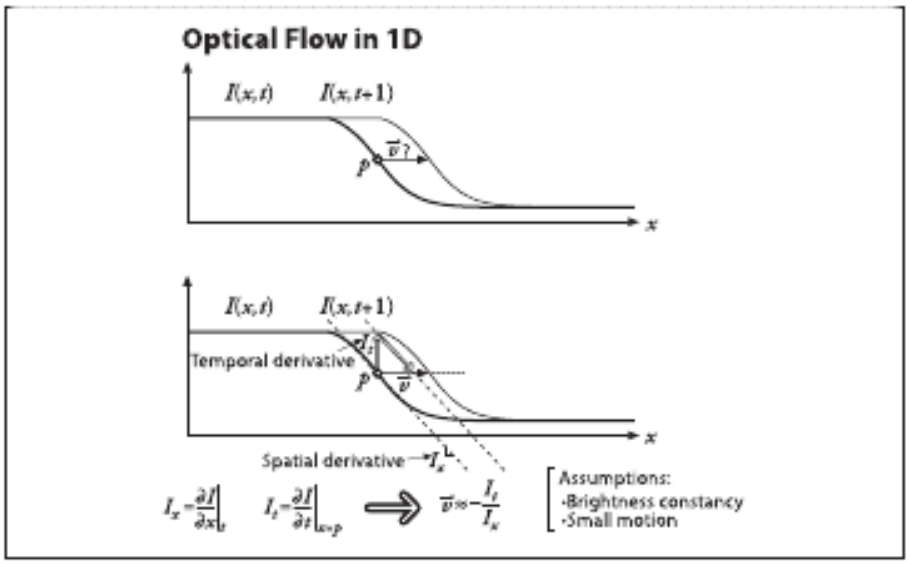

Figura 3.1: Esempio di edge che si sposta su di un'immagine ad una dimensione

$$v = -\frac{I_t}{I_x} \tag{3.4}$$

Per una migliore comprensione, si consulti la raffigurazione geometrica (Figura 3.1) rappresentante uno spigolo (edge) che si muove verso destra lungo le ascisse.

Dato che è impossibile ottenere immagini con luminosità del tutto stabile e che rispettino in ogni momento l'assunzione degli *small movements*, la velocità ottenuta in precedenza risulta essere affetta da un piccolo errore. Per risolvere questo problema viene usato un metodo iterativo per rifinire il risultato. La velocità ottenuta inizialmente viene usata come punto di partenza per riaplicare l'equazione 3.4 fino a che non si raggiunge la precisione desiderata. Se la prima valutazione risulta "abbastanza vicina" alla soluzione cercata allora l'agoritmo convergerà in pochi passi.

Se si volesse estendere questo procedimento alle 2 dimensioni il passaggio più naturale sarebbe quello di aggiungere l'asse delle ordinate al ragionamento precedente ottenendo:

$$I_x u + I_y v + I_t = 0 \tag{3.5}$$

Questa equazione presenta, però, due incognite per ogni pixel, fatto che la rende sotto-determinata. Per risolvere questo problema ci si deve appellare all'assunzione della *spatial coherence.* Grazie a ciò è, infatti, possibile asserire che tutti i pixel all'interno di una piccola finestra centrata sul punto da seguire si muovano alla stessa velocità di quest'ultimo. Se si prende una finestra di dimensione n piccolo e si estende , per tutti i pixel interni a tale finestra, l'equazione 3.5 si ottiene il seguente sistema:

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}}_{A^{n\times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d^{2\times 1}} = -\underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{bmatrix}}_{b^{n\times 1}} \tag{3.6}$$

Si può, a questo punto, risolvere il sistema 3.6 sovra-determinato attraverso l'uso della pseudoinversa che permette di trovare la soluzione che minimizza l'errore quadratico $min\,\|Ad-b\|^2$. Il vettore di velocità del pixel sarà quindi ottenuto grazie alla seguente espressione:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\left(A^{\mathrm{T}}A\right)^{-1}A^{\mathrm{T}}}_{pseudoinversa}b \tag{3.7}$$

L'unico vincolo da rispettare è che $\left(A^{\mathrm{T}}A\right)$ sia invertibile e perché accada ciò deve avere rango pieno. Si può affermare, anche se non verrà dimostrato, che le finestre che presentano al loro interno dei corner generano una matrice $\left(A^{\mathrm{T}}A\right)$ con le caratteristiche richieste. Per questo motivo, un approccio in cui si preprocessa l'immagine per estrarre le feature adatte ad essere tracciate e in seguito si calcola il flusso ottico di queste ultime tramite LK, risulta essere un ottimo modus operandi.

Come già detto, però, è impensabile che, con camere comuni che campionano a 30 Hz (troppo lentamente), le 2 assunzioni riguardanti piccoli spostamenti e coerenza spaziale continuino a valere. Grazie all'approccio piramidale del PLK questi problemi vengono superati.

3.1.2 Echo State Networks

Le *Echo State Network*s (ESN) [21] consistono in un'architettura per le reti neurali ricorrenti (RNN) e in un principio supervisionato di apprendimento.

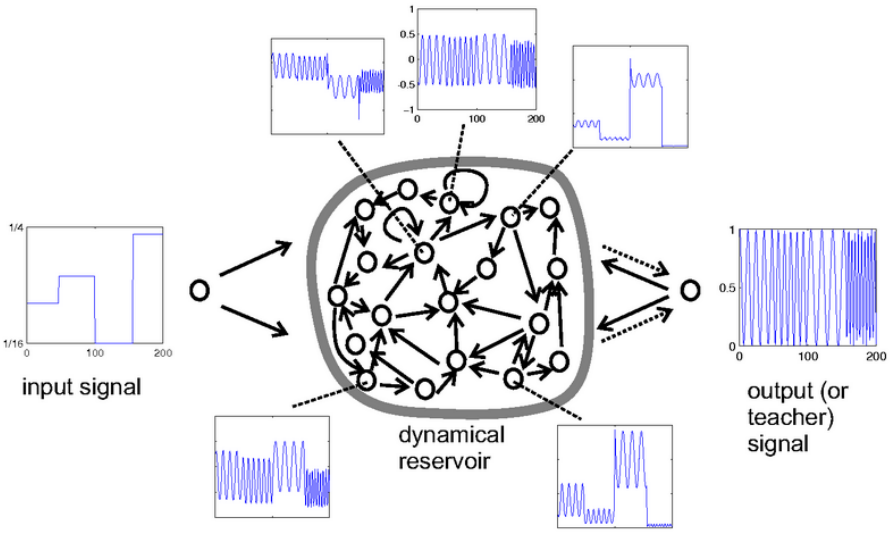

Figura 3.2: Struttura generica di un'Echo State Network

L'idea di fondo è quella di guidare con il segnale d'ingresso una grande rete ricorrente con pesi fissati e random, facendo in modo che ogni nodo fornisca una differente risposta non lineare. A questo punto si ottengono una o più uscite che sono il risultato di un combinazione lineare di tutte le risposte della rete ricorrente. Il punto cruciale di quest'architettura è il fatto che gli unici pesi addestrabili della rete sono quelli di output del combinatore lineare.

Topologicamente (Figura 3.2) la rete è formata da un layer di ingresso, un grosso insieme di neuroni connessi tra di loro in modo random (reservoir), ed un layer di uscita. Il layer di ingresso è totalmente connesso al reservoir. Quest'ultimo si può considerare come una rete ricorrente grazie ad i loop formati dalle sue connessioni random. Per finire il reservoir è totalmente connesso al layer esterno e viceversa (feedback connections), sebbene queste ultime potrebbero essere presenti o meno a seconda del task che si cerca di risolvere. Il motivo che rende interessante questo tipo di architettura è il fatto che le uniche connessioni a venire addestrate nella fase di apprendimento sono quelle che vanno dal reservoir al layer di uscita. Tutte le altre hanno pesi fissi che vengono inizializzati con piccoli valori random.

A validare questo tipo di approccio (oltre ad alcune proprietà teoriche che si mostreranno in seguito) vi sono gli studi di Schiller e Steil [39]. Questi ultimi hanno mostrato che nei metodi tradizionali di addestramento delle RNN, dove

vengono aggiornati tutti i pesi, le variazioni più significative avvengono nei pesi del layer di output. Nel campo delle neuroscienze è stato studiato un comportamento simile da Peter F. Dominey [12, 13] nel cervello dei mammiferi, specialmente nel riconoscimento vocale da parte degli esseri umani.

Tutte le architetture che si basano su questa idea vengono riunite nel nome di Reservoir Computing. A questa classe appartengono tra gli altri le *Liquid State Machines* [29](un'architettura equivalente alle ESN che è stata sviluppata simultaneamente da Wolfgang Maass) e la *Backpropagation Decorrelation* [39].

Equazioni del sistema

A questo punto verrà data una definizione più formale dell'architettura in modo da comprenderne al meglio struttura e funzionamento. Si inizia con il mostrare l'equazione di aggiornamento dello stato interno per un'ESN basilare con neuroni aventi come funzione di attivazione una sigmoide, con il reservoir di dimensione N, K neuroni di input e L di output:

$$\mathbf{x}(n+1) = f(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n)) \tag{3.8}$$

Nell'equazione 3.9 $\mathbf{x}(n)$ rappresenta lo stato del reservoir di dimensione N, f è una funzione sigmoide, $\mathbf{W}$ è la matrice $N \times N$ dei pesi del reservoir, $\mathbf{W}^{in}$ è la matrice $N \times K$ di pesi di input, $\mathbf{u}(n)$ è il segnale di input di dimensione K, $\mathbf{W}^{fb}$ è la matrice dei pesi delle feedback connections di dimensione $L \times N$ e $\mathbf{y}(n)$ è il segnale di output di dimensione L.

L'equazione di aggiornamento del layer di output è, invece, la seguente:

$$\mathbf{y}(n) = g(\mathbf{W}^{out}\mathbf{x}(n)) \tag{3.9}$$

dove g è la funzione di attivazione dell'output (solitamente identità o sigmoide) e $\mathbf{W}^{out}$ è la matrice dei pesi di output di dimensione $L \times N$.

Algoritmo di apprendimento

Come è stato già detto in precedenza gli unici pesi aggiornati durante la fase di apprendimento sono quelli del layer esterno. Sebbene si possa usare qualsiasi tipo di algoritmo di apprendimento per reti single layer, un buon modo di procedere risulta quello di utilizzare dei metodi diretti (non iterativi) per trovare i pesi migliori per minimizzare l'errore tra uscita della rete ed uscita desiderata.

Si prenda, ad esempio, una sequenza di apprendimento:

$[\mathbf{u}(1), ..., \mathbf{u}(n_{max}); \mathbf{d}(1), ..., \mathbf{d}(n_{max})]$ (dove $\mathbf{d}(n)$ è il vettore di dimensione L reppresentante l'output desiderato al tempo n) con la quale si guidi la rete ottenendo la sequenza di stati interni $\mathbf{x}(1), ..., \mathbf{x}(n_{max})$ ricavata attraverso l'equazione 3.8. Si tenga in considerazione che durante la generazione degli stati interni , nel caso di feedback connections, si dovrà sostituire $\mathbf{d}(n)$ al posto di $\mathbf{y}(n)$ all'interno dell'equazione 3.8 (teacher forcing). Si formi a questo punto una matrice $\mathbf{S}$ di dimensione $n_{max} \times N$ con uno degli stati interni precedentemente ottenuti in ogni riga. Nello stesso modo si formi una matrice $\mathbf{D}$ di dimensione $n_{max} \times L$ con in ogni riga un vettore di output desiderato della sequenza $\mathbf{d}(1), ..., \mathbf{d}(n_{max})$. Un modo per ottenere i pesi di uscita aggiornati minimizzando l'errore quadratico (LSE) è il risolvere il seguente sistema lineare:

$$\mathbf{W}^{out}\mathbf{S}^{\mathrm{T}} = \mathbf{D}^{\mathrm{T}} \tag{3.10}$$

Questo è possibile utilizzando la pseudoinversa $\mathbf{S}^{-1}$di $\mathbf{S}^{\mathrm{T}}$:

$$\mathbf{W}^{out} = \mathbf{S}^{-1}\mathbf{D}^{\mathrm{T}} \tag{3.11}$$

Proprietà della rete

Perché il principio alla base delle ESN funzioni, il reservoir deve possedere la *echo state property* (ESP). Detto in modo informale, l'ESP dice che il reservoir si ripulirà asintoticamente da qualsiasi informazione riguardante le condizioni iniziali. Questo vuol dire che se si guida la rete con una sequenza di training, dopo un certo numero di passi iniziali lo stato interno (le uscite dei neuroni del reservoir) dipenderanno esclusivamente dall'input e non più dallo stato iniziale. Per questo motivi prima di far apprendere o eseguire la rete si lascia passare un certo numero cicli iniziali. È stato dimostrato in modo empirico [21] che l'ESP è garantita per reservoir con neuroni aventi funzione di attivazione sigmoide e con matrice dei pesi con raggio spettrale (valore assoluto del massimo degli autovalori) inferiore ad 1.

Altra importante proprietà delle ESN è quella di poter realizzare con precisione arbitrariamente buona qualsiasi filtro non lineare con memoria limitata [29, 28].

Significato pratico dei parametri liberi

Le ESN presentano diversi parametri liberi che è necessario settare per garantire il corretto funzionamento della rete. Si procederà ora con una breve carrellata di questi parametri e dell'influenza che essi hanno nel comportamento della rete.

- Dimensione reservoir: Solitamente la dimensione della rete è strettamente collegata in modo proporzionale con la complessità della dinamica da apprendere. Bisogna però anche guardarsi dall'overfitting (scarsa generalizzazione della rete dovuta ad un apprendimento troppo preciso del training set) che potrebbe avvenire se la dimensione della rete viene scelta eccessivamente grande.

- Raggio spettrale: Il valore scelto per il raggio spettrale della matrice dei pesi del reservoir è strettamente collegato con la momoria della rete (numero di informazioni precedenti della sequenza da cui l'uscita della rete dipende). Più è alta la memoria che necessita il task che ci si appresta ad affrontare, più il valore del raggio spettrale deve avvicinarsi all'unità. Più è bassa la memoria richiesta, più il raggio spettrale deve essere piccolo. Inoltre, più ci si avvicina all'unità e minore è l'ampiezza di valori ottimi per il settaggio.

- Scalatura dell'input: Il grado di scalatura dell'input determina il grado di nonlinearità della dinamica dello stato interno. Se si sceglie un valore troppo piccolo il reservoir rende una risposta quasi lineare rispetto all'input, se invece si sceglie un valore troppo grande, le sigmoidi raggiungono la saturazione e il reservoir si comporterà con una dinamica con variazioni binarie.

Per comprendere meglio il modo di utilizzare questo tipo di reti si ricapitoleranno ora i passi principali dell'implementazione ed esecuzione di un'ESN. Per prima cosa si prende un insieme di neuroni connessi tra di loro in modo random e si inizializzano i pesi con piccoli valori casuali. Si modifica la matrice dei pesi in modo da portarne il raggio spettrale al valore desiderato (sotto l'unità). Si sceglie una sequenza di traning e la si utilizza per aggiornare i pesi del layer di uscita della rete (in modo online o batch a seconda dell'algoritmo utilizzato), ricordandosi di non utilizzare le uscite relativa ai primi valori della sequenza con cui si sta addestrando la rete (ESP). Si sceglie una sequenza di test con cui guidare la rete, ricordandosi anche in questo caso di scartare i primi valori.

3.1.3 Motivazioni

Adesso che si ha un'idea generale delle caratteristiche sia dell'algoritmo di flusso ottico che del modello di rete neurale ricorrente utilizzati si può passare al motivare il loro utilizzo.

Per quanto riguarda l'algoritmo di Lucas-Kanade i motivi sono vari. Innanzi tutto la necessità di dover interagire con il mondo e quindi dover realizzare un sistema che riesca a mantenere una velocità adeguata a portato a preferire un approccio sparso piuttosto che uno denso. Infatti nei test effettuati il flusso ottico doveva essere calcolato ad una frequenza al più di 30 Hz, essendo questa la velocità di campionamento della camera simulata. Scegliendo un algoritmo denso non sarebbe stato possibile garantire un simile requisito. Il fatto che sia sparso permette, inoltre, di eliminare il rumore di fondo intrinseco negli algoritmi densi, dato che si scelgono a priori le migliori feature su cui eseguire il tracking (i corner e gli edge dell'immagine). I metodi densi d'altro canto forniscono una copertura completa dell'immagine, evitando l'insorgere di parecchie zone prive di flusso. Facendo una riflessione sui vantaggi e gli svantaggi offerti da una classe piuttosto che l'altra si è scelto di optare per quelli sparsi. Il fatto di aver utilizzato proprio l'algoritmo di Lucas-Kanade deriva invece da differenti fattori. Primo fra tutti la sua precisione rispetto ad altri algoritmi di flusso ottico [7]. Secondo la sua popolarità e frequente impiego per risolvere vari task. E ultimo, ma non meno importante, la sua facile implementazione. Il Lucas-Kanade è forse l'algoritmo che si può trovare più facilmente implementato nelle librerie specializzate di vari linguaggi. Le OpenCV (libreria C++ usata per implementarlo in questo lavoro) forniscono, infatti, sia funzioni per il ritrovamento delle feature da utilizzare che per il calcolo del flusso ottico tramite questo algoritmo.

Il fatto di ottenere un flusso ottico sparso con diversi punti privi di vettori porta a dei dati di input per la rete ricchi di valori nulli in praticamente tutte le occasioni. Inoltre l'utilizzo di una matrice di vettori della dimensione dell'immagine come flusso ottico ha come effetto l'elevata dimensione di dell'input con conseguente difficoltà di apprendimento della rete. Per risolvere questi due problemi si è deciso di fare un preprocessing del flusso ottico in modo da ridurne la dimensione. Da qui deriva la scelta di dividere la matrice di vettori in zone di uguale dimensione e fornire alla rete come input i vettori rappresentanti la media delle velocità in ogni zona. Questo soluzione ha come vantaggi, inoltre, sia di mantenere le informazioni spaziali riguardo al flusso e sia un buon rapporto tra

il numero di input rappresentanti il flusso ed il numero di quelli rappresentanti i comandi motori.

Si arriva adesso a parlare del modulo centrale di tutta l'architettura: il modello per la predizione. Come detto in precedenza la scelta è ricaduta su di un tipo di rete ricorrente chiamato Echo State Network. Le caratteristiche principali di questo tipo di reti sono i motivi che le hanno portate ad essere la soluzione adottata nel nostro caso. Per iniziare, la natura ricorrente del Reservoir permette alla rete di fornire una predizione del flusso non solamente dipendente dalle informazioni del passo precedente ma dalla sequenza di flussi e comandi motori avvenuti fino a quel momento. A differenza di altre architetture di reti ricorrenti, le ESN hanno il vantaggio di avere una topologia fissa o quasi, un facile settaggio dei parametri, essere veloci nell'esecuzione e nell'apprendimento e di essere di facile implementazione. La caratteristica di avere una topologia fissa permette a chi le utilizza di concentrarsi maggiormente sul problema da risolvere limitandosi semplicemente a scegliere se utilizzare o no un feedback layer o se aggiungere connessioni dirette tra input e output. La velocità della rete è, invece, un dato fondamentale per il task che si cerca di risolve in queste pagine. Come già detto, il ciclo di controllo del sistema, per essere sincrono con le immagini provenienti dalla camera, deve essere almeno di 30 Hz. Essendo la generazione dell'immagine sintetica un processo ricco di sotto moduli, il carico computazionale apportato da ognuno di essi è meglio che sia il minore possibile. Per questo la velocità di esecuzione tipica delle ESN è così importante. Molto utile è anche la loro velocità di addestramento. Infatti, se si volesse far apprendere la rete anche durante il funzionamento del sistema e non solamente a priori, la velocità di addestramento diventa cruciale.

Per finire bisogna dire che la scelta di utilizzare i comandi in velocità dei giunti del collo come comandi motori è data dal fatto che in questa tesi si è cercato di investigare maggiormente il comportamento della rete nella relazione tra movimenti di rotazione e configurazioni dell'ambiente. Si è ritenuto che questo genere di test fosse adatto per uno studio iniziale del problema.

3.2 Strumenti utilizzati per l'implementazione

Prima di passare ad una descrizione dettagliata dell'implementazione del sottosistema di calcolo e predizione del flusso ottico, si procederà con una carrellata degli strumenti utilizzati per realizzarla. Si inizierà con una presentazione delle

librerie usate per il calcolo del flusso ottico (OpenCV), il toolbox di matlab servito per implementare l'ESN e il simulatore sul quale si sono testati i risultati.

3.2.1 OpenCV

Le OpenCV [1], come si comprende facilmente dal nome, sono delle librerie per la Computer Vision open source scritte in C e C++. Sono state pensate per essere multipiattaforma e possono quindi essere usate su Linux, Windows e Mac OS X. Sono, inoltre, in corso di sviluppo interfacce per Python, Matlab ed altri linguaggi.

Lo scopo principale per cui sono sono state sviluppate è quello di fornire uno strumento di semplice utilizzo per lo sviluppo di applicazioni di Computer Vision. Le OpenCV permettono di realizzare, in modo semplice e veloce, applicazioni basate sulla visione piuttosto soffisticate. Queste librerie sono state scritte con codice ottimizzato che le rende computazionalmente efficienti. Gli sviluppatori si sono focalizzati sul renderle il più possibile adatte alle applicazioni in tempo reale.

Le OpenCV contengono diverse funzioni che implementano i più comuni algoritmi di vari campi della visione come analisi di immagini mediche, sicurezza, interfacce utente, visione stereo, robotica, etc... Contengono, inoltre, una discreta libreria di Machine Learning (MLL) che fornisce tutto il necessario per creare applicazioni di base in questo settore. Questa sottolibreria è stata pensata maggiormente come supporto ad applicazioni di computer vision e contiene al suo interno l'implementazione degli algoritmi più adatti a tale scopo.

Le OpenCV si dividono in 4 sottolibrerie principali, piu una (CvAux) contenente sia aree di studio abbandonate che algoritmi sperimentali. Ecco elencate le 4 sottolibrerie:

CV: Contiene tutto ciò che serve per l'image processing di base e gli algoritmi di alto livello di Computer Vision.

ML: Contiene le principali funzioni di base del machine learning, specializzandosi maggiormente in classificatori statistici e strumenti per la clusterizzazione.

HighGUI: Contiene le routine di I/O e le funzioni per la memorizzazione e il caricamento di video e immagini.

CXCore: Contiene le strutture dati e le funzioni di base necessarii per il funzionamento dell'intera libreria.

Il flusso ottico nelle OpenCV

Tra i vari task relativi alla computer vision per cui le OpenCV forniscono un ottima serie di strumenti vi è l'estrazione del flusso ottico. La libreria open surce, infatti, presenta al suo interno numerose funzioni che implementano sia algoritmi sparsi (pyramid Lucas-Kanade) che densi (Lucas-Kanade, Horn-Schunck, etc...).

Dato che si è deciso, all'interno dell'architettura proposta, di implementare il blocco di generazione del flusso ottico con l'algoritmo PLK, adesso verrà analizzata più nel dettaglio la funzione delle OpenCV che lo implementa. Il nome di questa funzione è *cvCalcOpticalFlowPyrLK ()*:

```
void cvCalcOpticalFlowPyrLK
(
        const CvArr      *imgA,
        const CvArr      *imgB,
        CvArr              *pyrA,
        CvArr              *pyrB,
        CvPoint2D32f    *featuresA,
        CvPoint2D32f    *featuresB,
        int                      count,
        CvSize             winSize,
        int                      level,
        char               *status,
        float              track_error,
        CvTermCriteria  criteria,
        int                      flags
);
```

Questa funzione, a partire dalla coppia di immagini imgA e imgB, calcola lo spostamento dei punti di imgA presenti in featuresA e ne memorizza in featuresB la loro posizione di arrivo in imgB. Gli elementi pyrA e pyrB sono buffers allocati per registrarci sopra le immagini della piramide. Il campo count rappresenta il numero di feature di cui tener traccia presenti all'interno di featuresA. Come si può facilmente intuire, winSize è la dimensione della finestra usata dall'algoritmo, mentre level contiene il numero di livelli della piramide. Il campo status è un array di dimensione uguale a featuresA che contiene 0 se il corner corrispondente di featuresA non è stato ritrovato nella seconda immagine o altrimenti contiene 1. Il vettore track_error contiene, per ogni corner, la

differenza tra l'insieme di pixel attorno a lui in imgA e tra quelli attorno al suo punto di arrivo in imgB.

In supporto a questi algoritmi, le OpenCV forniscono anche funzioni per la selezione di feature sul quale eseguire il tracking. Questo scopo ha la funzione *cvGoodFeatureToTrack ()* che trova i pixel dell'immagine in cui sono presenti "corner" o simili. Molti algoritmi sparsi per il calcolo del flusso ottico richiedono, però, una precisione maggiore del singolo pixel nella posizione delle feature. Per risolvere questo problema le OpenCV possiedono la funzione *cvFindCornerSubPix ()* grazie alla quale è possibile, a partire dall'immagine e dall'elenco dei pixel contenenti i corner estratti in precedenza, trovare una posizione più precisa dei corner stessi.

3.2.2 Toolbox Matlab per le ESN

Per quanto riguarda l'implementazione delle ESN si è deciso di usare un semplice, ma potente, toolbox per Matlab sviluppato da H. Jaeger (inventore del modello). Questo software permette di implementare, in modo veloce ed intuitivo, delle ESN di discreta complessità in codice matlab. Con il toolbox è infatti possibile scegliere sia i parametri più banali (dimensione di output input e layer interno, raggio spettrale, input ed output scaling/shifting), sia alcuni un po' più particolari (algoritmi di learning, apprendimento offline od online, tipo di neuroni utilizzati, funzione di attivazione delle varie unità). Il toolbox fornisce, inoltre, alcune funzioni per la visualizzazione dei risultati e il calcolo dell'errore.

3.2.3 iCub Simulator

All'interno della presente tesi, per testare le prestazioni dell'implementazione descritta più avanti, si è deciso di utilizzare il simulatore open source per il controllo della piattaforma robotica iCub [37, 38]. La sua natura open source, facilità di utilizzo ed il fatto che simuli il comportamento di una piattaforma robotica ampiamente utilizzata in ambito di ricerca universitaria (ciò la rende facilmente disponibile per futuri test sul robot in ambiente non simulato), lo hanno reso un'ottima scelta per valutare i risultati del lavoro descritto fin'ora. L'interfaccia tra robot e controllore avviene per mezzo di un'architettura software chiamata YARP. Il simulatore utilizza la stessa architettura per controllare il robot nell'ambiente simulato. Si procederà, adesso, con una descrizione delo stesso YARP, della piattaforma robotica iCub e del suo simulatore.

YARP

Uno dei problemi di base che si presentano quando si cerca di controllare una piattaforma robotica complessa è dato dall'elevato numero di sensori e attuatori da gestire. Le gestione dell'interazione tra i vari dispositivi appartenenti al robot e il sitema di controllo risulta, per questo motivo, per nulla banale. Questo problema si presenta in modo particolare nei robot umanoidi, dove l'elevata quantità di gradi di libertà (con rispettivi sensori ed attuatori necessari per la loro realizzazione) e la complessità dei task di controllo da realizzare rendono la loro interazione molto più complessa. YARP [4] è un'architettura software che consiste in un sistema per gestire e semplificare questa interazione.

Librerie, protocolli e strumenti da cui è composto YARP garantiscono una completa indipendenza tra moduli di controllo e dispositivi da controllare. Questa è una caratteristica molto importante, che permette di scollegare il software dall'hardware. La robotica è un campo di ricerca in continua e rapida espansione, nel quale capita spesso di dover aggiornare le componenti hardware dell'intera piattaforma o parte di esse. Quindi è poco raccomandabile realizzare del software fortemente dipendente da librerie e dispositivi specifici, dato che si corre il rischio che diventino obsoleti in breve tempo. Grazie a YARP i software di controllo implementati risultano più stabili e longevi dato che è possibile cambiare dispositivi alla piattaforma, quali sensori e ed attuatori, apportando minime modifiche al codice.

Dato l'ammontare di dati da gestire in tempo reale durante il controllo di una piattaforma robotica, la richiesta di risorse computazionali da parte del software adibito a questo compito è molto elevata. Spesso ci si trova con la necessità di utilizzare una rete di computer che si spartiscano le elaborazioni più pesanti tra loro. Per questo motivo YARP permette di gestire le informazioni provenienti dai vari dispositivi indipendentemente dal tipo di macchina dal quale provengono, attraverso un'accurata gestione dell'interazione tra processi comunicanti.

YARP è, inoltre, multipiattaforma ed è compatibile con tutte le maggiori architetture di sistema operativo (Windows, MacOS, Linux e Solaris).

Se a tutto questo aggiungiamo i vantaggi dati dalla sua natura open source, è facile capire come quest'architettura sia adottata da molti ricercatori in robotica.

La libreria di YARP è composta da tre componenti:

libYARP_OS: È l'insieme di librerie dedite alla gestione del flusso di dati per mezzo di thread, che permettono di gestire il controllo di diverse macchine

aventi ciascuna il proprio sistema operativo.

libYARP_sig: Sono le libreria atte alla gestione del processamento dei segnali (audio, video, etc...) e dell'interazione con altre librerie comunemente utilizzate in questo campo, quali ad esempio le OpenCV.

libYARP_dev: Le librerie che permettono di interfacciarsi con i dispositivi più comunemente utilizzati in robotica (telecamere, schede di controllo per motori, etc...).

Questi componenti sono mantenuti separati. Il nucleo del sistema è libYARP_OS, che deve essere disponibile prima che gli altri componenti possano essere usati. Per le operazioni in tempo reale YARP utilizza una rete isolata ed è consigliabile che resti separata dalla rete globale. I dispositivi hardware sono spesso dipendenti dal sistema operativo che supportano. Quindi libYARP_dev è strutturata per interfacciare facilmente quel particolare dispositivo e per proteggere il resto del sistema da quel particolare codice in modo da rendere agevole il cambio dell'hardware. Si analizzarà di seguito come la libreria gestisce le comunicazioni tra i vari processi.

La comunicazione in YARP segue il pattern Observer. Lo stato degli oggetti porta può essere assegnato ad un certo numero di observers, in un certo numero di processi in esecuzione su macchine diverse. YARP [36] gestisce queste connessioni in modo da isolare l'observed dall'observer e cosa più importante isola gli observer tra loro. In YARP una porta è un oggetto attivo che gestisce connessioni multiple per una data unità di dati sia in ingresso che in uscita. Ogni connessione ha uno stato che può essere manipolato tramite comandi esterni che gestiscono la connessione o ottengono informazioni sullo stato da essa. Le porte possono essere di input o di output. Una porta di input può ricevere dati da connessioni multiple a differenti data rate utilizzando protocolli diversi. Una porta di output può spedire dati a diverse destinazioni con diverso data rate e usando diversi protocolli. Vengono creati canali di servizio temporanei per l'handshaking tra porte, in tal caso il protocollo utilizzato è il TCP. L'uso di diversi protocolli permette di sfruttare le loro migliori caratteristiche.

Le porte possono essere connesse a programma o a runtime. La comunicazione è totalmente asincrona e la consegna dei messaggi non è garantita a meno di non prevedere particolari disposizioni. Il comportamento di default delle porte di YARP è quello di trattare con messaggi inviati continuamente, dove la perdita di uno di essi non compromette l'integrità del sistema. Si tratta

di una caratteristica di dati quali immagini e suono, dove è molto più importante "tenere il passo" valutando i dati attuali che processare ogni bit ricevuto. Un tipico esempio è l'acquisizione di immagini da telecamera e la consegna a molte macchine che eseguono la trasformazione in parallelo. I processi più lenti possono semplicemente non usufruire di tutti i frame disponibili dello stream perdendone alcuni.

Le porte hanno nomi simbolici e sono gestite nella rete dal name server. Il name server mappa questi nomi (stringhe) in una tripla composta da indirizzo IP, numero di porta e nome dell'interfaccia. Queste informazioni sono le uniche richieste per stabilire una comunicazione socket tra due terminali. Una descrizione della topologia della rete è mantenuta staticamente nella server table ed usata per rispondere alle registrazioni o alle richieste di connessione dei clienti. La prima operazione che ogni porta deve effettuare è la registrazione del suo nome nella server table. La registrazione è di norma seguita dalla connessione ad una porta che usi lo stesso data type. Le porte possono gestire qualsisasi tipo di dato. Per i tipi di dato semplice (quelli che non contengono puntatori) la classe della porta è provvista di un appropriato codice di comunicazione. I tipi complessi sono trattati specializzando le porte con i template C++ per il nuovo tipo e provvedendo alle funzioni di serializzazione e di deserializzazione. La serializzazione è realizzata utilizzando liste di blocchi di memoria per memorizzare le copie. Le porte sono di norma istanziate con un tipo preciso.

Ad esempio, se si volessero ricevere immagini, si potrebbe creare una porta di input all'interno del programma attraverso le seguenti istruzioni:

```
YARPInputPortOf<ImageOf<PixelRgb> > in;
in_port.Register ("/my_in_port");
```

Il passo successivo è registrare la porta al server name con il nome arbitrario "/my_in_port". Un mittente ipotetico creerà una porta nel modo seguente:

```
YARPOutputPortOf<ImageOf<PixelRgb> > out_port;
out_port.Register ("/my_out_port");
```

Questa è una porta di output che utilizza il protocollo di default (TCP), in alternativa il tipo di protocollo è determinato dalla porta di output. Questo perché quella di input può ricevere attraverso qualsiasi tipo di protocollo. Il passo successivo è attendere quindi i dati in arrivo sulla porta

```
if (in_port.read ())
{
```

```
        IPLImage img = in_port.content ();
}
```

In questo modo si effettua una read bloccante sulla porta e si acquisiscono i dati in ingresso attraverso il metodo content. Se la chiamata a read ha successo, allora l'oggetto ritornato dalla seguente chiamata a content sarà il dato ricevuto, ed è garantito che questo non venga modificato né sovrascritto sino alla successiva chiamata a read. Se, nel frattempo, un nuovo dato viene inviato sulla porta, il trattamento di quest'ultimo dipenderà dalla politica di buffer utilizzata dalla porta. Ad esempio il dato può essere salvato all'interno di un apposito buffer e quindi reso disponibile dopo la successiva chiamata a read.

Per concludere si andrà a descrivere come viene gestito l'image processing all'interno di YARP. L'efficienza in questo settore è il parametro più importante, percui è stato definito un approccio basato su un' interfaccia verso le librerie maggiormente "performanti". Per aiutare gli sviluppatori a scrivere routine efficienti di processazione di immagini la Intel ha rilasciato la *Image Processing Library* (IPL). Questa libreria è formata da un insieme di funzioni C che implementano operazioni di base per le immagini, come conversioni di colori e convoluzioni. Le IPL sono, inoltre, il nucleo delle librerie OpenCV che implementano sofisticate routine per l'image processing come il filtering, il face tracking, etc... La classe di base atta a gestire l'image processing in YARP è compatibile con la libreria IPL, ciò permette ad ogni utente di sfruttare a pieno i vantaggi dell'uso di queste librerie e delle OpenCV.

La piattaforma iCub

Il robot umanoide iCub [37, 38] (Figura 3.3) ha le dimensioni di un bambino di tre anni di età ed è stato progettato per essere capace di compiere i normali gesti di un bambino in fase di sviluppo: può camminare a quattro zampe e sedersi, le sue mani permettono la manipolazione, e ha testa ed occhi entrambi articolati. Possiede capacità visive, vestibolari e aptiche. Essendo un sistema aperto, il progetto e la documentazione di tutto l'hardware e software possiede licenza Free Software Foundation GNU, così che l'intero sistema possa essere riprodotto, utilizzato e modificato [3]. Concordemente a questo principio è stato distribuito un software di simulazione che gestisce il robot in un ambiente simulato che riproduce le leggi fisiche ed è dunque possibile testare il modello presentato come se fosse implementato sul robot reale. In questo sotto-paragrafo viene descritta la piattaforma robotica.

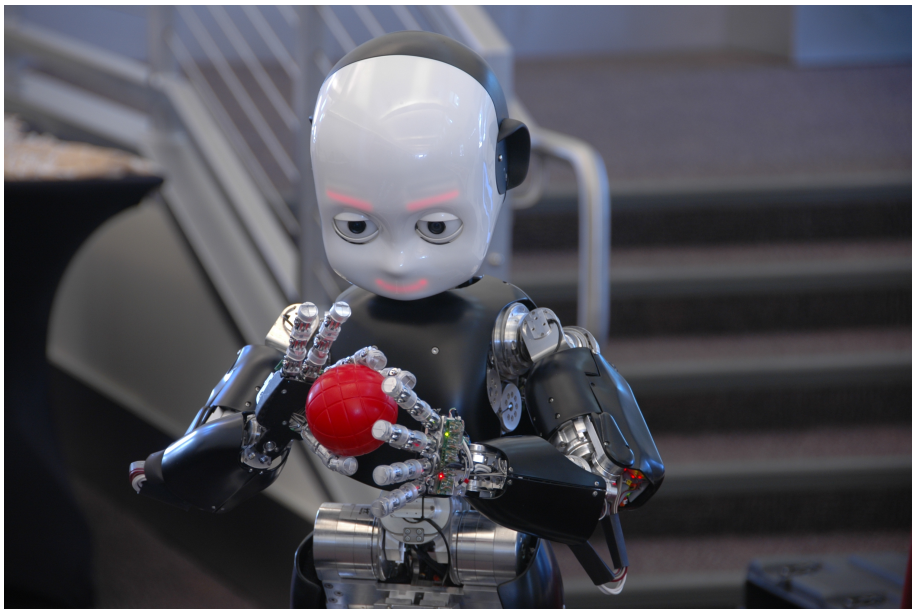

Figura 3.3: La piattaforma robotica iCub

Questo robot possiede testa, torso, due braccia con mani e due gambe usate principalmente per camminare a quattro zampe. L'iCub è alto circa 90 cm, pesa 23 Kg ed ha in totale 53 gradi di libertà distribuiti come segue: 7 per ogni braccio, 8 per ogni mano, 6 per la testa, 3 per il dorso e 7 per ogni gamba. Il sottosistema occhi-testa include primitive di processamento visivo di base, controllo di basso livello, sensori visivi, inerziali e propriocettivi. La maggior parte dei sistemi umanoidi esistenti hanno una struttura della testa semplificata con un ridotto numero di 6 gradi di libertà.

In modo da garantire un facile assemblaggio e semplici procedure di manutenzione, l'architettura meccanica è completamente modulare, in questo modo è possibile rimuovere e sostituire un modulo senza dover smontare l'intera struttura. Per la visione sono state utilizzate due telecamere di tipo DragonFly con risoluzione VGA e 30 fps di velocità. Queste telecamere sono particolarmente facili da installare in quanto il sensore CCD è montato su una testina remota connessa con il resto dell'elettronica attraverso un cavo flessibile. In questo modo il sensore è montato nel globo oculare, mentre l'elettronica è fissata un una scheda non mobile nella parte posteriore della testa. La testa contiene sensori inerziali MTi che possono fornire la posizione del corpo. Sono stati inoltre in-

stallati diversi microfoni intorno alla testa per poter localizzare la provenienza del suono in tutta l'area circostante. Tutte le schede di controllo sono integrate all'interno della testa e sono connesse ad un computer remoto attraverso un bus CAN. Per misurare la posizione dei giunti della testa, i motori sono forniti di encoders magnetici.

Simulatore

Per la fase di test è stato utilizzato un simulatore del robot iCub sviluppato, inizialmente per gli utenti di un passato progetto europeo (RoboCub [2]), come alternativa alla piattaforma fisica. Come base utilizza il motore grafico open-source Open Dynamics Engine (ODE). Questo motore include una interfaccia alle OpenGL che facilita la resa grafica degli oggetti nell'ambiente (sfera, scatola, etc...).

Il simulatore ha il ruolo di calcolare le leggi fisiche e la dinamica dei motori, permettendo al robot simulato di considerare il ruolo dei vincoli fisici all'interno dell'ambiente riprodotto. Inoltre, può calcolare e risolvere forze che emergono attraverso l'interazione di oggetti. In questo modo rende possibile la valutazione delle forze relative ai vincoli nei giunti. Il modello simulato è una replica dell'attuale piattaforma iCub che utilizza dati e matrici di inerzia. La piattaforma e i dati stessi possono essere aggiornati e modificati per una migliore resa.

L'obiettivo centrale di questo software è rendere la simulazione più vicina possibile al robot reale usando lo stesso tipo di interfaccia. Infatti attraverso la struttura software YARP è possibile controllare i dispositivi del robot simulato, le telecamere e gli attuatori di testa e braccia. I giunti del robot all'interno del simulatore hanno le stesse caratteristiche delle controparti reali, in questo modo è possibile fare uso degli stessi comandi e degli stessi parametri. Per questo motivo gli sviluppatori possono sperimentare lo stesso ambiente software con l'addizionale sicurezza dell'esecuzione in un mondo simulato, garantendosi la possibilità del totale riuso del codice.

Tutti i sensori della controparte reale dell'iCub sono disponibili in quella simulata, compresi i dati provvenienti dalle camere. In questo modo lo sviluppatore può ricevere ed analizzare i dati da entrambe le camere degli occhi, il sensore inerziale, gli encoder, etc...

L'ambiente offerto dal simulatore consiste nella controparte simulata dell'iCub inserita all'interno di uno spazio vuoto con delle texture per il pavimento, sfondo e cielo. È possibile aggiungere diversi tipi di oggetti all'interno del mon-

do simulato, a partire da forme base quali cubi, sfere e cilindri, fino ad arrivare al poter importare mesh realizzate dall'utente, video e quant'altro.

Per poter interagire con il robot il simulatore utilizza YARP. Una volta avviato, il software di simulazione apre una serie di porte di input ed output ciascuna collegata ad un sensore o ad un scheda di controllo del robot. All'utente che ha intenzione di interfacciarsi con una di queste componenti basterà aprire una porta locale e collegarla con quella corrispondente al sensore/controllore con cui intende interagire. Se, ad esempio, si volessero ottenere le immagini provenienti dall'occio sinistro del iCub simulato, si potrebbe utilizzare il seguente codice

```
BufferedPort<ImageOf<PixelRgb> > imagePort;
vector<ImageOf<PixelRgb> > imagesBuff;

imagePort.open ("/image/in");
Network::connect ("/icubSim/cam/left", "/image/in");
while (...)
{
        imagesBuff.push_back (*(imagePort.read ()));
}
```

La prima istruzione (non considerando le dichiarazioni presenti nelle prime due righe) serve a creare ed aprire una porta di input che verrà utilizzata come per la ricezione delle immagini. La seconda connette la porta relativa alla camera sinistra del robot con quella precedentemente creata. Per finire, l'istruzione all'interno del ciclo while legge l'immagine attuale catturata dalla camera e la registra all'interno di un vettore di immagini.

Per quanto riguarda lo studio del comportamento del modello analizzato in questa tesi le performance e le funzionalità del simulatore sono più che sufficienti. Infatti, per effettuare i test riguardanti le prestazioni del sistema di predizione, basta un ambiente modificabile, il controllo in velocità ed in posizione dei singoli giunti del robot, la risposta fornita dagli encoder in ogni istante e le immagini provenienti dalla camere ad una frequenza nota. Il simulatore dell'iCub risponde perfettamente a ciascuna di queste esigenze.

3.3 Implementazione

Ora che sono stati descritti sia i modelli e gli algoritmi scelti che gli strumenti utilizzati per implementarli si mostrerà come tale implementazione è stata

realizzata. Prima di ciò, però, è opportuno chiarire un importante aspetto implementativo. Si noti che il flusso ottico è una matrice di vettori bidimensionali rappresentanti le velocità sul piano dell'immagine. D'altro canto l'ESN accetta come ingresso un vettore di reali. Per questo motivo si è deciso di dividere il flusso ottico in due matrici di reali, una per gli spostamenti sull'asse delle x e l'altra per quelli sull'asse delle y. Con il flusso organizzato in questa maniera, si sono implementate 2 ESN con gli stessi settaggi sui parametri, una che predice il flusso sulle ascisse e l'altra sulle ordinate.

Si inizierà con il presentare l'implementazione dell'algoritmo di LK per il calcolo del flusso ottico, passando poi a quella delle reti neurali ricorrenti. Si forniranno, inoltre, i dettagli su come queste due implementazioni interagiscano tra di loro. Per finire saranno descritti alcuni accorgimenti e piccole implementazioni necessari per la creazione dei training set usati per testare il modello.

3.3.1 Calcolo del flusso ottico

Per implementare il blocco rappresentante il generatore di flusso ottico del modello si è deciso di utilizzare le OpenCV. Per il calcolo del flusso si è usata la funzione di liberia *cvCalcPticalFlowPyrLK ()* descritta in precedenza, partendo da un insieme di feature ottenute dall'utilizzo delle funzioni *cvGoodFeaturesToTrack ()* e *cvFindCornerSubPix ()*. Trovandosi a lavorare in un ambiente simulato, con immagini con basso contenuto di rumore ed effettuando la fase di learning offline a priori, non si è vista la necessità di analizzare in dettaglio il settaggio dei parametri delle funzioni adoperate. Per questo motivo, le prestazioni ottenute assegnando i valori più comunemente usati in letteratura sono risultate più che soddisfacenti. In questo modo si è potuto focalizzare maggiormente l'attenzione sullo studio dell'implementazione delle reti ricorrenti. Per essere più precisi si è calcolato il flusso ottico con la finestra di dimensione 10x10, l'uso di massimo 100 feature e 5 livelli della piramide.

Il programma per il calcolo del flusso è organizzato nel modo seguente:

1. Legge una alla volta le immagini da cui si vuole generare il flusso ottico.
2. Sceglie le migliori feature di cui intende tener traccia.
3. Calcola il vettore di velocità appartenenti alle feature a partire dall'immagine precedente e da quella attuale.
4. Filtra il risultato eliminando le velocità troppo alte e troppo basse per eliminare parte del rumore.

5. Memorizza in due matrici di float (una per gli spostamenti in x e l'altra per quelli in y), di dimensione pari a quella delle immagini, i vettori delle velocià appena ottenuti. Questo viene fatto partendo da due matrici di 0. Per ogni feature, in ciascuna matrice, vengono inserite lo spostamento in x (o y a seconda della matrice) nell'elemento rappresentante la posizione della feature nell'immagine di partenza.

3.3.2 Implementazione dell'ESN

Le due Echo State Network usate per calcolare il flusso ottico predetto sono state implementate con il toolbox di matlab descritto precedentemente in questo capitolo. Entrambe presentano 13 neuroni di input, 12 corrispondenti al flusso ottico semplificato (nel prossimo paragrafo viene spiegato più dettagliatamente come ottenerlo) e 1 al comando motorio. Per quanto riguarda l'output invece sono presenti solo i 12 neuroni rappresentanti il flusso predetto. Più specificatamente, una delle reti prende in ingresso sequenze in cui ogni elemento è formato da:

1. 12 reali corrispondenti alle componenti in x (y per l'altra rete) di ciascuno dei 12 vettori del flusso ottico semplificato.
2. 1 reale corrispondente al comando motorio in velocità (espresso in gradi) dato al giunto di yaw (pitch per l'altra rete).

Mentre in output:

1. 12 reali corrispondenti alle componenti in x (y per l'altra rete) di ciascuno dei 12 vettori del flusso ottico semplificato predetto.

Per addestrare e testare ciascuna delle reti è stato usato il seguente procedimento. Presa una sequenza di flusso ottico semplificato con i relativi comandi motori, si sono formate una sequenza di training ed una di test. Fatto ciò, si è inizializzata la rete con i parametri opportuni (scelta dei parametri e motivazioni a riguardo verranno discussi nel prossimo capitolo) utilizzando la funzione *generate_esn ()*. Ottenuta la rete la si è addestrata utilizzando la funzione *train_esn ()*. Come algoritmo di apprendimento è stato utilizzato il metodo batch descritto nel sotto-paragrafo 3.1.2, che riduce l'errore quadratico tra uscita della rete e uscita desiderata per mezzo della pseudoinversa. Per far ciò si e fatta "girare" la rete dandole in ingresso la sequenza di training, memorizzando le uscite della rete ad eccezion fatta dei primi elementi scartati in funzione dell'*Echo State Propriety*. Una volta addestrara si è testata attraverso la funzione *test_esn ()*, scartando anche in questo caso i valori iniziali.

3.3.3 Creazione del training set

Per quanto riguarda la fase di learning dell'Echo State Network, si è scelto di effettuare l'addestramento offline a priori con l'algoritmo batch per il calcolo dell'errore quadratico mostrato precedentemente. Per effettuare questo tipo di apprendimento offline, è necessario fornirsi di un trainig set formato dal flusso ottico (nella versione semplificata) istante per istante e dalla sequenza di comandi motori corrispondenti. Quindi, ogni elemento della sequenza di trainig avrà, come input i vettori del flusso ottico semplificato e i comandi motori inviati in un determinato istante, mentre come output il flusso ottico ottenuto dall'esecuzione di tali comandi. Per estrarre queste informazioni dal simulatore è necessario compiere alcuni passi:

1. Estrarre una sequenza di immagini dal simulatore con i comandi motori che le hanno generate. Queste due sequenze dovranno essere sincronizzate in modo tale da sapere quale comando motorio è servito per passare da un'immagine all'altra. Perché i dati analizzati siano il più possibile simili a quelli ottenuti da una camera reale, bisognerebbe estrarre le immagini ad una frequenza di 30 Hz.

2. Calcolare la sequenza di flussi ottici relativa alla sequenza di immagini precedentemente estratte e affiancarci i comandi motori corrispondenti.

3. Semplificare i flussi ottici ottenuti dividendo l'immagine in zone e calcolando per ogni zona il vettore media delle velocità delle feature rilevate in quella zona.

4. Da questa sequenza, formata da flussi semplificati e comandi motori, ricavarsi le sequenze di input ed output desiderate.

Vedremo ora le soluzioni implementative realizzate per ciascuno di questi punti.

Estrazione immagini e comandi motori

Per riuscire ad ottenere una sequenza di immagini e relativi comandi motori necessari per far effettuare un movimento prestabilito al robot, è stato scritto un processo in C++ utilizzando le librerie YARP per la comunicazione con il robot. Dato che, per il funzionamento del modello è necessaria una sola telecamera, vengono estratte solamente le immagini provenienti dall'occhio sinistro del robot. Per far compiere il movimento desiderato all'iCub, vengono utilizzati

solamente i comandi motori in velocità dei due giunti del collo corrispondenti allo yaw e al pitch (imbardata e beccheggio) della testa. Si è optato per questa scelta perché, come si vedrà meglio nel prossimo capitolo, questi tre dati sono sufficenti per effettuare i test necessari ad analizzare il funzionamento della rete.

La velocità del mondo nel simulatore è pari a 100 passi di simulazione al secondo. Per questo motivo, si è deciso di realizzare il processo di controllo in modo tale che compia le sue azioni ogni tre passi di simulazione (circa 30 Hz), accertandosi di non saltarne nessuno.

I movimenti da utilizzare per generare il training set vengono espressi tramite le sequenze di comandi motori in velocità necessari per effettuare tali movimenti. Ogni tre passi l'estrattore/controllore memorizza l'immagine attuale della camera sinistra in un vettore ed esegue l'elemento sucessivo della sequenza di comandi. Ultimato il movimento desiderato il processo memorizza su disco le immagini presenti nel vettore ed un file di testo contenente i comandi motori eseguiti per ottenerle (la coppia di comandi motori n°4 del file di testo rappresenta il controllo che è stato necessario per passare dalla quarta immagine alla quinta).

Calcolo del flusso ottico

Per il calcolo del flusso viene utilizzato l'estrattore di flusso ottico di cui è stata descritta l'implementazione nel sotto-paragrafo 3.3.1. Gli si fornisce in ingresso la sequenza di immagini ottenuta dal simulatore e si ottiene come uscita le 2 sequenze dei flussi ottici (una con le matrici delle ascisse ed una con quelle delle ordinate).

Semplificazione del flusso

Per ottenere il flusso ottico semplificato vengono processate le 2 sequenze dei flussi ottici nel seguete modo. Per ogni elemento di ciascuna sequenza viene:

1. Suddivisa la matrice di reali in sotto-matrici, una per ogni zona in cui si intendere suddividere l'immagine.
2. Fatta la media dei punti diversi da zero di ciascuna sottomatrice.
3. Costruito un vettore contenente l'insieme delle medie appena ottenute.

In questo modo si ottengono le 2 sequenze contenenti i flussi ottici semplificati.

Creazione delle sequenze di input e output

Per finire dalle 2 sequenze di fllussi ottici semplificati e da quella di comandi motori vengono creati i due training set (uno per la rete che deve predire le velocità in x e uno per quella che le deve predire in y). Per ciascuna rete, l'input dell'i-esimo elemento della sequenza di training conterrà:

1. Il flusso calcolato dalle immagini (i -1)-esima ed i-esima.
2. Il comando motorio (yaw per le x e pitch per le y) necessario per passare dall'immagine i-esima alla (i + 1)-esima.

L'output dell'i-esimo elemento conterrà solamente il flusso calcolato dalle immagini i-esima ed (i + 1)-esima.

Nel presente capitolo sono stati mostrati la struttura, gli algoritmi e gli stumenti scelti per l'implementazione del modello di predizione. Sucessivamente, nel capitolo 4, verrano riportati i risultati dei test effetuati che dimostrano la validità della tesi proposta.

Capitolo 4

Test sulla rete ed analisi dei risultati

Fino ad ora è stata descritta un'implementazione del modello interno predittivo necessario per la realizzazione di un sistema di controllo predittivo. Per dimostrare, però, la validità ed il corretto funzionamento di tale implementazione, è necessario effettuare dei test. Nei paragrafi che seguono saranno descritti, proprio, i test scelti per questo scopo. Come si è detto in precedenza, le varie prove sono state effettuate all'interno di un ambiente simulato. In questo ambiente è presente una controparte digitale del robot umanoide iCub, che risulta essere una piattaforma valida per le esigenze dei test svolti.

Il presente capitolo contiene un primo paragrafo in cui si descriveranno in dettaglio le specifiche dell'ambiente di testing e la tipologia di prove effettuate. Questo sarà utile per poter passare al paragrafo sucessivo in cui verranno forniti i risultati dei test ed una loro attenta analisi. Il tutto sarà concluso da un breve riassunto dei risultati ottenuti.

4.1 Descrizione dei test svolti

Per ottenere delle prove sufficienti a validare il corretto funzionamento del modello in un ambiente reale/casuale, è stato deciso di utilizzare alcune configurazioni delle simulazioni scelte ad hoc. Per fare questo, si è optato per una particolare organizzazione dell'ambiente simulato, utilizzando come test quattro diverse sequenze di comandi motori. Ognuna di queste prove stata ripetuta

sia muovendo il giunto di yaw della testa (simulazione di spostamenti laterali), che muovendo quello di pitch del torso (simulazione di spostamenti in avanti). In quest'ultimo, per mantenere l'asse z della testa sempre parallelo al terreno, è stato mosso contemporaneamente il pich della testa con comando motorio invertito. I test sono stati effettuati sulla rete atta a predire le ascisse del flusso ottico semplificato, essendo l'andamento sulle ordinate molto simile. Per addestrare e testare la rete, sono stati utilizzati le sequenze di spostamenti sulle ascisse assieme ai comandi motori di yaw e pitch. Per tutti i test è stata utilizzata come training set una sequenza ottenuta a partire da un segnale random di comandi motori. Per finire si è scelto di effettuare un test con comandi motori casuali su entrambi i giunti contemporaneamente. In tutti test vengono usati comandi motori in velocità.

4.1.1 Scelta dell'ambiente simulato

Saranno descritte, adesso, la configurazione dell'ambiente simulato utilizzata per eseguire tutti i test. Si è deciso di utilizzare una struttura più semplice possibile, in modo da ridurre il rumore dato dall'algoritmo di flusso ottico. La configurazione presenta una matrice di sfere colorate di fronte al robot (Figura 4.1) con le seguenti caratteristiche:

1. Le sfere si trovano a distanza variabile e casuale dal robot.
2. Se presa su di un piano perpendicolare all'asse delle z del robot, la distanza dei centri delle sfere risulta pari a due volte il loro diametro.

È stato scelto questo test perchè si è reputato fosse una buona aprossimazione delle situazioni che potrebbero presentarsi in un ambiente reale.

Fornendo una sequenza di immagini in cui il flusso ottico risulta uniformemente distribuito, si ottiene una semplificazione delle situazioni in cui l'ambiente è ricco di features. Questa configurazione permette, quindi, di constatare se la rete riesca ad apprendere la relazione tra uscita e comando motorio.

In secondo luogo si riesce a simulare la situazione di ambienti con feature posizionate a distanze differenti dal robot. In questo modo si può capire se la rete riesca ad apprendere la relazione che intercorre tra ampiezza del vettore di flusso ottico e distanza della feature dal robot.

Sono stati effettuati quattro differenti test basati su diverse sequenze motorie, grazie alle quali è stato comandato il robot simulato ed estratto il flusso

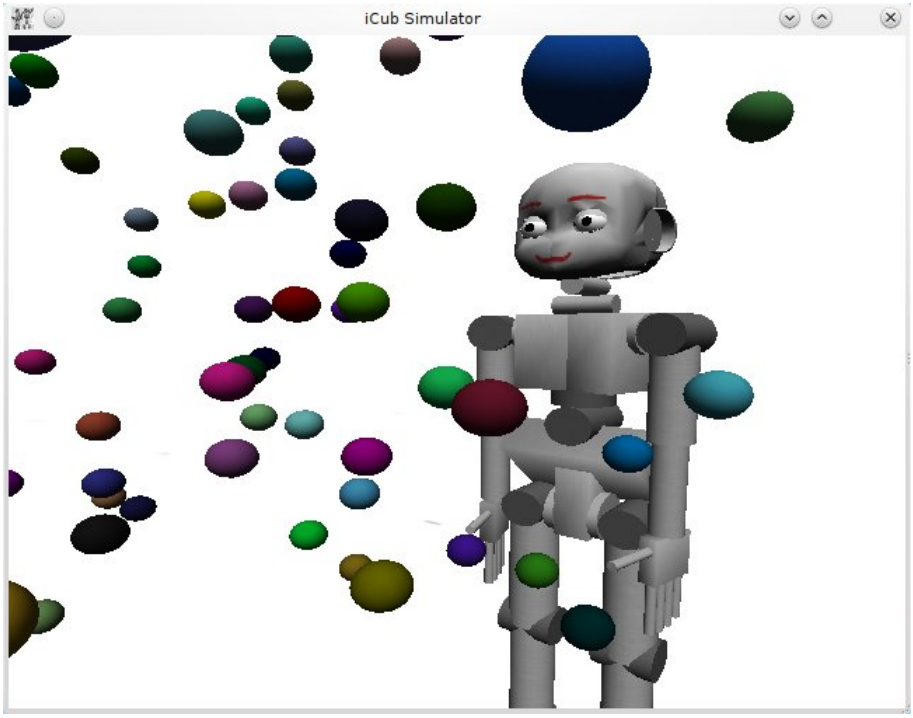

Figura 4.1: Ambiente di simulazione utilizzato

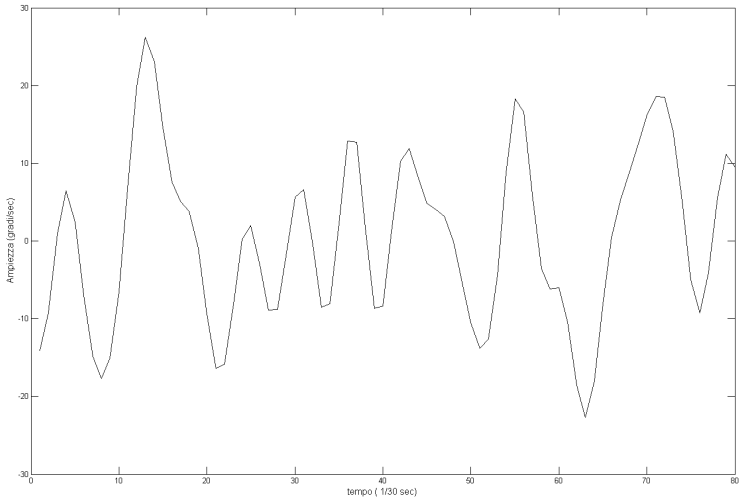

Figura 4.2: Porzione del segnale random usato per il training

ottico. Per tutti i test è stato ottenuto il training set a partire da un segnale random di comandi motori. Questo segnale corrisponde ad un rumore colorato con banda passante compresa tra 0.1 e 0.8 (finestra con guadagno decrescente ad aumentare della frequenza) ed ampiezza di 40 gradi (Figura 4.2).

4.1.2 Scelta dei test

Nel seguente sottoparagrafo si fornirà una descrizione delle quattro tipologie di test utilizzate.

Per ogni test sono state campionate 100 immagini, collegate ai rispettivi comandi motori, da cui è stato estratto il flusso ottico semplificato. Si sono, poi, ottenuti i vari test set affiancando il flusso ottico con i comandi motori corrispondenti.

- Gradino: Una sequenza di comandi motori che parte da 1 nei primi 50 elementi, per poi fare un salto a 15 gradi nei restanti 50.
- Rampa: La seconda sequenza di comandi motori è formata da una rampa crescente (da 1 a 15) che inizia al 25° elemento e termina al 75°. La rampa è preceduta da una sequnza di 25 comandi pari a 1 grado e seguita da una di 25 comandi pari a 15 gradi.
- Sinusoide sempilce: La sequenza di comandi motori più semplice su cui si sono testate le reti è una sinusoide con ampiezza di 15 gradi e frequenza 0.2.
- Movimenti casuali: L'ultima sequenza di comandi motori è stata creata a partire da un rumore colorato dello stesso genere di quello usato per il training. Sono stati campionati 100 elementi.

Ciascuno di questi test rappresenta un particolare comportamento che si desidera che la rete impari.

Il gradino corrisponde ad uno spostamento della testa a velocità costante iniziato in modo brusco. Nel caso degli esperimenti sul giunto di pitch del torso questo rappresenta uno spostamento a velocità costante in avanti iniziato improvvisamente.

La rampa rappresenta, invece, degli spostamenti della testa senza cambi di direzione, a velocità crescente ed accelerazione costante. Ad esempio, l'inizio di una camminata in avanti o lo spostamento della testa di lato a partire da fermo.

La sinusoide genera degli spostamenti della testa con cambi di direzione e con velocità ed accelerazione variabili (sebbene ciclici). Questo tipo di segnale può essere una semplificazione di cambi di direzione tra camminata in avanti e all'indietro.

Il moto random rappresenta una generalizzazione dei diversi movimenti possibili.

4.2 Analisi dei risultati ottenuti

Saranno mostrati, adesso, i risultati ottenuti nei vari test. In ognuno di questi è stata usata la seguente configurazione di rete:

- 14 Nodi di input (12 per il flusso ottico e 2 per i comandi motori sullo yaw della testa e pitch del busto)
- 30 Nodi interni
- 12 Nodi di output (flusso ottico semplificato)
- Raggio spettrale 0.8

Per quanto riguarda il numero di neuroni interni, è stato analizzato l'errore medio delle uscite all'aumentare della dimensione della rete (Figura 4.3) e si è notato che quest'ultimo migliora fino a circa 30 nodi. Per questo motivosi è deciso di scegliere quest'ultima come dimensione del reservoir.

Il raggio spettrale , invece, non sembra influire in modo significativo sulle prestazioni della rete e quindi si è scelto il valore che ha fornito, mediamente, migilori risultati.

Essendo stati investigati solamente i risultati relativi agli spostamenti sull'asse delle x, per tutti i test presentati in questo capitolo saranno mostrati e analizzati uscite ed errore delle quattro zono centrali. Si è presa questa decisione per facilitare analisi e presentazione dei risultati. Questo è stato possibile per il fatto che le tre fasce dell'immagine (quattro zone superiori, quattro centrali e quattro inferiori) hanno un comportamento del tutto simile, dato che si analizza il comportamento sulle ascisse.

La tabella 4.1 mostra gli errori sulle quattro uscite della rete per ogni caso studiato. Gli errori presentati in questa tabella sono una media normalizzata dell'errore quadratico medio per ogni neurone di uscita. I valori di errore ottenuti sono circa un decimo rispetto all'errore in pixel. Questo vuol dire che

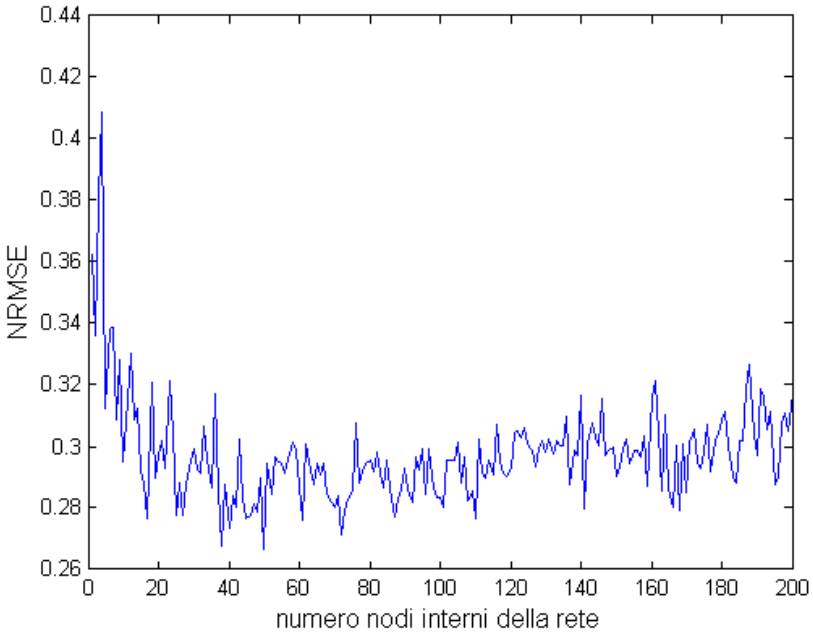

Figura 4.3: Errore quadratico medio normalizzato delle uscite della rete all'aumentare della dimensione del Reservoir. La rete è stata addestrata su di una sequenza random analizzata in seguito.

TRAIN/TEST NRMSE				
SEQUENZA	Uscita 1	Uscita 2	Uscita 3	Uscita 4
Spostamenti Yaw della Testa				
Training	0.25015	0.2502	0.25207	0.25187
Gradino	0.49438	0.40566	0.40555	0.39227
Rampa	0.58501	0.52713	0.54531	0.5157
Sinusoide	0.22703	0.2345	0.23656	0.2329
Random	0.26533	0.27163	0.26786	0.26874
Spostamenti Pitch del Busto				
Training	0.37441	0.50828	0.56694	0.37821
Sinusoide	0.40038	0.50279	0.59165	0.2999
Random	0.50953	0.39472	0.32539	0.32369
Spostamento su Entrambi i Giunti				
Training	0.20191	0.19961	0.19579	0.19268
Test	0.17435	0.18156	0.18287	0.17843

Tabella 4.1: Tabella che mostra l'errore quadratico medio normalizzato delle quattro zone analizzate. Nel nostro caso l'NRMSE equivale a circa un decimo dell'errore in pixel. Alcuni valori grandi sono dati dal fatto che il segnale del flusso ottico presenta diverso rumore, mentre la rete estrapola l'andamento generale del flusso.

l'errore durante i vari test oscilla tra i due e i cinque pixel. Questi risultati potrebbero sembrare eccessivi, contando che il segnale random va dai 40 ai -40 pixel. Se però si va ad analizzare più dettagliatamente il segnale di flusso ottico estratto, si può notare come il rumore stesso sia dell'ordine di 2-4 pixel. Questo sposta la causa dei valori lievemente alti di errore alla non ottimale cattura del flusso ottico. Questo lavoro si concentra maggiormente sull'implementazione della rete piuttosto che sulla ricerca di un algoritmo di flusso ottico ottimo. Sebbene questi valori di errore, sarà fatto notare come la rete si comporti nel modo richiesto.

Nei due sottoparagrafi che seguono verranno esposti i risultati ottenuti sia con le quattro sequenze di comandi motori per lo yaw della testa che quelli ottenuti con le sequenze di comandi per il pitch del busto.

4.2.1 Comandi per lo yaw della testa

Questi comandi servono a fornire un primo semplice task da risolvere per la rete. In questo caso l'andamento del flusso ottico risulta essere particolarmente collegato al comando motorio fornito in ingresso. Ciascuno di questi test è stato

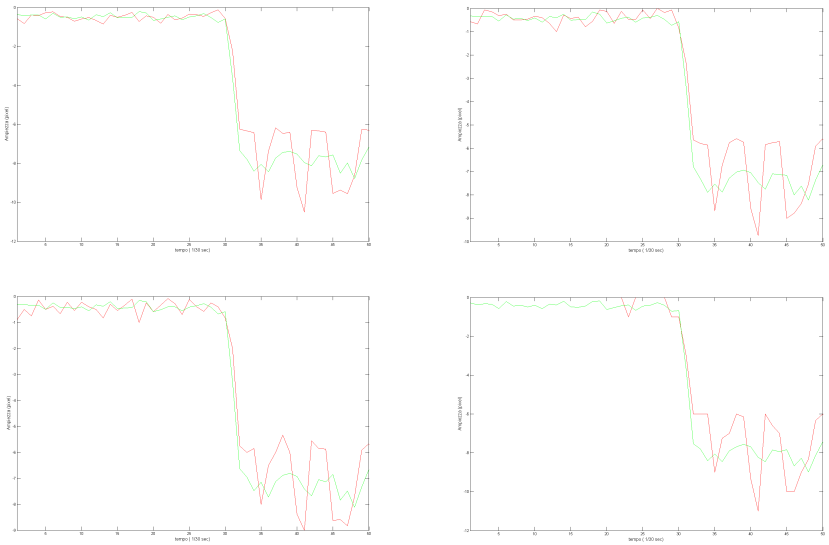

Figura 4.4: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sul gradino. (Rosso: uscita predetta, Verde: uscita desiderata)

effettuato su una rete addestrata con 1900 elemementi apparteneti alla sequenza di movimenti random descritta in prececdenza. Per evitare che l'uscita della rete dipendesse troppo dalla configurazione di pesi iniziale, si è deciso di scartare i primi cento passi. Una volta eseguito il training attraverso il metodo della pseudoinversa, sono stati eseguiti i vari test. Nelle seguenti pagine saranno mostrati e analizzati i grafici relativi ad una porzione delle varie sequenze di test per i quattro neuroni di output.

Gradino: Il primo test effettuato è la sequenza di comandi motori corrispondente ad un gradino. In questo caso, come in quello sucessivo della rampa, l'errore è circa il doppio di quello ottenuto con sinusoide e sequenza random. Questo fatto è da imputare a due ragioni: il segnale si discosta notevolmente da quello presente nel training ed il segnale di flusso ottico utilizzato presenta una grossa quantità di rumore. Nonostante questo errore, comunque, se si osserva la Figura 4.4, si nota come la rete, grazie al comando motorio, riesca a predire l'andamento generale del flusso evitando di seguire il grosso rumore presente nel segnale. Questa, come sarà analizzato più dettagliatamente in seguito, è una caratteristica piuttosto

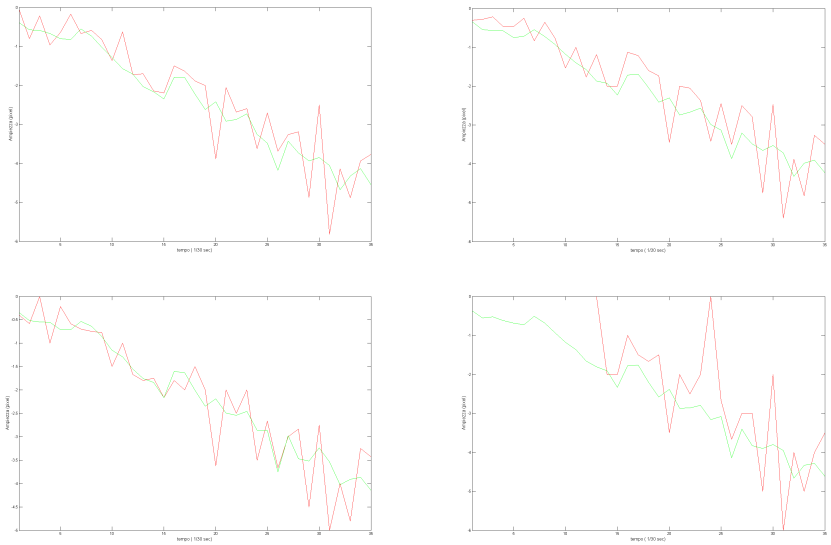

Figura 4.5: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sulla rampa. (Rosso: uscita predetta, Verde: uscita desiderata)

utile per i nostri scopi.

Rampa: La rampa mostra le stesse caratteristiche rilevate nel test del gradino. Anche qui il flusso ottico ha un comportamento notevolmente affetto da rumore, e anche qui la rete riesce comunque ad estrapolare l'andamento generale del flusso. (Figura 4.5)

Sinusoide: La sinusoide risulta il test più semplice da compiere essendo comunque ad una frequenza compresa all'interno del range di quelle del segnale random. Inoltre in questo caso il segnale di flusso ottico risulta notevolmente più pulito. Per queste ragioni l'errore raggiunge valori di circa 2 pixel. (Figura 4.6)

Random: Per effettuare questo test sono stati presi 100 campioni di un segnale random dello stesso tipo rispetto a quello usato per il training. A differenza del training, però, la disposizione casuale delle sfere era differente. Il vantaggio nell'utilizzare questo tipo di segnale è quello di simulare un'approssimazione dei movimenti ottenuti attraverso una camminata. Anche

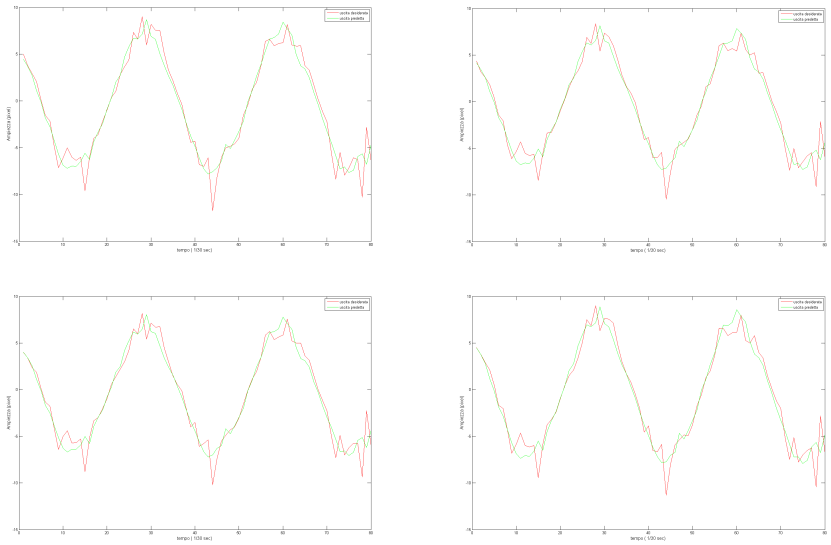

Figura 4.6: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sulla sinusoide. (Rosso: uscita predetta, Verde: uscita desiderata)

in questo caso la rete fornisce i risultati sperati, ottenendo un errore medio di circa 2 pixel. (Figura 4.7)

Considerazioni

Da questi risultati è possibile constatare alcune importanti caratteristiche della rete. Prima fra tutte, la capacità di apprendere varie configurazioni mai viste prima del segnale di test (basate su movimenti plausibili) a partire da un segnale random di training. Secondo, il fatto chei suoi risultati non sono particolarmente influenzati dal rumore, anzi estraggono l'andamento generale del flusso ottico. Questa caratteristica è molto importante se si considera il seguente fatto. All'interno dell'architettura proposta il flusso ottico predetto viene utilizzato per il calcolo dell'immagine sintetica. Se la rete predicesse anche l'errore intrinseco nell'algoritmo di flusso ottico, quest'ultimo verrebbe riportato anche nell'immagine sintetica, portando ad un errore nel confronto. Questa caratteristica sarebbe importante se il confronto avvenisse a livello di flusso ottico e non a livello di immagini.

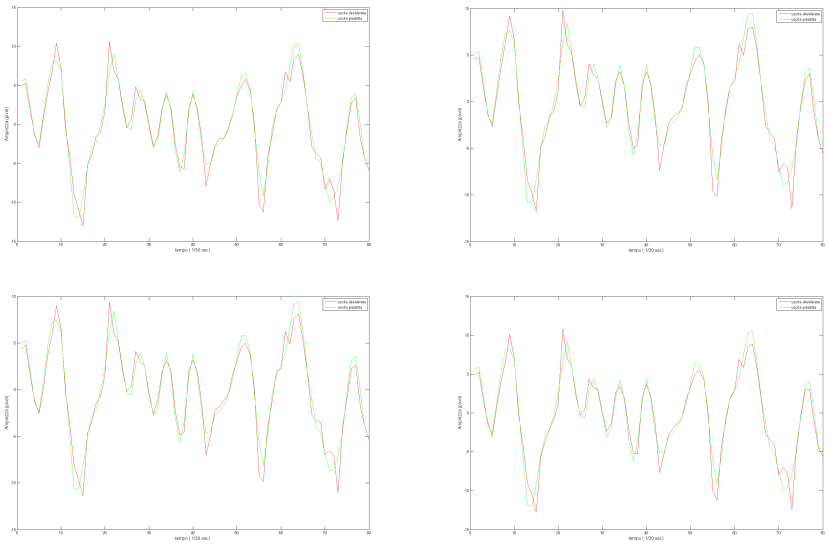

Figura 4.7: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sul segnale random.

Grazie a questi test è stato possibile dimostrare il funzionamento della rete per movimenti laterali. Nel prossimo paragrafo si analizzerà il problema dei movimenti avanti ed indietro, molto frequenti nella camminata.

4.2.2 Comandi per il pitch del busto

Per quanto riguarda un'analisi degli spostamenti sulle ascisse del flusso ottico, una delle dinamiche più interessanti fornite dai movimenti in avanti e all'indietro è il fatto che le zone nella parte destra e quelle nella parte sinistra si comportano in maniera opposta. Infatti, se si analizza un flusso derivato da uno spostamento in avanti, le zone sinistre conterranno dei vettori direzionati verso sinistra, mentre quelle destre direzionati verso destra. L'ampiezza di questi vettori tenderà, inoltre a diminuire via via che ci si avvicina verso il centro dell'immagine fino ad annullarsi. Per capire se la rete riesca ad apprendere questo andamento, sono stati effettuati nuovamente i test sulla sinusoide e sulla sequenza random di comandi motori sul giunto di pitch del busto.

In questo caso gli errori si assestano ad un valore di circa il doppio rispetto ai test precedenti. Il motivo è sempre principalmente da attribuirsi al rumore del flusso ottico. Analizzando i grafici della sinusoide si può facilmente notare il

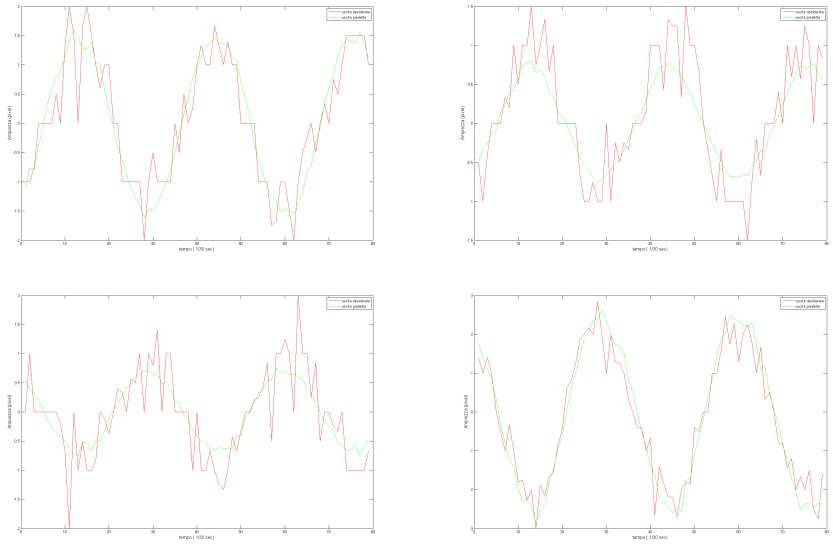

Figura 4.8: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sulla sinusoide nel caso di spostamento del giunto del busto. (Rosso: uscita predetta, Verde: uscita desiderata)

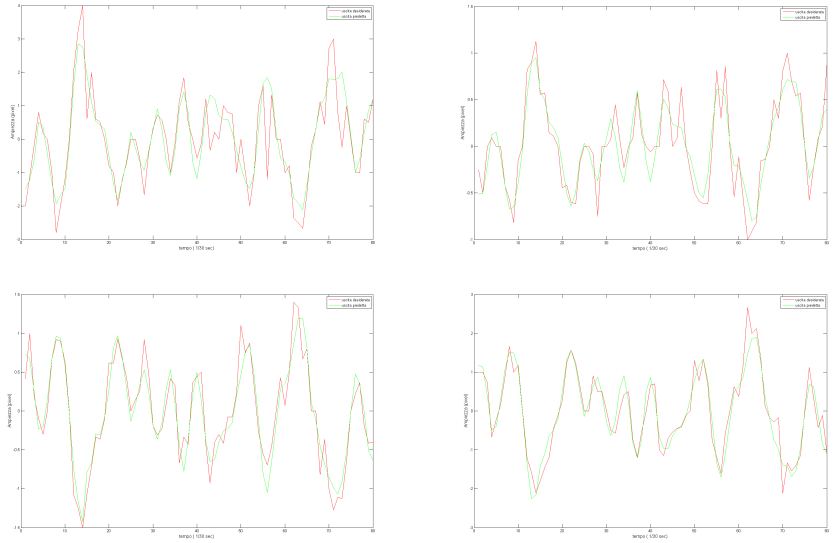

Figura 4.9: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sul segnale random nel caso di spostamento del giunto del busto. (Rosso: uscita predetta, Verde: uscita desiderata)

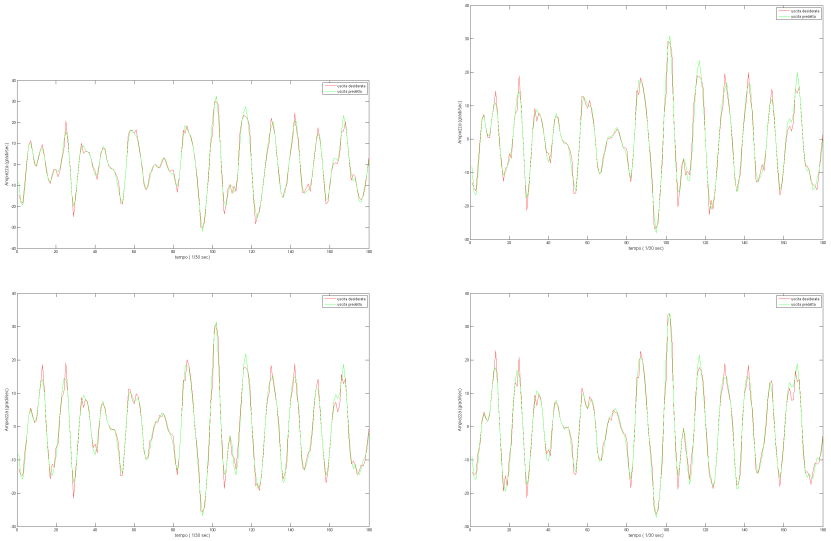

Figura 4.10: Porzione dei valori di output dei quattro neuroni di uscita presi in considerazione durante il test sul segnale random nel caso di spostamento su entrambi i giunti. (Rosso: uscita predetta, Verde: uscita desiderata)

comportamento descritto in precedenza. La sinusoide risulta sfasata nelle uscite di sinistra rispetto a quelle di destra. Inoltre l'ampiezza del flusso nei settori centrali è inferiore rispetto a quella dei settori esterni. È possibile notare come questa dinamica sia perfettamente appresa dalla rete.

4.2.3 Test finale su entrambi i giunti

L'ultimo test che si è effettuato riguarda uno spostamento simultaneo di entrambi i giunti. Il training set è stato formato nello stesso modo in cui sono stati effettuati i precedenti, con la differenza che è stata data una diversa sequenza random di comandi motori ad entrambi i giunti (precedentemente veniva mosso solo un giunto a seconda del training). Per il test è stata utilizzata una sequenza random simile, dalla quale è stato ottenuto un test set di 200 elementi.

Anche in questo caso l'errore rimane sotto la soglia di 2 pixel, fornendo il risultato sperato. Questo permette di capire come la rete riesca ad apprendere anche in situazioni in cui le dinamiche di due movimenti random si intersecano fra di loro, dimostrandosi un ottimo strumento per assolvere i compiti dell'architettura.

Conclusioni

Si è arrivati finalmente al punto in cui bisogna tirare le somme. In primis verrà brevemente ricapitolato il lavoro svolto.

Dopo un'attenta analisi dello stato dell'arte è stata proposta un'architettura per il controllo della locomozione di un robot attraverso la visone. Alla base di questo sistema di controlo è presente un modello interno basato sulla predizione dell'input sensoriale (le immagini della camera). Si è deciso di realizzare il modello interno di tale sistema attraverso la predizione del flusso ottico, sfruttandolo per la generazione di un immagine sintetica rappresentante l'input visivo futuro. Una volta decisa l'architettura la si è implementata attraverso l'utilizzo di una ESN. Dopo una serie di accurati test, è stata dimostrata l'adeguatezza della rete per gli scopi prefissi.

I test effettuati permettono di valutare l'efficacia della rete scelta in presenza di differenti tipi di movimento. In particolare sono stati effettuati test con una tipologia di movimento random il cui contenuto frequenziale è simile a quello presente durante una camminata umana. Sebbene in presenza di dati di input molto rumorosi, a causa del simulatore della piattaforma iCub che presenta diverse inefficienze computazionali, l'errore di predizione risulta sufficientemente piccolo per lo scopo di questo lavoro. Quindi è stato dimostrato che è possibile predire l'andamento del flusso ottico conseguente al comando motorio desiderato. La predizione avviene a partire dalla conoscenza della storia della sequenza del flusso ottico passato e dal comando motorio stesso. È dunque possibile ipotizzare un uso iterativo della rete al fine di ottenere una predizione di flusso ottico a lungo termine. Il comportamento predittivo delle conseguenze sensoriali delle azioni è alla base dell'architettura di controllo dell'EP.

Quest'architettura rappresenta un approccio biologicamente plausibile ed in linea con gli ultimi sviluppi in questo campo. Per validare al meglio il sistema, sarà necessaria un'implementazione su piattaforma robotica, utilizzando dati di

input reali ed un'implementazione ottimale dell'algoritmo di estrazione del flusso ottico. In questo caso bisognerà porre particolare attenzione ai limiti risultanti dalla divisione in zone, causa principale del forte rumore presente nel flusso ottico. Una possibilità consiste nell'aumentare il numero delle zone considerate e quindi l'input alla rete, oppure di valutare un altro tipo di semplificazione.

Ringraziamenti

Prima di tutto GRAZIE Davide ed Egidio.

Finalmente posso parlare in prima persona e non impersonalmente! Iniziamo con i ringraziamenti allora.

Ringrazio la professoressa Laschi per la pazienza avuta e l'aiuto fornito durante lo sviluppo della tesi.

Un immenso grazie a tutto il RobotAn e ospiti. Giovanni per avermi impedito di venire alla lurea in infradito, Laura per avermi fatto notare che il numero di c*****e che dico in laboratorio e esponenzialmente maggiore delle cose serie, Elena che mi ha fatto rigare dritto, Emma per il momento "la sai l'ultima" giornaliero, Giacomo, Carmine, Fabio e Bano.

Grazie al professor Micheli per avermi introdotto all'apprendimento automatico durante questa specialistica e per essersi mostrato sempre disponibile.

Grazie Sorellona, anche se sei lontana (Londra, Barcellona, Malesia, Egitto, Praga, Parigi, Cape Town, etc etc etc etc) so che posso sempre contare su di te. Grazie a Mamma e Papà che in questi 26 anni mi hanno dato il loro amore ed affetto, sostenendomi in tutto quello che faccio.

Grazie Moni che mi hai sopportato nel bene e nel male in questi anni Pisani.

Mauri, anche se ci sentiamo poco, quando esco di testa trovi sempre il modo di riportarmi sulla retta via. Sei il uno degli amici migliori che si possano avere ayooooooooooo.

Siamo arrivati al gruppo di amici più ganzo che una persona possa avere, i Fumbles and coo! Dadde, Ste, Peppio, Pica, Cinzia, Robi, Marti, Piba, Enri per ognuno di voi non basterebbero tutte le pagine di questa tesi. Shingoo kiiick.

Grazie alla marmaglia di via Battelli 21. Andre mi sto laureando... è fighisssssiiiimoooo. Questi giorni sono stati terribbbili. Nico grazie per la tua sopportazione quotidiana (come riesci a vivere con me il brocco e metallo proprio non

lo capisco) e Brocco... beh Brocco è Brocco non c'è altro da aggiungere. E io aggiungo anche Christian e Anto, nel nostro cuore siete sempre nostri coinquilini!

Non posso non ringriaziare la mia controparte Mimmo, grazie alla nostra fusion è venuto alla luce Nimmo (famoso essere mitologico). Grazie anche a tutti i Palmarini Pendenti, in particolare Nicco e Chris con cui ho passato quest'estate di tesi/divertimento (forse un po' troppo poco del primo e troppo del secondo), Galo che rende possibile l'esistenza di questo bellissimo gruppo di Capoeira qui a Pisa e Paahppi per tutto quello che continua ad insegnarmi.

Grazie a Mathe (ma ci va l'acca oppure non centra niente?), soprattuto per il periodo in cui si studiava assieme in biblioteca!

Olluuuuuuu. Minca ti geliiii. Neeeeh neeeeeh neeeh. Indovinate un po' chi sto per ringraziare? Ovviamente tutto il gruppo di sardi e acquisiti qui a Pisa Niho il gaggio di Cagliari, Dani e Gino i Bidduncoli di Orroli, Greg il re dei Gaggi nonchè sindaco di Nurri, Enzo "oh Eeenzo" e Giulia, PaoloPicciPaoloPicciPaoloPicci, Vespa, Cois, Bat Roberto, Marta, Bossini e Leonardo.

Grazie a Cristiano e Lucio con i quali non mi faccio mai sentire, ma che sanno che gli voglio bene!!

Un super ringraziamento anche a tutto il gruppo degli agrari che mi ha permesso di svagarmi con verie partite di calcetto.

Grazie a Ron Gilber per avermi fatto passare dei mesi felici.

Per finire un Grazie enorme a tutti quelli che non sono stati nominati, ma sanno di errere stati importanti per me.

Ah Davide... Ora l'ho finita sul serio la tesi!!!

Elenco delle figure

Bibliografia

[1] Opencv web page: http://surceforge.net/projects/opencvlibrary/.

[2] Robotcub web page: http://www.robotcub.org.

[3] wiki robocub: http://eris.liralab.it/wiki/main_page.

[4] Yarp web page: http://eris.liralab.it/yarp/.

[5] C G Atkeson. Learning arm kinematics and dynamics. *Annual review of Neuroscience*, 12:157–183, 1989.

[6] A Barrera and C Laschi. Anticipatory visual perception as a bio-inspired mechanism underlying robot locomotion. *EMBC*, 2010.

[7] J L Barron, D J Fleet, and S S Beauchemin. Performance of optical flow techniques. *IJCV*, 12(1):43–77, 1994.

[8] A Berthoz. *The sense of moviment.* Harvard University Press, 2002.

[9] G Bradski and A Kaehler. *Learning OpenCV.* O'Reilly, 2008.

[10] E Datteri, G Teti, C Laschi, G Tamburrini, P Dario, and E Guglielmelli. Expected perception: an anticipation-based perception-action scheme in robots. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 934–939, October 2003.

[11] E Datteri, G Teti, C Laschi, G Tamburrini, P Dario, and E Guglielmelli. Expected perception in robots: a biologically driven perception-action scheme. *Proceedings of ICAR 2003, 11th International Conference on Advanced Robotics*, 3:1405–1410, 2003.

[12] P F Dominey. Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biol. Cybernetics*, 73:265–274, 1995.

[13] P F Dominey, M Hoen, and T Inui. A neurolinguistic model of grammatical construction processing. *Journal of Cognitive Neuroscience*, 18(12):2088–2107, 2006.

[14] S E Fahlman and C Lebiere. *The Cascade-Correlation learning architecture.* National Science Foundation, 1991.

[15] B Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

[16] R Grasso, S Glasauer, Y Takei, and A Berthoz. The predictive brain: anticipatory control of head direction for the steering of locomotion. *NeuroReport*, 7:1170–1174, 1996.

[17] R Grasso, P Prévost, Y P Ivanenko, and A Berthoz. Eye-head coordination for the steering of locomotion in humans: an anticipatory synergy. *Neuroscience Letters*, 253:115–118, 1998.

[18] H M Gross, V Stephan, and T Seiler. Neural architecture for sensorimotor anticipation. *Cybernetics and Systems Research*, 2:593–598, 1998.

[19] S Haykin. *Neural Networks: a Comprehensive Foundation.* Prentice Hall, 1999.

[20] H Hoffmann. Perception throught visuomotor anticipation in a mobile robot. *Neural Networks*, 20:22–33, 2007.

[21] H Jaeger. The echo state approach to analysing and training recurrent neural networks. *GMD Report*, 148, 2001.

[22] R Johansson. Sensory input and control of grip. *Sensory Guidance of Movements*, pages 45–59, 1998.

[23] M I Jordan and D E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[24] M Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9:718–727, 1999.

[25] T Kohonen. *Self-Organizating Maps.* Springer-Verlag, 1997.

[26] C Laschi, G Asuni, E Guglielmelli, G Teti, R Johansson, M C Carrozza, and P Dario. A bio-inspired neural sensory-motor coordination scheme for robot reaching and preshaping. *Autonomous Robots*, 25:85–101, 2008.

[27] B D Lucas and T Kanade. An iterative image registration tedhnique with an application to stereo vision. *Proceedings of the 1981 DARPA Imaging Understanding Workshop*, pages 121–130, 1981.

[28] W Maass, P Joshi, and E Sontag. Computational aspects of feedback in neural circuits. *PLOS Computational Biology*, 3(1):1–20, 2007.

[29] W Maass, T Natschlaeger, and H Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[30] R C Miall, D J Weir, D M Wolpert, and J F Stein. Is the cerebellum a smith predictor? *Journal of Motor Behavior*, 25(2):203–216, 1993.

[31] R C Miall and D M Wolpert. Forward models for physiological motor control. *Neural Networks*, 9(8):1265–1279, 1996.

[32] T Mitchell. *Machine Learning.* McGraw Hill, 1997.

[33] R R Murphy. *Introduction to AI Robotics.* The MIT Press, 2000.

[34] M J D Powell. The theory of radial basis function approximation in 1990. *Advances in Numerical Analysis*, II:105–210, 1992.

[35] D E Rumelhart and J L McClelland. *Parallel Distribuited Processing.* MIT Press, 1986.

[36] G Sandini, P Metta, and P Fizpatrick. Yarp. *International journal on Advanced Robotics Systems*, 2006.

[37] G Sandini, P Metta, and D Vernon. Robotcub: An open framework for research in embodies cognition. *International Journal of Humanoid Robotics*, 8(2), November 2004.

[38] G Sandini, P Metta, and D Vernon. The icub cognitive humanoid robot: An open-system research platform for enactive cognition. *In 50 Years of AI*, pages 359–370, 2007.

[39] U D Schiller and J J Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63C:5–23, 2005.