UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

ROBOT LEARNING FROM HUMAN DEMONSTRATIONS: CONFRONTO TRA DMM E GMM

RELATORE: Prof. Emanuele Menegatti

CORRELATORE: Dott. Stefano Michieletto

LAUREANDO: Alberto Rizzi

Anno Accademico 2012-2013

Alla mia famiglia
e a tutte le persone che mi hanno sostenuto.

Sommario

L'apprendimento da dimostrazioni consente di istruire un robot a compiere una azione prestabilita. Solitamente l'utente conosce il comportamento e le corrette modalità per insegnarlo al robot. Tuttavia l'utente, in quanto umano, può commettere errori nell'eseguire le dimostrazioni iniziali. Comunemente a quanto avviene nell'apprendimento umano o animale possiamo imparare dai nostri sbagli: anche le dimostrazioni sbagliate, come quelle corrette, contengono informazioni utili. Ad esempio sono indicatori di cosa non imitare. Nell'approccio presentato in [1] e [2] l'obiettivo è di apprendere unicamente dagli errori: a partire dalle dimostrazioni non corrette, si utilizza un modello probabilistico esplorativo, il Donut Mixture Model, per ottenere l'esecuzione corretta del comportamento stabilito inizialmente. Il primo obiettivo del nostro lavoro consiste nell'analisi e nella realizzazione di questo nuovo sistema di apprendimento.

Nella parte innovativa del nostro lavoro consideriamo il Donut Mixture Model nell'apprendimento da dimostrazioni corrette fornite da utenti diversi e catturate attraverso il sensore RGB-D Kinect. Come nel RLfD, vogliamo riprodurre un determinato comportamento a partire da una serie di dimostrazioni corrette, attraverso un robot umanoide. A differenza dell'apprendimento da dimostrazioni corrette, in questa situazione disponiamo di dimostrazioni iniziali corrette ma diverse tra loro: in particolare ogni utente esegue naturalmente e secondo il proprio stile il comportamento, senza conoscerlo alla perfezione. Considerando che il robot gode di un numero minore di gradi di libertà, mappiamo i movimenti umani creando movimenti validi per il robot, ma che non mantengono la proprietà iniziale di correttezza. Ci riconduciamo ad un caso intermedio tra il RLfD e il RLfF: possiamo quindi applicare il modello Donut Mixture confrontandolo rispetto al modello Gaussian Mixture.

Indice

Capitolo 1

Introduzione

Il lavoro svolto per questa tesi di Robotica Autonoma prevede l'analisi di un recente approccio al *Robot Learning from Failure (RLfF)*, ovvero l'apprendimento autonomo di azioni o comportamenti da parte di un robot a partire da dimostrazioni non corrette. Tale tecnica, assieme al più comune apprendimento da dimostrazioni corrette (Robot Learning from Demonstrations - RLfD), costituisce il paradigma generale dell'apprendimento da dimostrazioni, *Learning from Demonstrations*, nell'ambito del *Machine Learning*.

Come si può intuire, una delle differenze sostanziali tra le due tecniche riguarda la modalità di interpretazione delle dimostrazioni:

- il RLfD si fonda sull'idea che i dati forniti dall'utente siano corretti e, quindi, devono essere riprodotti fedelmente dal robot;
- il RLfF presuppone invece che i dati di partenza siano non corretti e, quindi, utilizzabili dal robot come modello negativo da cui discostarsi.

Tuttavia lo scopo conseguito è il medesimo: fare in modo che l'automa esegua correttamente il task perseguito.

La tesi prende spunto dal lavoro descritto nei due paper [1, 2] di Grollman e Billard, nei quali viene introdotto il nuovo approccio al RLfF, focalizzandosi principalmente sulla descrizione del modello probabilistico alla base, il *Donut Mixture Model (DMM)*. Il primo obiettivo del nostro lavoro è stato approfondire tale modello e fornire un'implementazione corretta e robusta dell'intero sistema di apprendimento da fallimenti. Correttezza e robustezza sono state testate cercando di insegnare una semplice azione, come il tiro a canestro (basketball), ad un robot umanoide reale, nel nostro caso il robot Nao[1], a partire da sole dimostrazioni sbagliate. Non limitandoci al solo RLfF, abbiamo successivamente considerato la possibilità di istruire il robot da dimostrazioni fornite da utenti diversi, catturate attraverso il sensore RGB-D Kinect. Inizialmente mappiamo movimenti naturali corretti sul robot umanoide, tenendo in considerazione gli errori introdotti dal sensore e le possibilità di movimento offerte dal robot stesso, limitate dal numero inferiore di gradi di libertà rispetto all'utente umano. Otteniamo nuovi dati che non mantengono necessariamente la proprietà di correttezza iniziale: il nostro secondo obiettivo consiste nel verificare la

[1] Aldebaran Robotics, `http://www.aldebaran-robotics.com/en/`

bontà del comportamento generato sfruttando la nuova tecnica di RLfF, il modello Donut Mixture, e valutandolo rispetto al modello Gaussian Mixture.

Come vedremo dettagliatamente nei prossimi capitoli, il nostro lavoro spazia attraverso una serie di tematiche appartenenti ad ambiti scientifici diversi:

- nell'ambito della robotica autonoma l'apprendimento da dimostrazioni, alcune tecniche di *Machine Learning*, il controllo uomo-macchina;
- nell'ambito della probabilità e statistica, i modelli probabilistici di misture gaussiane (*GMM - Gaussian Mixture Model*) e di misture di funzioni Donut (*DMM - Donut Mixture Model*);
- nell'ambito dell'algebra, le tecniche di ottimizzazione di funzioni multi-variate.

1.1 Stato dell'arte

Entrando nello specifico, descriviamo ora l'evoluzione del Learning from Demonstrations, i concetti fondamentali alla base e infine analizziamo le differenze tra RLfD e RLfF.

Nella Robotica Autonoma e in particolare nell'ambito del Machine Learning (ML), l'apprendimento autonomo da dimostrazioni, conosciuto come *Programming by Demonstrations (PbD)* o *Imitation Learning*, è quella tecnica utile per impartire o per migliorare l'apprendimento ad animali, essere umani, o artefatti, come i robot[3]. Iniziato circa trenta anni fa, a partire dalla prima metà degli anni '80, e cresciuto di importanza negli ultimi dieci anni, il PbD è nato come strumento di aiuto nell'automatizzare la programmazione manuale dei robot nel campo manifatturiero. Gli svantaggi della programmazione manuale dei robot, complessa e tediosa per l'utente stesso, e la crescita progressiva della ricerca scientifica nell'ambito dell'intelligenza artificiale e della robotica, hanno indotto alla creazione di nuovi strumenti e tecniche di apprendimento che portassero vantaggi in termini di efficienza, riduzione della complessità e miglioramento dell'interazione uomo-macchina. Attraverso il PbD la prospettiva di apprendimento cambia radicalmente:

- l'utente è in grado di insegnare un nuovo comportamento impartendo delle dimostrazioni iniziali, ad esempio operando direttamente il robot o tele-operandolo;
- al robot stesso spetta il compito di interpretare i dati di partenza per giungere all'obiettivo desiderato;
- l'apprendimento si basa sulla percezione e azione e la loro interazione;
- il rapporto uomo-macchina si riduce alla fase iniziale, di insegnamento e di costruzione del modello cognitivo, e alla fase successiva di verifica delle riproduzioni del comportamento appreso.

Le sfide da affrontare nel PbD sono le seguenti:

- *cosa imitare* (learning a skill);

- *come imitare* (encoding a skill);
- *quando imitare*;
- *chi imitare.*

1.1.1 Cosa imitare

Diversi sono gli approcci legati a cosa si deve apprendere a partire dalle dimostrazioni iniziali:

- l'utente può fornire le informazioni da imitare esplicitamente, solitamente nel caso di contesti di operabilità nuovi o mai visti;
- dalle informazioni iniziali il robot estrae automaticamente le caratteristiche (*feature*) fondamentali del comportamento da riprodurre.

Nel primo caso siamo in presenza di un tentativo di pura imitazione dell'input fornito dall'utente. Nel secondo caso, più complesso, il robot necessita di un *modello* di apprendimento che gli permetta di riprodurre il task e di una tecnica che ne consenta la valutazione. Nella valutazione può intervenire direttamente l'utente iniziale, come si può notare dallo schema in Figura 1.1, oppure ci si può affidare alla costruzione di una *metrica* (metric of imitation performance). Quest'ultima fornisce un modo per esprimere quantitativamente le intenzioni dell'utente durante le dimostrazioni e per valutare l'esattezza della riproduzione. Solitamente corrisponde ad una funzione di costo da ottimizzare. Utente o metrica possono fungere o meno da feedback per il miglioramento del modello di apprendimento.

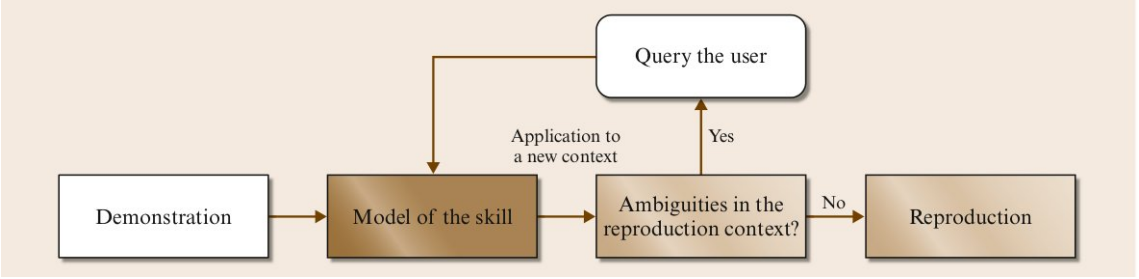


Figura 1.1: Schema base dell'apprendimento da dimostrazioni.

1.1.2 Come imitare

Due sono i principali approcci nella rappresentazione di un'abilità:

Symbolic Encoding

Rappresentazione, ad alto livello, che scompone il comportamento in una sequenza di unità azione-percezione. La conoscenza a priori delle possibili azioni consente una segmentazione efficiente dell'abilità. Il task viene poi ricreato attraverso tecniche di Machine Learning, come ad esempio il *Hidden Markov Model (HMM).*

Trajectories Encoding
Rappresentazione a basso livello dell'abilità, nella forma di una mappa non lineare (traiettoria) tra le informazioni date dai sensori e dai motori. In questo tipo di rappresentazione vengono spesso usate tecniche di riduzione di dimensionalità, come *Principal Component Analysis (PCA)* o *Indipendent Component Analysis (ICA)*, che proiettano le traiettorie registrate in uno spazio di dimensionalità ridotta. Il Trajectories Encoding viene sfruttato principalmente per codificare i movimenti umani.

Vediamo le principali tecniche relative al PbD.

Tecniche basate su modelli statistici

In questo caso la variabilità delle dimostrazioni viene modellata attraverso tecniche di apprendimento non supervisionato (unsupervised learning), ad esempio:

Hidden Markov Model (HMM)
Modello usato per codificare le variazioni temporali e spaziali di segnali complessi e per modellare, riconoscere e riprodurre vari tipi di movimenti.

Mimesis Model
Modello nel quale un singolo HMM codifica un insieme di traiettorie e multiple HMM vengono usate per ottenere movimenti generalizzati, attraverso un processo stocastico.

Gaussian Mixture Model/Regression (GMM/GMR)
GMM, modello composto da un miscuglio di funzioni gaussiane, usato nel RLfD per codificare un insieme di traiettorie. GMR, modello di regressione, usato per generare nuove traiettorie, simili a quelle di ingresso.

Donut Mixture Model (DMM)
Nuovo modello composto da un miscuglio di funzioni Donut, usato nel RLfF per generare nuove traiettorie in grado di discostarsi dalle traiettorie iniziali.

Tecniche basate su sistemi dinamici

Si propongono soluzioni all'insegnamento attraverso processi adatti ai sistemi dinamici, quindi robusti ai cambiamenti. Si sfruttano principalmente tecniche di apprendimento supervisionato (supervised learning), tra le quali *Locally Weighted Regression (LWR)* e *Receptive Field Weighted Regression (RFWR)*. Questi sono algoritmi di Lazy Learning e Memory-based Learning, in cui la generalizzazione sui dati di training, ossia la cattura delle regolarità presenti in questi e la predizione su nuovi dati, viene effettuata successivamente all'interrogazione del sistema, contrariamente a quanto accade nel Eager Learning [4], in cui la funzione obiettivo è calcolata in fase di training del sistema stesso. LWR combina la semplicità della regressione ai minimi quadrati e la flessibilità della regressione non lineare. Tra le proprietà più importanti di questi algoritmi spiccano:

- la velocità nell'apprendere il task;

- la robustezza ai turbamenti, permettendo di aggiustare le traiettorie prodotte on-line;
- l'utilizzabilità on-line per diversi tipi di task, in cui sono richieste solamente modifiche ad un piccolo numero di parametri.

Tecniche di insegnamento graduale

I sistemi che realizzano il PbD dovrebbero essere in grado di apprendere un comportamento nel minor numero di dimostrazioni iniziali possibili. Il robot potrebbe cominciare a riprodurre il task desiderato immediatamente e migliorare la propria esecuzione in modo graduale, automaticamente o grazie all'ausilio dell'utente stesso, che può indicare i punti positivi della riproduzione, da ripetere, o gli errori commessi, da evitare, come nel nostro approccio.

Tecniche di apprendimento per rinforzo

Tra le tecniche di apprendimento più recenti spicca il *Reinforcement Learning (RL)*, nella quale lo scopo consiste nel creare un sistema di apprendimento in grado di adattarsi ai cambiamenti dell'ambiente in cui è inserito, quindi in grado di eseguire lo stesso comportamento in nuove situazioni. L'applicazione del rinforzo si basa sulla valutazione delle prestazioni del sistema stesso, che può avvenire, o meno, sfruttando una metrica di performance.

Tecniche orientate alla biologia

I modelli che appartengono a questo ambito sono tra i più recenti in circolazione e prendono spunto da fenomeni naturali o biologici, come ad esempio le *reti neurali artificiali*, modello basato sulle reti neurali biologiche. Una rete neurale artificiale è un sistema adattivo che cambia la sua struttura basandosi su informazioni esterne o interne che scorrono attraverso la rete durante la fase di apprendimento. In termini pratici, le reti neurali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare.

Alcuni dei modelli correnti si basano su *Mirror Neuron System (MNS)*, con funzionamento simile al cervello. L'idea sostanziale è che una tipologia di comportamento possa essere codificata usando una rappresentazione comune e indipendente dall'utente che lo ha generato. Dunque il sistema corrisponde ad una rete costituita da diverse aeree, come nel caso del cervello, ciascuna in grado di attivarsi per il riconoscimento e la riproduzione dello stesso tipo di movimento.

1.2 RLfD vs RLfF

In questa sezione cerchiamo di mettere in luce gli aspetti che hanno portato all'apprendimento da sole dimostrazioni non corrette.

Come già accennato in precedenza, il RLfD impone che l'apprendimento iniziale avvenga utilizzando dimostrazioni effettuate in modo corretto. Eventuali dimostrazioni non ottimali potrebbero compromettere la corretta costruzione del modello di apprendimento, portando successivamente a risultati errati in fase di riproduzione. Se da un lato il disporre di sole dimostrazioni corrette potrebbe sembrare un enorme vantaggio nel tentare di istruire un robot, dall'altro comporta alcuni svantaggi:

- la quantità di tempo speso dall'utente per imparare a effettuare il task correttamente e per istruire il robot potrebbe essere non indifferente. Inoltre può variare a seconda dell'utente e in modo direttamente proporzionale alla difficoltà del task.

- L'utente necessita di conoscere alla perfezione il comportamento da insegnare. Questa condizione potrebbe non essere strettamente necessaria nella realizzazione di azioni semplici, ad esempio nel caso di azioni che muovono pochi giunti come nell'alzare o abbassare un braccio di robot umanoide. Mentre potrebbe rivelarsi fondamentale nel caso di task più complessi, basta pensare ad esempio alla realizzazione della camminata in un robot umanoide oppure al movimento coordinato dei giunti di un braccio manipolare che gli consenta di prendere e spostare un oggetto nell'ambiente. In questi casi sono indispensabili tecnici adeguati che comportano costi aggiuntivi rispetto all'utente comune.

- Alcuni task possono essere così complessi da non poter essere realizzati sfruttando le sole capacità dell'utente. Ad esempio, potrebbe essere necessario sollevare grossi carichi, o effettuare movimenti innaturali per una persona.

Una riduzione di questi costi e svantaggi si può ottenere cercando di istruire il robot anche attraverso dimostrazioni non andate a buon fine. Nel RLfD tali dimostrazioni possono essere utilizzate nella fase di miglioramento del modello, per aggiustare la politica di apprendimento. In questo caso si considera l'idea che il robot possa imparare dai propri errori, ma pur sempre partendo da una base corretta.

Nell'apprendimento da dimostrazioni non corrette, si stravolge l'idea precedente: si tiene conto della natura umana dell'utente che istruisce il robot. L'utente può anch'esso commettere in buona fede degli errori nella fase di learning, dai quali il robot può trarre informazioni importanti per la realizzazione del task:

- le dimostrazioni andate non a buon fine sono esempi di cosa non fare, quindi da evitare;

- le dimostrazioni andate non a buon fine sono indicative di come l'utente ha interpretato l'esecuzione corretta del task;

- tentativi non a buon fine multipli indicano uno spazio di esplorazione su cui ricercare la corretta soluzione al task.

Come si può intuire, l'esplorazione è uno dei fattori dominanti nel RLfF. Le dimostrazioni anche se sbagliate contengono al loro interno traccia della correttezza: sta al modello di apprendimento scovarla.

1.3 Obiettivi e struttura della tesi

A partire da quest'ultima considerazione ha inizio il lavoro di questa tesi. Nella prima parte verrà descritto dettagliatamente il modello Donut Mixture, l'algoritmo implementato e i test condotti. Nella seconda parte della tesi verrà considerato il modello Donut Mixture come strumento utile per l'apprendimento da dimostrazioni umane naturali, mettendolo a confronto con il modello Gaussian Mixture.

Capitolo 2

Donut Mixture Model

Questo capitolo contiene la descrizione del sistema di apprendimento da dimostrazioni non corrette e del modello probabilistico Donut Mixture su cui si basa, presentati in [2].

2.1 Nuovo approccio al RLfF

L'obiettivo in [2] è di creare un sistema autonomo, in grado di apprendere un determinato comportamento solamente a partire da dimostrazioni non andate a buon fine dello stesso. La linea generale alla base del sistema è tipica del Robot Learning:

1. si colleziona un'insieme di dimostrazioni iniziali;
2. si costruisce il modello di apprendimento;
3. si utilizza il modello per la riproduzione del comportamento;
4. si aggiorna il modello.

Le novità introdotte stanno nella realizzazione del passo 2, quindi nella scelta del modello di apprendimento, e del passo 3, di utilizzo del modello per la fase di riproduzione. Le considerazioni relative al RLfF, e in particolare i tre aspetti evidenziati nella sezione 1.2, intervengono nella costruzione del modello. In particolare le caratteristiche principali che questo deve possedere, sono:

- minimizzare la probabilità che il comportamento eseguito sia errato;
- esplorare lo spazio interno fornito dalla variabilità delle diverse dimostrazioni.

Ad alto livello, dunque, l'algoritmo alla base del sistema si modifica nel seguente modo[1], come illustrato in Figura 2.1:

1. si colleziona un insieme di dimostrazioni iniziali fallite (X);
2. si costruisce il modello θ da queste;
3. il modello genera il nuovo tentativo che **esplora** un intorno delle dimostrazioni ($\bar{x}$);

4. si esegue il tentativo e si aggiorna il modello;

5. si ripetono i passi 3 e 4 fino al successo o alla convergenza.

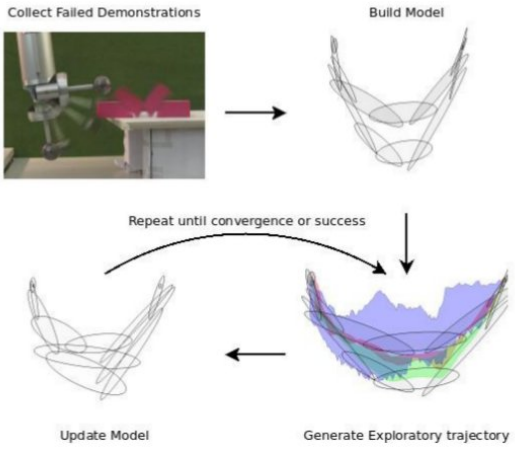


Figura 2.1: Sistema di apprendimento da dimostrazioni non corrette

2.2 Costruzione del modello

Inizialmente, dalle dimostrazioni viene estratto il dataset Ξ di partenza: ad ogni dimostrazione corrisponde una traiettoria, τ_s, costituita da un insieme di coppie posizione-velocità (datapoint), $\tau_s = \{\xi_t, \dot{\xi}_t\}_{t=1}^{T_s}$. ξ rappresenta lo stato D-dimensionale del robot (posizioni dei giunti) e $\dot{\xi}$ le loro velocità. Il dataset è dunque la collezione delle traiettorie, $\Xi = \{\tau_s\}_{s=1}^{S} = \{\xi_n, \dot{\xi}_n\}_{n=1}^{N}$, dove $N = \sum_{s=1}^{S} T_s$, corrisponde al numero di datapoint.

Il dataset viene modellato attraverso un miscuglio di componenti gaussiane (GMM), $\theta = \{K, \{\rho^k, \mu^k, \Sigma^k\}_{k=1}^{K}\}$, definito dai parametri:

K numero di componenti gaussiane;

ρ pesi (priors) associati ai componenti (reali positivi, $\sum_{k=1}^{K} \rho^k = 1$);

μ medie di ogni componente (vettore $2D$-dimensionale);

Σ matrici di covarianza di ogni componente (matrice positiva semi definita di dimensione $2D \times 2D$);

L'idea di fondo è che possiamo considerare la relazione tra le posizioni dei giunti, ξ, e le loro velocità, $\dot{\xi}$, come una funzione non lineare, $\dot{\xi} = f_\theta(\xi)$, rappresentata dal GMM. In particolare, la probabilità di una coppia posizione-velocità è data

$$P_{\texttt{GMM}}(\xi, \dot{\xi}|\theta) = \sum_{k=1}^{K} \rho^k \mathcal{N}(\xi, \dot{\xi}|\mu^k, \Sigma^k) \tag{2.1}$$

Per ottenere il GMM viene eseguito l'algoritmo di Machine Learning chiamato *Expectation Maximization (EM)*: in questo modo tra i possibili valori per i parametri si determinano quelli appartenenti al modello migliore, che più si adatta ai dati in ingresso[5]. Il processo di EM inizia con l'algoritmo *K-means* che consente una prima clusterizzazione dei dati iniziali, calcolando i valori iniziali dei parametri del modello, $\{\rho_0^k, \mu_0^k, \Sigma_0^k\}$. Iterativamente, i parametri vengono raffinati attraverso le due fasi successive di:

- *expectation (E-step)*, in cui sostanzialmente, attraverso il calcolo delle probabilità a posteriori, si ottengono le informazioni relative a quanto i dati sono stati ripartiti correttamente tra i cluster;
- *maximization (M-step)*, in cui si calcolano nuovamente i parametri sfruttando le probabilità a posteriori ottenute al passo precedente.

Alla fine di ogni iterazione si valuta la log-verosimiglianza, $\mathcal{L}(\Xi|\theta)$, del modello ottenuto, rispetto a quella del modello all'iterazione precedente. Il modello finale è quello che determina il valore massimo di log-verosimiglianza.

In generale aumentare K migliora l'adattamento, ma questo potrebbe portare ad un eccessivo adattamento del modello ai dati (overfitting). Per evitare questo fenomeno si usa l'algoritmo *BIC (Bayesian Information Criterion)*.

A questo punto possiamo sfruttare il modello per il calcolo delle velocità a partire dalle posizioni, $\dot{\xi} = f_\theta(\xi)$. In particolare, condizioniamo il GMM ottenuto rispetto alla posizione, ottenendo la funzione di densità condizionata delle velocità, in formula 2.2.

$$P_{\texttt{GMM}}(\dot{\xi}|\xi,\theta) = \sum_{k=1}^{K} \tilde{\rho}^k(\xi,\theta)\mathcal{N}(\dot{\xi}|\tilde{\mu}^k(\xi,\theta),\tilde{\Sigma}^k(\theta)) \tag{2.2}$$

$$\tilde{\mu}^k(\xi,\theta) = \mu_{\dot{\xi}}^k + \Sigma_{\dot{\xi}\xi}^k \Sigma_{\xi\xi}^{k^{-1}}(\xi - \mu_\xi^k) \tag{2.3}$$

$$\tilde{\Sigma}^k(\theta) = \Sigma_{\dot{\xi}\dot{\xi}}^k - \Sigma_{\dot{\xi}\xi}^k \Sigma_{\xi\xi}^{k^{-1}} \Sigma_{\xi\dot{\xi}}^k \tag{2.4}$$

$$\tilde{\rho}^k(\xi,\theta) = \frac{\rho^k \mathcal{N}(\xi;\mu_\xi^k,\Sigma_{\xi\xi}^k)}{\sum_{k=1}^{K}\rho^k\mathcal{N}(\xi;\mu_\xi^k,\Sigma_{\xi\xi}^k)} \tag{2.5}$$

Il risultato è anch'esso un GMM, con parametri $\{\tilde{\mu}^k, \tilde{\Sigma}^k, \tilde{\rho}^k\}$ derivati dai parametri del modello iniziale e dipendente dalla posizione, che può essere utilizzato per generare le velocità probabilisticamente, ad esempio campionando, o deterministicamente, attraverso l'aspettazione. Ad esempio, nel caso di dimostrazioni con successo, i valori di velocità possono essere generati a partire dall'aspettazione della distribuzione condizionata precedente

$$f_\theta^{\texttt{success}}(\xi) = \tilde{E}[\dot{\xi}|\xi,\theta] = \sum_{k=1}^{K}\tilde{\rho}^k\tilde{\mu}^k \tag{2.6}$$

in modo da garantire un movimento scorrevole e quindi evitare rapide oscillazioni del robot.

Usare il valore atteso della distribuzione condizionata ha senso solo quando i dati osservati sono ugualmente distribuiti attorno al successo, una forte assunzione nel caso di dimostrazioni fallite. Considerando le dimostrazioni non favorevoli, è preferibile calcolare *traiettorie esploratorie*, invece che medie, dipendenti dalla varianza. In particolare, come si può notare in Figura 2.2, le traiettorie generate seguono l'andamento delle dimostrazioni nelle aree a bassa varianza, mentre si discostano in quelle dove la varianza è alta.

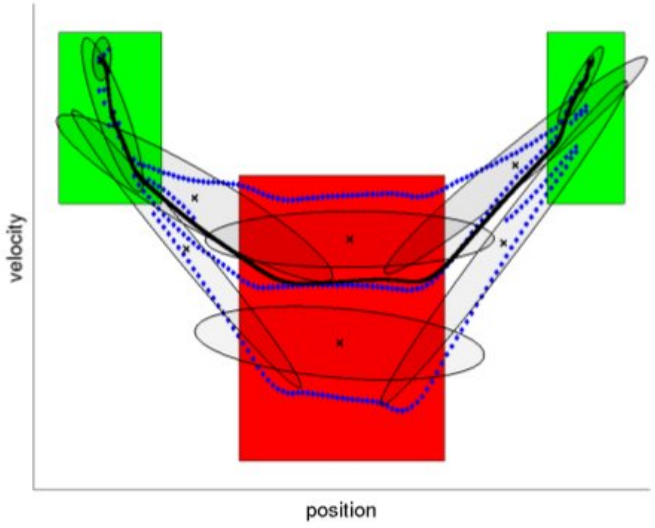


Figura 2.2: In verde le aree a bassa varianza tra le dimostrazioni, in rosso quelle ad alta varianza, su cui spazia il modello. In blu e tratteggiate le traiettorie generate dal modello, in nero la traiettoria media.

Il modello Donut Mixture (DMM) consente di realizzare questo comportamento. Derivato dal GMM, tale modello è costituito da un miscuglio di funzioni *Donut*, in figura 2.3, la cui distribuzione di probabilità è definita come differenza tra due funzioni *Normali*

$$\mathcal{D}(\vec{x}|\mu_\alpha, \mu_\beta, \Sigma_\alpha, \Sigma_\beta, \gamma) \quad = \quad \gamma\mathcal{N}(\vec{x}|\mu_\alpha, \Sigma_\alpha) - (\gamma - 1)\mathcal{N}(\vec{x}|\mu_\beta, \Sigma_\beta) \tag{2.7}$$

dove $\gamma > 1$, nel nostro caso impostato a 2. L'obiettivo è quello di ottenere una distribuzione di probabilità il cui andamento è caratterizzato tipicamente dalla presenza di tre punti di ottimo: due massimi locali e tra questi una regione in cui la funzione decresce raggiungendo un punto di minimo locale. Nel nostro caso consideriamo $\mu_\alpha = \mu_\beta$, e parametrizziamo la distribuzione Donut sfruttando due nuovi parametri r_α e r_β, dipendenti dall'esplorazione, così che $\Sigma_\alpha = \frac{1}{r_\alpha^2}\Sigma, \Sigma_\beta = \frac{1}{r_\beta^2}\Sigma$. Si ottiene

$$\mathcal{D}(\vec{x}|\mu, \Sigma, r_\alpha, r_\beta, \gamma) \quad = \quad \gamma\mathcal{N}(\vec{x}|\mu, \Sigma/r_\alpha^2) - (\gamma - 1)\mathcal{N}(\vec{x}|\mu, \Sigma/r_\beta^2) \tag{2.8}$$

Questa nuova formula ci consente di rimpiazzare ogni componente gaussiano della distribuzione condizionata con una funzione Donut, mantenendo gli stessi parametri

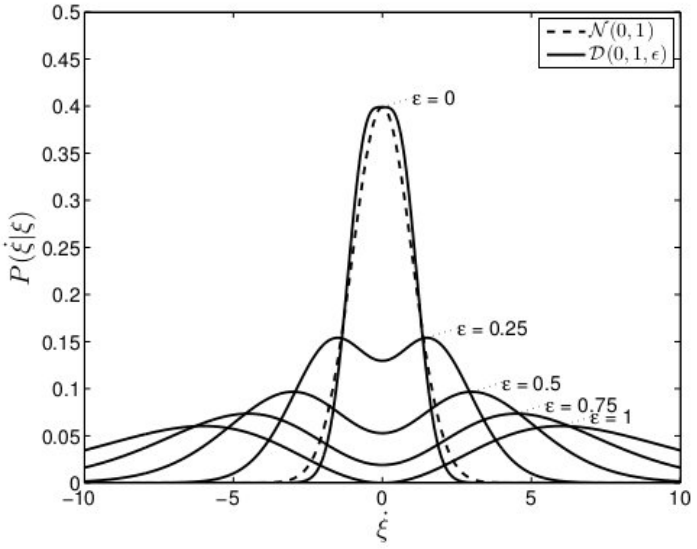


Figura 2.3: Confronto tra una funzione di densità Normale, tratteggiata, e le corrispondenti funzioni Donut, al variare del fattore di esplorazione ϵ.

$\{\rho^k, \mu^k, \Sigma^k\}$. La distribuzione condizionata del DMM risulta

$$P_{\texttt{DMM}}(\dot{\xi}|\xi) = \sum_{k=1}^{K} \tilde{\rho}^k \mathcal{D}(\dot{\xi}|\tilde{\mu}^k, \tilde{\Sigma}^k, \epsilon) \tag{2.9}$$

in cui ϵ ($0 \leq \epsilon \leq 1$) rappresenta il fattore di esplorazione, dipendente dalla varianza complessiva del modello Gaussian Mixture

$$\epsilon = 1 - \frac{1}{1 + ||\tilde{V}[\dot{\xi}|\xi, \theta]||} \tag{2.10}$$

$$\tilde{V}[\dot{\xi}|\xi, \theta] = -\tilde{E}[\dot{\xi}|\xi, \theta]\tilde{E}[\dot{\xi}|\xi, \theta]^\top + \sum_k \rho_k(\mu_k \mu_k^\top + \Sigma_k) \tag{2.11}$$

Così facendo si crea il legame tra la varianza delle dimostrazioni iniziali e l'esplorazione dello spazio posizioni-velocità. Osservando la formula 2.10, si può notare che il comportamento del sistema rientra in quello preposto:

- quando ϵ tende a 1, ci troviamo nella situazione di massima esplorazione, in caso di aree ad alta variabilità;
- quando ϵ tende a 0, ci troviamo nella situazione di minima esplorazione, in caso di aree a bassa variabilità.

Inoltre dal grafico in Figura 2.3, nel caso di esplorazione minima, il comportamento del DMM si riduce ad un comportamento molto simile a quello del corrispondente GMM, e quindi al caso di apprendimento da successo.

2.3 Generazione dei tentativi

L'approccio in [2] prevede di generare le traiettorie predicendo le velocità iterativamente, a partire da una posizione iniziale. In particolare, ad ogni iterazione (vedi algoritmo 1)

1. si costruisce la distribuzione di probabilità del DMM, equazione 2.9, a partire dalla posizione corrente dei giunti interessati, ξ_t^*;

2. si determina la velocità, $\dot{\xi}_t^*$, che massimizza la densità di probabilità, sfruttando la tecnica di ottimizzazione del gradiente ascendente;

3. si aggiorna la posizione corrente dei giunti sommando la velocità ottenuta alla posizione precedente, $\xi_{t+1}^* = \xi_t^* + \dot{\xi}_t^*$.

4. si ripetono i passi 1-3 fino al raggiungimento di un criterio di arresto.

Algorithm 1 Approccio al learning from failure basato sul DMM

```
Collect S human failed attempts
Build a GMM (θ) as described in Section 2.2.
while robot has not succeeded do {Robot trials}
  t = 0
  ξ_t = Current state of the system
  ξ̇_t ~ DMM(ξ̇|ξ, θ)
  while |ξ̇| ≠ 0 and not timeout do
    Maximize P(ξ̇_t|ξ_t) with gradient ascent
    apply ξ̇ to system
    t = t + 1
    ξ_t = Current state of the system {Nominally, ξ_{t−1} + ξ̇_{t−1}}
    ξ̇_t = ξ̇_{t−1} {Start gradient ascent here}
  end while
  update θ as in Section 2.4
end while
```

L'utilizzo di una tecnica di ottimizzazione nel passo 2 è dovuta al fatto che non esiste una funzione analitica per il calcolo dei massimi di un DMM. In questo caso si applica il gradiente ascendente, che garantisce di trovare un massimo, ma non necessariamente un massimo globale. Per ovviare a questo problema, il gradiente ascendente comincia la propria ricerca da un valore di velocità casuale alla prima iterazione, mentre nelle iterazioni successive parte dal valore di massimo ottenuto precedentemente.

Come indicato in [1], l'iterazione si arresta quando:

- il valore di velocità determinato, $\dot{\xi}^*$, è nullo;

- si è superata una lunghezza temporale prestabilita, 1.5 volte la lunghezza della dimostrazione più lunga.

2.4 Aggiornamento del modello

Poiché si apprende da dimostrazioni non corrette, le traiettorie generate potrebbero essere sbagliate e quindi utilizzabili come dimostrazioni da cui apprendere. In [2] vengono proposti due approcci all'aggiornamento del modello:

- *approccio naive*, nel quale si utilizzano tutti i dati non corretti (dimostrazioni e tentativi) e si stima nuovamente il modello θ usando EM, la cui computazione aumenta man mano che incrementa il numero di dati in ingresso;

- *approccio di campionamento e fusione*, nel quale dato il modello θ iniziale costituito da N datapoint e una traiettoria generata τ di N' datapoint, si scelgono N' campioni, pesati $\frac{N'}{N}$, dal dataset iniziale, si aggiungono i dati provenienti da τ, con peso unitario, creando un nuovo dataset, sul quale applicare nuovamente EM, senza K-means, per creare un nuovo modello θ'. In questo modo si mantiene il numero di componenti gaussiane K costante.

Capitolo 3

Implementazione

In questo capitolo presentiamo l'implementazione, da noi realizzata, riguardante il sistema di apprendimento da fallimenti introdotto nel capitolo precedente. La linea seguita si basa principalmente sull'algoritmo definito in [1] e in [2], il cui pseudocodice è presentato nell'algoritmo 1.

Tuttavia il nostro lavoro non è stato un tentativo di copia di quanto già realizzato. Ciò che lo contraddistingue sono le soluzioni adottate e le modifiche introdotte riguardo ad alcune fasi dell'algoritmo: ad esempio, non abbiamo assolutamente modificato il modello Donut Mixture, ma abbiamo analizzato e proposto una diversa soluzione alla fase di ottimizzazione, esaminando ulteriori tecniche.

Le modifiche introdotte sono la conseguenza dell'obiettivo da noi preposto: abbiamo deciso di puntare alla creazione di un sistema funzionante, tralasciando l'efficienza o la possibilità di una esecuzione on-line. L'approccio di *Grollman* e *Billard* prevede di poter applicare direttamente al sistema (il robot) le velocità ottenute, in fase di esecuzione. Noi preferiamo invece operare off-line, andando prima a creare le nuove traiettorie e successivamente verificando i risultati ottenuti attraverso un'analisi matematica. Infine eseguiamo le nuove traiettorie prima in simulazione e, in seguito, attraverso il robot. La simulazione consente di ottenere un primo riscontro grafico dei movimenti che si andranno ad eseguire: possiamo dunque evitare spiacevoli inconvenienti che potrebbero verificarsi usando direttamente il robot. A differenza di quando svolto in [1] in cui viene utilizzato un robot manipolatore, nel nostro caso sfruttiamo un robot umanoide. Dunque un controllo fondamentale riguarda la verifica della stabilità durante le esecuzioni: movimenti troppo bruschi potrebbero far oscillare o far cadere il robot danneggiandolo. Inoltre possiamo valutare che i valori di velocità e di posizione non oltrepassino le soglie stabilite: se ciò accadesse rischieremmo di compromettere i motori di movimento dei giunti.

L'implementazione è stata realizzata sfruttando il middleware *ROS*, Robot Operating System[1], nel linguaggio di programmazione C++. I test sono stati effettuati attraverso ROS, in particolare per la simulazione sul robot tramite l'ambiente *Gazebo*[2], mentre per la visualizzazione grafica dei risultati si è optato per l'utilizzo di Matlab[3]. I test sul robot reale sono stati eseguiti utilizzando il robot umanoide Nao,

[1] `http://www.ros.org/wiki/`

[2] vedi `http://gazebosim.org/documentation.html` o quanto descritto in [6]

[3] `http://www.mathworks.it/it/help/matlab/`

prodotto dalla Aldebaran Robotics, integrato in ROS.

3.1 Implementazione dei modelli

Nella fase di inizializzazione, l'algoritmo crea il GMM a partire dall'insieme di punti delle traiettorie iniziali e successivamente il DMM. Per la creazione del primo abbiamo utilizzato il pacchetto ROS `GaussianMixture` già sviluppato dallo IAS-Lab[4], testato e migliorato in alcune funzionalità, mentre per il secondo abbiamo creato un nuovo pacchetto, denominato `DMM`.

3.1.1 Il pacchetto `GaussianMixture`

All'interno di questo pacchetto è contenuto il codice necessario per la creazione del relativo modello. La classe `GaussianComponent` implementa la funzione di distribuzione Normale multivariata, fornendo i metodi per il calcolo della densità di probabilità e per settare o ottenere i valori della media, covarianza e peso (priors). La classe `GaussianMixture` realizza la mistura di gaussiane, come vettore di componenti gaussiane.

Per ottenere i parametri $\{\tilde{\mu}^k, \tilde{\Sigma}^k, \tilde{\rho}^k\}$ del modello a partire dai dati iniziali, si utilizzano i metodi della classe `EM_OpenCV`, che sfrutta le librerie *OpenCV*[5] per realizzare l'algoritmo EM. In particolare si inseriscono le traiettorie di partenza con il metodo `addData()` e successivamente si richiama il metodo `process()` per l'esecuzione vera e propria di EM. In quest'ultimo, per prima cosa, viene eseguito l'algoritmo K-means grazie al metodo `train()`, in modo da ottenere una prima clusterizzazione dei dati e valutazione dei parametri, i quali poi vengono raffinati grazie al metodo `trainE()` che, effettua le iterazioni di aspettazione e massimizzazione.

Alla fine si ottiene un modello con parametri fittati ai dati di ingresso. Uno dei problemi legati a questa implementazione è che non viene utilizzata nessuna tecnica per individuare il corretto numero di componenti gaussiane, K. Il numero di componenti viene fissato precedentemente alla fase di creazione.

Individuazione del numero delle componenti gaussiane

Per ovviare al problema precedente, la soluzione da noi proposta prevede l'utilizzo di tecniche di Information Criterion, basate sul calcolo della verosimiglianza, tra cui *AIC (Akaike Information Criterion)* e *BIC (Bayesian Information Criterion)*. Queste tecniche infatti sono spesso usate per determinare il numero appropriato di componenti per un modello, quando questo non è specificato a priori. Entrambe sfruttano la log-verosimiglianza (*log-likelihood*), come strumento per indicare il fitting del modello sui dati: maggiore è il valore di verosimiglianza e maggiore è l'adattamento del modello. Entrambe le tecniche evitano che il modello ecceda nell'overfitting, e perciò in una dipendenza troppo stretta dai dati iniziali, introducendo un termine penalizzante nel valore di likelihood. Come di può osservare dalle formule

[4]Intelligent Autonomous Systems Laboratory, `http://robotics.dei.unipd.it/`

[5]`http://opencv.org/`

3.1 e 3.2 questo fattore è più elevato in BIC e meno accentuato in AIC.

$$\texttt{BIC} = 2 * NlogL + m * log(n) \tag{3.1}$$
$$\texttt{AIC} = 2 * NlogL + 2 * m \tag{3.2}$$
$$L(\Xi|\theta) = ln\mathcal{L}(\Xi|\theta) \tag{3.3}$$

- n, numero dei dati con cui viene allenato il modello;
- D, dimensionalità dei dati;
- K, numero di componenti gaussiane;
- m, numero di parametri liberi, ottenuto come $m = (K-1) + K(D + 0.5 * D(D+1))$;
- $NlogL$, negative log-likelihood del modello sugli n dati.

Poichè OpenCV non permette di calcolare la verosimiglianza correttamente, abbiamo creato un metodo apposito per ottenere il numero corretto delle componenti. A seconda del criterio scelto, BIC o AIC, il metodo `bestK()` applica iterativamente l'algoritmo EM ai dati di partenza, incrementando ad ogni iterazione il numero delle componenti iniziali K e calcolando per ogni modello restituito il valore del criterio. Infine si ottiene come modello maggiormente fittato quello che minimizza il valore del criterio.

In tabella 3.1 vediamo il confronto tra i risultati di entrambi i criteri applicati su uno stesso dataset iniziale e la differenza tra la nostra implementazione e quella di OpenCV. In figura 3.1 possiamo notare la clusterizzazione e i GMM prodotti da EM, basandosi sul numero di componenti gaussiane fornite rispettivamente dalle tecniche BIC e AIC.

K	*LL OpenCV*	*LL C++/Matlab*	*BIC*	*AIC*
1	-2445.49	1648.85	-3263.56	-3287.71
2	-2303.91	2496.43	-4917.73	-4970.86
3	-2195	3004.96	-5893.81	-5975.92
4	-2130.45	3254.1	-6351.12	-6462.21
5	-2085.27	3374.99	-6551.92	-6691.98
6	-2054.06	3449.61	-6660.18	-6829.22
7	-2028.74	3491.57	-6703.12	-6901.14
8	-2012.2	3520.89	-6720.77	-6947.77
9	-2003.83	**3543.34**	**-6724.71**	-6980.69
10	-1994.72	**3554.89**	-6706.82	**-6991.78**
11	-1993.04	3553.53	-6663.13	-6977.06
12	-1980.82	3566.79	-6648.67	-6991.59
13	-1980.38	3568.71	-6611.52	-6983.42
14	-1979.95	3570.67	-6574.46	-6975.33
15	-1968.53	3579.89	-6551.93	-6981.78
16	-1971.84	3575.24	-6501.65	-6960.48
17	-1965.57	3582.16	-6474.5	-6962.31
18	-1972.59	3571.85	-6412.9	-6929.69
19	-1967.33	3580.02	-6388.28	-6934.05
20	-1968.79	3578.72	-6344.69	-6919.43

Tabella 3.1: Confronto tra la log-likelihood (LL) calcolata con OpenCV e dal metodo `likelihood` nel pacchetto `DMM`, al variare del numero di componenti (K). Confronto tra BIC e AIC al variare di K: in blu evidenziato il miglior K determinato da BIC, in giallo quello da AIC. Utilizzato il dataset ideale descritto in sezione 4.1.1.

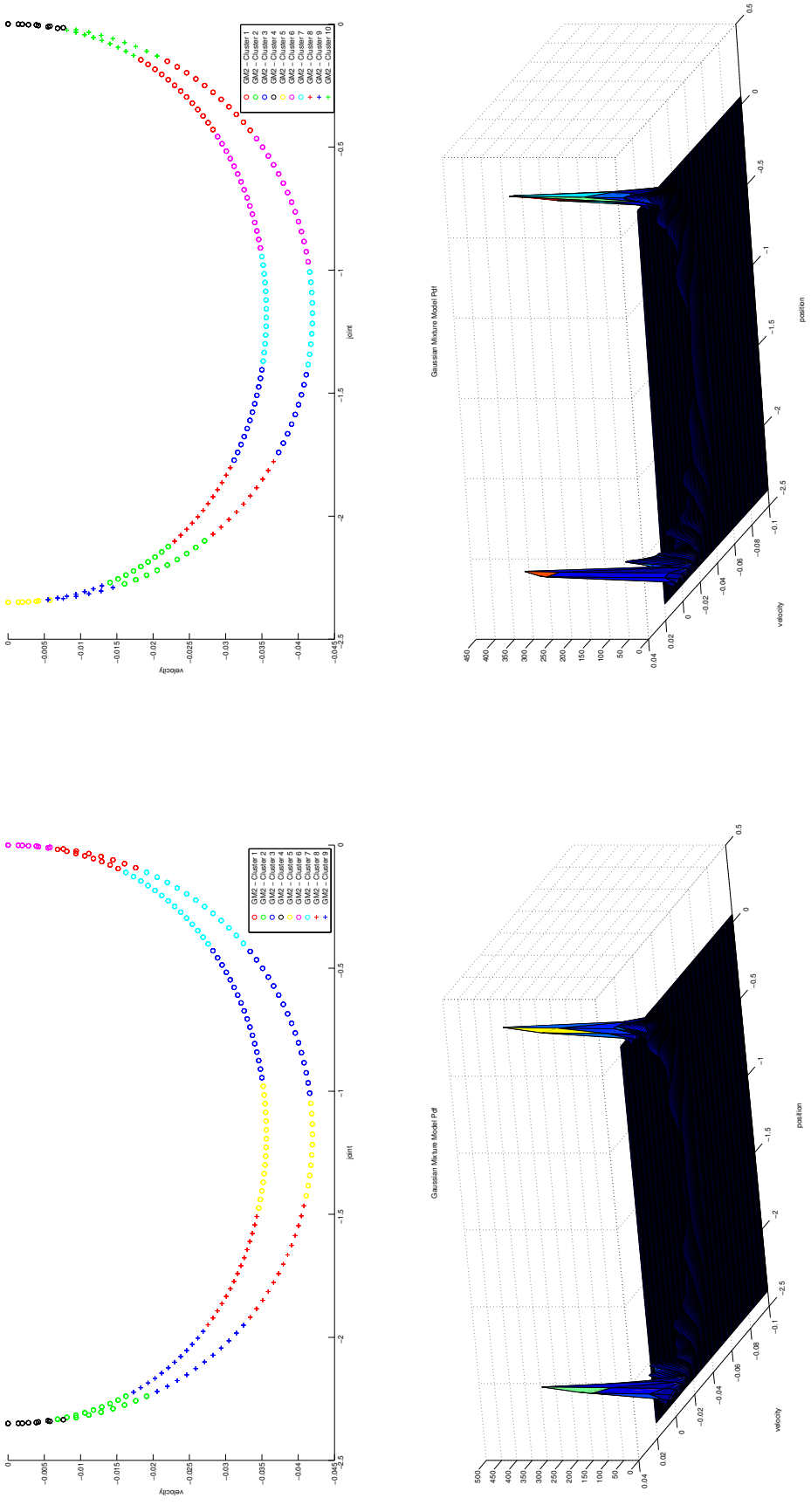


Figura 3.1: Risultati ottenuti nella fase di EM sugli stessi dati iniziali, a partire dai valori ottenuti dalle due tecniche BIC e AIC. A sinistra è raffigurata la clusterizzazione (sopra) e il GMM (sotto) ottenuto dal numero di componenti restituito da BIC (9), mentre a destra da AIC (10). Utilizzato il dataset ideale descritto in sezione 4.1.1.

3.1.2 Il pacchetto `DMM`

All'interno di questo pacchetto sono contenuti le classi e i metodi per la realizzazione del modello Donut Mixture e dell'ottimizzazione, vedi figura 3.2. In particolare la classe `DonutComponent` contiene la realizzazione della funzione di densità Donut, con i metodi relativi al calcolo della densità di probabilità e del gradiente, in formula 3.4, e i metodi per settare e ottenere i parametri, come la media, la covarianza, il peso. La classe `DonutMixture` realizza il modello come insieme di funzioni Donut, su cui è possibile richiamare i metodi per calcolare la funzione di densità totale e il gradiente totale, come somma dei gradienti delle singole componenti.

$$\begin{aligned} \nabla_x \mathcal{D}(\vec{x}|\mu, \Sigma_\alpha, \Sigma_\beta, \gamma) &= -\gamma \mathcal{N}(\vec{x}|\mu, \Sigma_\alpha)\Sigma_\alpha^{-1}(\vec{x} - \mu) \\ &+ (\gamma - 1)\mathcal{N}(\vec{x}|\mu, \Sigma_\beta)\Sigma_\beta^{-1}(\vec{x} - \mu) \end{aligned} \tag{3.4}$$

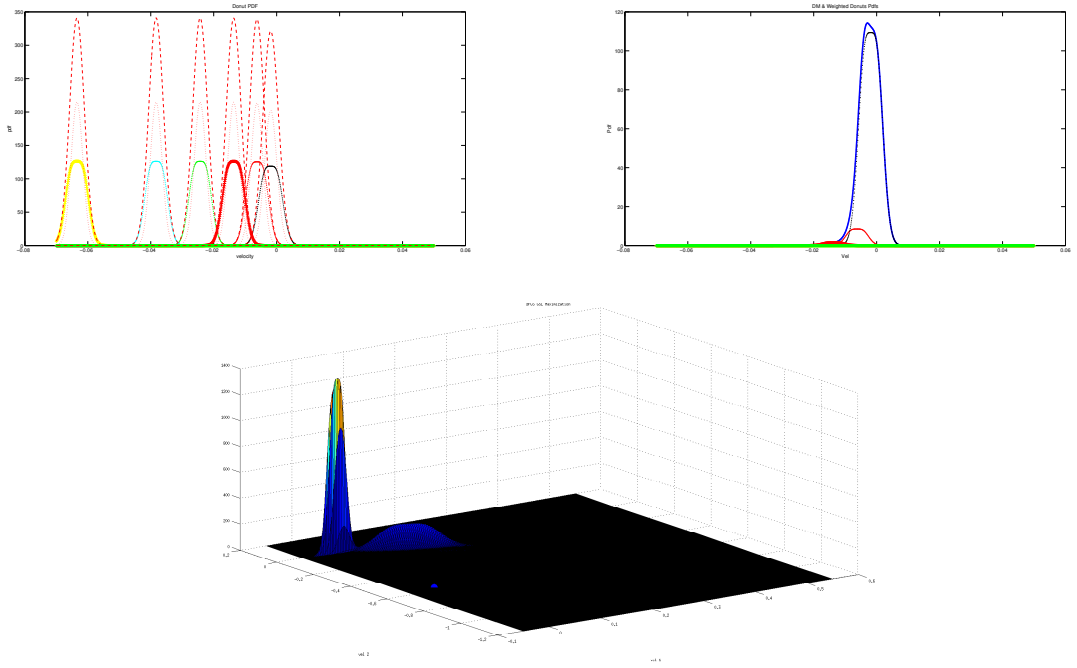


Figura 3.2: Esempio di funzioni Donut e Donut Mixture bidimensionali, in alto. A sinistra rappresentiamo l'insieme delle componenti Donut non pesate: ogni funzione è rappresentata assieme alle due componenti gaussiane, in rosso tratteggiato. A destra rappresentiamo le componenti precedenti pesate e in blu il DMM ottenuto dalla loro combinazione. In basso un esempio di Donut Mixture tridimensionale.

3.1.3 Parametri condizionati

Ottenuto il GMM iniziale, il nostro approccio procede come in [2], calcolando i parametri della distribuzione Gaussian Mixture condizionata sulle posizioni dei giunti (vedi formule 2.3, 2.4 e 2.5). L'implementazione è realizzata all'interno del metodo `transform()` della classe `TildeTransformation`, che riceve in ingresso il GMM iniziale e lo stato dei giunti raggiunto e restituisce un GMM costituito dai parametri della distribuzione condizionata. A questo punto possiamo sostituire le componenti gaussiane con le Donut e quindi creare il DMM.

3.2 Ottimizzazione

Per calcolare i valori di velocità dalla posizione dobbiamo ottimizzare la funzione di densità del DMM, in particolare eseguire una massimizzazione, in modo da ottenere, ad ogni iterazione, la velocità con la maggior probabilità di essere riprodotta. La tecnica di ottimizzazione utilizzata in [2] è quella del gradiente ascendente (steepest gradient), mentre la nostra soluzione ricade sull'utilizzo dell'algoritmo *BFGS (Broyden Fletcher Goldfarb Shannon)*. Siamo andati a considerare la tecnica migliore, valutando i risultati prodotti da diverse tecniche[6], tra cui i metodi Quasi-Newton o quelli basati sul simplesso. La differenza tra i due consiste nelle modalità:

- nei Quasi-Newton, si utilizzano le informazioni relative al gradiente e alla matrice hessiana della funzione da ottimizzare, per ricostruire la pendenza e la curvatura della funzione stessa e quindi calcolare la giusta direzione di ricerca del minimo;
- nei metodi basati sul simplesso, come il metodo di Nelder Mead, si parte da un vettore iniziale a n dimensioni per costruire un simplesso a $n + 1$ vertici, che attraverso trasformazioni geometriche, come la riflessione, contrazione, espansione, si muove verso il minimo, dove si contrae.

L'implementazione è contenuta nella classe `Optimization` e realizzata attraverso la libreria *GSL (GNU Scientific Library)*[7]. Le tecniche di ottimizzazione offerte da GSL sono[8]

- algoritmo del gradiente coniugato[9] di Fleetcher-Reeves;
- algoritmo del gradiente coniugato di Polak-Ribiere;
- algoritmo BFGS[10];
- algoritmo del gradiente discendente (Steepest Descent)[11];
- algoritmo di Nelder-Mead[12].

dove i primi quattro appartengono alla tipologia dei Quasi-Newton e l'ultimo del simplesso.
GSL offre un framework comune per la convergenza verso il minimo. Iterativamente, dopo aver definito il minimizzatore:

1. si inizializza lo stato del minimizzatore, s, per l'algoritmo di ottimizzazione T;
2. si aggiorna lo stato s;

[6] vedi [7, 8] e link Matlab Unconstrained Nonlinear Optimization
[7] `http://www.gnu.org/`
[8] vedi link GSL Multidimensional Minimization
[9] vedi [9, 10, 11]
[10] vedi [12, 13, 14, 15]
[11] vedi link Gradiente discendente
[12] vedi [16]

3. si testa la convergenza dello stato s.

Al passo 1, il minimizzatore s viene inizializzato a minimizzare una determinata funzione f a partire da un punto di partenza x. Al passo 2, il minimizzatore procede iterativamente con l'esecuzione dell'algoritmo *line search*[13] si cerca lungo la direzione d di ricerca il punto x' che minimizza la funzione f, solitamente quando il gradiente g della funzione in quel punto è ortogonale alla direzione di ricerca d, rispetto a un valore di tolleranza, *tol*. Infine al passo 3 si controlla se il nuovo punto x' corrisponde al minimo, situazione che si verifica quando il gradiente è nullo, altrimenti si procede con la ricerca.

Per effettuare la massimizzazione invece che la minimizzazione della funzione, come nel nostro caso, basta invertire sia la funzione che il gradiente e applicare nuovamente l'ottimizzazione.

Da notare è che gli algoritmi di minimizzazione sono in grado di scoprire solo un minimo alla volta, non importa se questo sia locale o globale. Quando ci sono numerosi minimi nell'area di ricerca, il primo minimo trovato viene restituito. Questo dipende principalmente dal punto di partenza dell'ottimizzazione e dalla curvatura della funzione nel proprio intorno. Si possono presentare due scenari:

- il punto di partenza è situato in una zona a curvatura nulla, quindi con gradiente nullo, tale da terminare immediatamente la ricerca;
- la funzione presenta vari minimi locali e globali e il punto di partenza è situato nell'intorno di un minimo locale.

Nel caso dell'ottimizzazione del DMM la prima situazione si verifica quando la ricerca parte solitamente da un punto a bassa densità di probabilità. La seconda invece accade frequentemente nel caso in cui il DMM sia costituito da picchi isolati tra loro: in questo caso la ricerca procede verso il minimo più vicino e potrebbe non individuare il minimo globale.

Per ridurre questi problemi proponiamo tre soluzioni:

- nella prima, denominata *Check Start*, si effettua un controllo del punto di partenza, precedente all'ottimizzazione;
- nella seconda, denominata *Multi-way Search*, si effettua una ricerca del minimo a partire da più punti;
- nella terza, si effettua una combinazione tra le prime due.

Il primo approccio prevede la verifica che il punto di partenza sia situato in una zona non a curvatura nulla, controllando che il gradiente non sia nullo. In caso positivo si procede all'ottimizzazione. In caso negativo allora si procede iterativamente nel seguente modo:

1. si considera una regione attorno al punto di raggio r;
2. si individuano casualmente N punti;

[13] vedi [7, 17]

3. per ciascun punto x_i si determina la norma all'infinito del gradiente[14];

4. si verifica di avere almeno un punto con norma superiore ad una certa soglia, *tol*.

In caso positivo si termina l'iterazione e si considera il primo punto trovato come il nuovo punto da cui partire. Altrimenti si aggiornano il raggio, $r' = 2r$, e il numero di punti, $N' = 2N$ e si prosegue con una nuova iterazione, fino a convergenza o al superamento del massimo numero di iterazioni.

Il secondo approccio invece prevede di eseguire N ottimizzazioni a partire da N punti e di utilizzare le informazioni ottenute per convergere al minimo o massimo globale. Nel caso del DMM i punti di partenza corrispondono alle K medie delle K componenti Donut. L'idea alla base è che il DMM, per come è costruito, possiede almeno una componente non nulla, le cui medie risiedono in zone a curvatura non nulla, anche se questo dipende molto dal fattore di esplorazione e dalla struttura del modello. Infatti, in questo caso non si parla di tecnica di risoluzione completa, ma di riduzione dei problemi precedenti.

In figura 3.3 confrontiamo le tecniche di ottimizzazione presenti in GSL, tabella di sinistra, e in Matlab, tabella di destra: consideriamo come elementi valutativi il numero di iterazioni per convergere al punto di massimo, l'avvenuta convergenza, il punto di massimo determinato, il valore della funzione in quel punto e infine la tipologia dell'ottimo individuato (globale o locale). Possiamo notare come la tecnica del gradiente ascendente (SD), sfruttata in [2], non sempre converge e in caso positivo più lentamente rispetto alle altre tecniche. Le tecniche migliori risultano BFGS, Conjugate PR e Conjugate FR.

	GSL					
x_0		*BFGS*	*BFGS2*	*Conj. PR*	*Conj. FR*	*SD*
0	N. Iters	2	2	2	2	194
	Convergence	Yes	Yes	Yes	Yes	No
	x_{max}	-0.0029	-0.0029	-0.0029	-0.0029	(-0.0029)
	$f(x_{max})$	114.23	114.23	114.23	114.23	(114.23)
	Loc/Glob	G	G	G	G	-
-0.02	N. Iters	4	2	4	4	194
	Convergence	Yes	Yes	Yes	Yes	No
	x_{max}	-0.0029	-0.0029	-0.0029	-0.0029	(-0.0029)
	$f(x_{max})$	114.23	114.23	114.23	114.23	(114.23)
	Loc/Glob	G	G	G	G	-
0.02	N. Iters	5	1	5	5	194
	Convergence	Yes	No	Yes	Yes	No
	x_{max}	-0.0029	(0.02)	-0.0029	-0.0029	(-0.0029)
	$f(x_{max})$	114.23	0.0	114.23	114.23	(114.23)
	Loc/Glob	G	-	G	G	-

	MATLAB			
x_0		*BFGS*	*DFP*	*SD*
0	Function count	15	15	30
	Convergence	Yes	Yes	Yes
	x_{max}	-0.0029	-0.0029	-0.0029
	$f(x_{max})$	114.23	114.23	114.23
	Loc/Glob	G	G	G
-0.02	N. Iters	15	15	18
	Convergence	Yes	Yes	Yes
	x_{max}	-0.0029	-0.0029	-0.0029
	$f(x_{max})$	114.23	114.23	114.23
	Loc/Glob	G	G	G
0.02	N. Iters	1	1	1
	Convergence	No	No	No
	x_{max}	(0.02)	(0.02)	(0.02)
	$f(x_{max})$	0.0	0.0	0.0
	Loc/Glob	-	-	-

Figura 3.3: Confronto tra le tecniche di ottimizzazione in GSL, a sinistra, e in Matlab, a destra. Eseguiamo l'ottimizzazione dello stesso DMM a partire da tre punti di partenza diversi $x_0 = 0, x_0 = 0.02, x_0 = -0.02$, senza utilizzare nessuna tecnica ausiliaria. Il DMM è costruito a partire dai dati di sezione 4.1.1, durante la generazione della prima traiettoria.

In figura 3.4 possiamo valutare graficamente i risultati precedenti relativi alle tecniche di ottimizzazione di GSL. Per ogni riga abbiamo i risultati della stessa

[14]La norma all'infinito corrisponde al massimo del valore assoluto del gradiente della funzione.

tecnica applicata a partire da $x_0 = 0$, a sinistra, $x_0 = 0.02$, al centro, e $x_0 = -0.02$, a destra. Per ogni colonna possiamo valutare i risultati delle tecniche a partire dallo stesso punto di partenza. Nelle tabelle 3.2, 3.3 e 3.4 abbiamo considerato la tecnica BFGS, applicandola a partire da una zona a curvatura nulla, considerando o meno le tre tecniche ausiliarie descritte in precedenza. I risultati grafici sono riportati in figura 3.5. Possiamo notare il miglioramento ottenuto: non utilizzando alcuna tecnica l'ottimizzazione si arresta immediatamente determinando il massimo nel punto di partenza; utilizzando le tecniche otteniamo il massimo corretto, a discapito di un aumento della computazione, maggiore nel Multi-way.

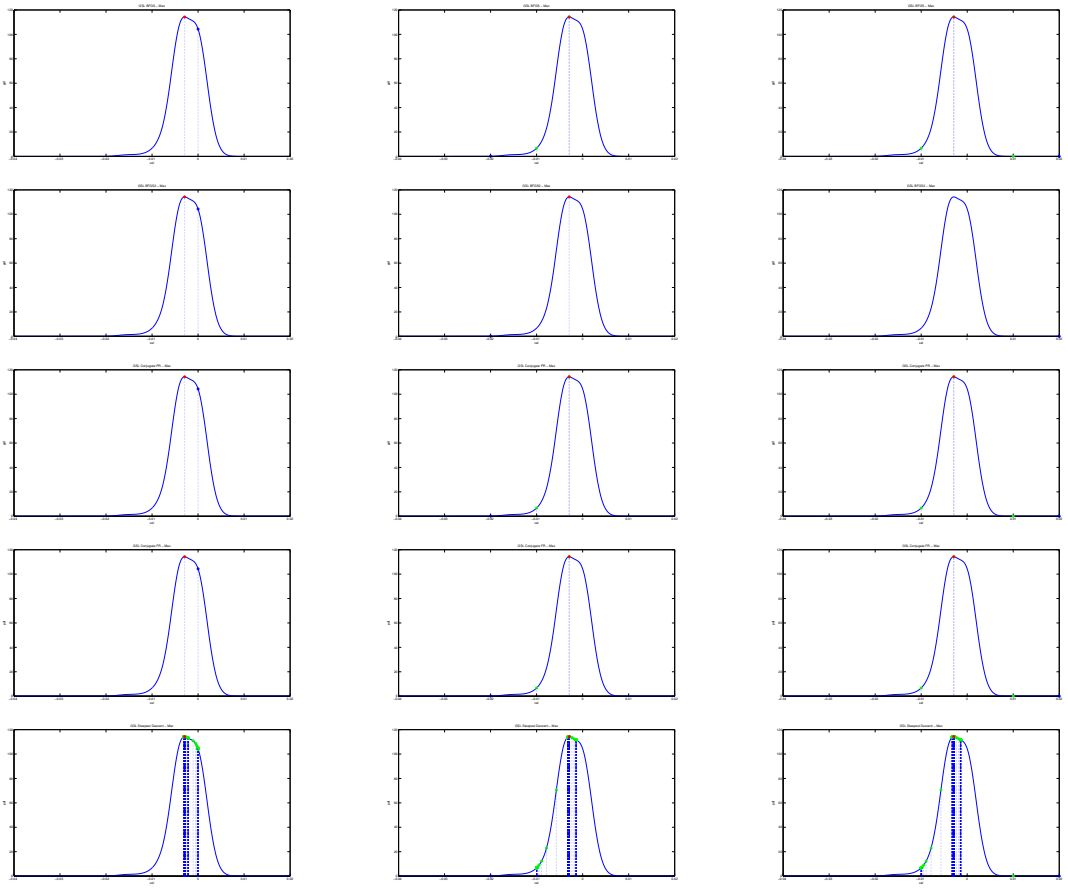


Figura 3.4: Confronto tra le tecniche GSL. Ogni riga raffigura i risultati della stessa tecnica applicata a partire dai tre valori di x_0; ogni colonna raffigura le tecniche a partire dallo stesso punto, rispettivamente la tecnica BFGS, BFGS2, Conjugate PR, Conjugate FR e Steepest Descent dall'alto verso il basso. In ogni figura il pallino blu indica il punto di partenza dell'ottimizzazione, quello rosso il punto di massimo trovato e quelli verdi i punti intermedi determinati durante la procedura iterativa.

		BFGS			
x_0		*Nothing*	*StartCheck*	*Multi-Way(*)*	*SC + MW(*)*
	SC Iters	-	4	-	68
	New x_0	-	-0.019	-	-
	g(x_0)	0.0	124.98	-	-
	Opt. Iters	1	3	24	31
-0.07	Conv.	No	Yes	Yes	Yes
	x_{max}	(-0.07)	-0.0029	-0.0029	-0.0029
	$f(x_{max})$	0.0	114.23	114.23	114.23
	Loc/Glob	-	G	G	G

Tabella 3.2: Risultati delle soluzioni ai problemi di ottimizzazione. StartCheck (SC) parameters: `r` = 0.005, `N` = 10, `maxiters` = `50`. (*)Per Multi-Way (MW) e combinazione SC con MW vedere tabelle 3.3 e 3.4

	MULTI-WAY						
	x_{mean}	$g(x_{mean})$	*Opt. iters*	*Conv.*	x_{max}	$f(x_{max})$	*L/G*
DC 1	-0.0062	26597.8	3	Yes	-0.0029	114.23	G
DC 2	-0.25	0	1	No	(-0.25)	0	-
DC 3	-0.024	0.24	5	Yes	-0.0029	114.23	G
DC 4	-0.66	0	1	No	-0.66	0	-
DC 5	-0.038	$7.29 * 10^{-6}$	5	Yes	-0.0029	114.23	G
DC 6	-0.001	-2062.36	2	Yes	-0.0029	114.23	G
DC 7	-0.013	216.03	3	Yes	-0.0029	114.23	G
DC 8	-0.063	$-2.48 * 10^{-19}$	2	Yes	-0.053	$-1.47 * 10^{-17}$	L
DC 9	-0.114	0	2	Yes	-0.10	$-1.73 * 10^{-84}$	L

Tabella 3.3: Soluzione Multi-Way

	STARTCHECK + MULTI-WAY							
	SC iters	*New* x_0	$g(x_0)$	*Opt. iters*	*Conv.*	x_{max}	$f(x_{max})$	*L/G*
DC 2	8	-0.031	1.25	4	Yes	-0.0029	114.23	G
DC 4	50	-	-	1	No	(-0.66)	0	-
DC 5	1	-0.034	0.048	5	Yes	-0.0029	114.23	G
DC 8	3	-0.034	0.027	5	Yes	-0.0029	114.23	G
DC 9	5	0.006	-1597.3	3	Yes	-0.0029	114.23	G

Tabella 3.4: Soluzione combinata SC con MW, elencando solo le Donut che comportano un cambiamento nell'ottimizzazione

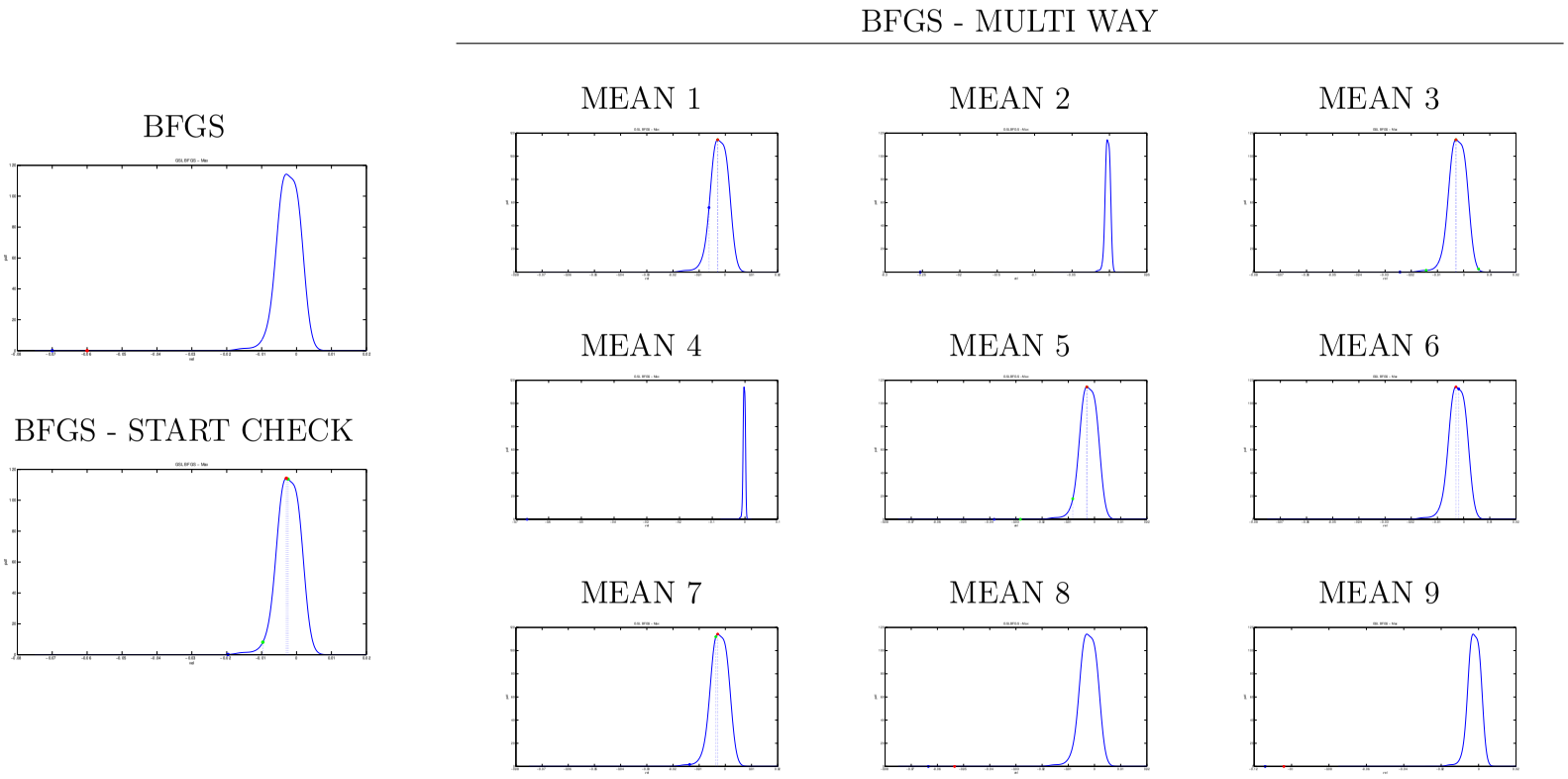


Figura 3.5: Confronto tra l'ottimizzazione senza tecniche ausiliarie e applicata assieme alla tecnica di Start Check, sulla sinistra, mentre applicata assieme alla tecnica Multi Way a destra. Il punto di partenza $x_0 = -0.07$ è situato in una zona a curvatura nulla.

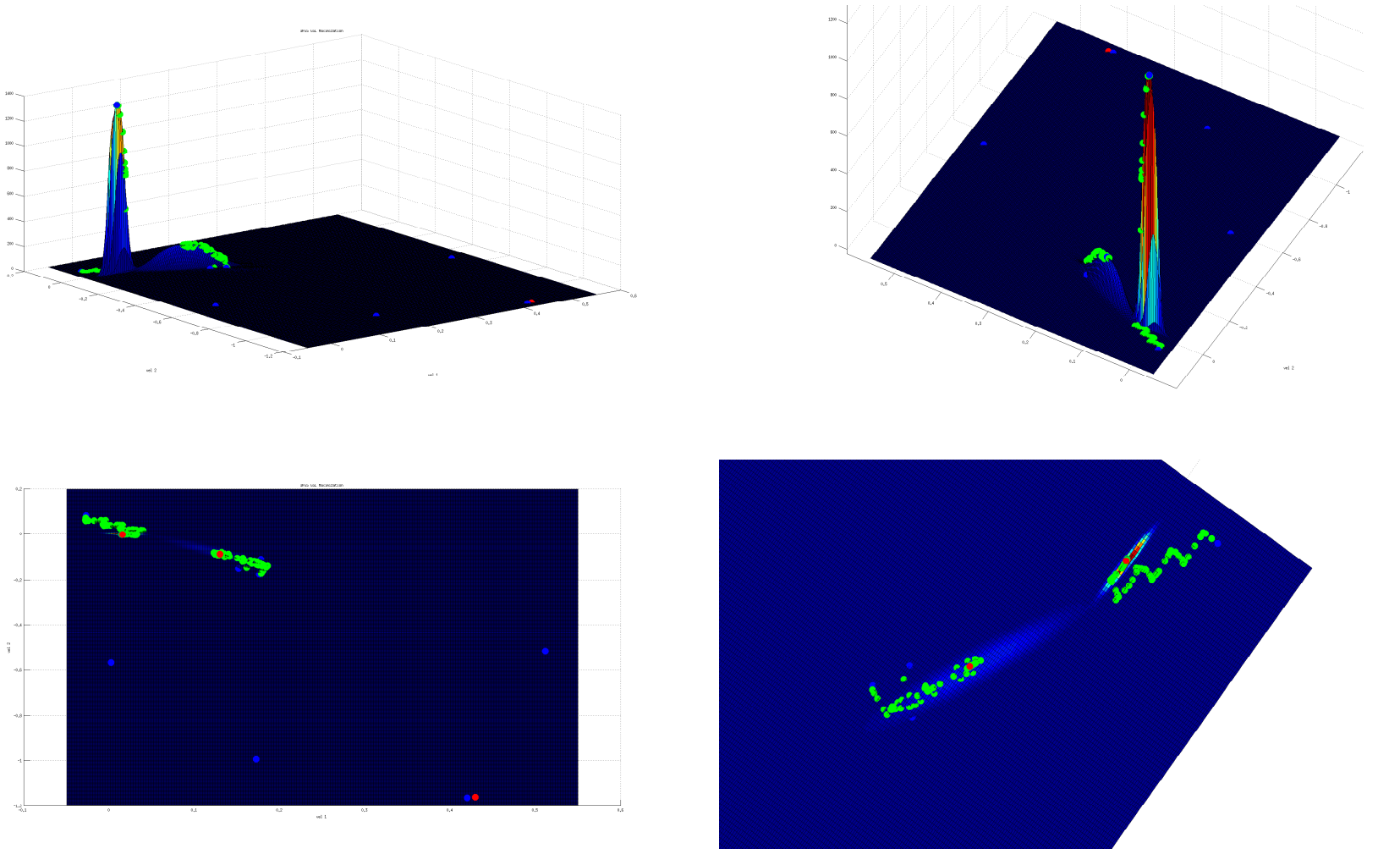


Figura 3.6: Esempio di ottimizzazione BFGS, utilizzata con tecnica Multi-way e Start Check, applicata su un DMM a tre dimensioni. I pallini blu indicano le posizioni dei 10 punti di partenza, le medie delle Donut, in verde i punti intermedi della fase di ottimizzazione e in rosso la posizione dei punti di ottimo determinati.

3.3 Criteri di arresto

Ottenuta la velocità, la si utilizza per il calcolo dello stato successivo del robot: le nuove posizioni dei giunti corrispondono alla somma tra la velocità restituita dalla massimizzazione e le posizioni precedenti. Prima di procedere con una nuova iterazione del sistema e quindi ricominciare con la il calcolo della GMM condizionata a partire dal nuovo stato, si eseguono le seguenti valutazioni su posizioni e velocità:

- si controlla se la velocità ottenuta è nulla[2];
- per ogni giunto si controlla se il nuovo valore di posizione supera i limiti stabiliti.

Nel primo caso l'arresto del sistema avviene quando tutte le componenti della velocità sono nulle o al di sotto di una soglia stabilita. Nel secondo invece il sistema si arresta quando tutte le posizioni hanno oltrepassato le soglie massime o minime stabilite per ogni giunto: in particolare nel momento in cui una singola posizione oltrepassa una soglia, vengono aggiornati sia i valori di posizione, fissato al valore della soglia, sia della velocità, alla differenza tra il valore di soglia con la posizione precedente. Questo tipo di controllo serve ad evitare la creazione di traiettorie che possano compromettere la stabilità o più drasticamente il funzionamento del robot o di una parte di esso.

Un ulteriore criterio di arresto riguarda la lunghezza della traiettoria generata[1]: la procedura di creazione termina nel momento in cui il numero di punti che costituiscono la nuova traiettoria supera di una volta e mezzo quello della traiettoria di ingresso più lunga.

3.4 Aggiornamento del sistema

Per quanto riguarda l'aggiornamento del sistema, non disponendo di un algoritmo on-line che ci permetta di avere un riscontro immediato di quanto prodotto, ci limitiamo a considerare i tentativi creati come sbagliati da inserire nel dataset delle traiettorie iniziali e da utilizzare nel tentativo successivo. In questo modo andiamo a rispettare l'approccio naive introdotto nella sezione 2.4.

Capitolo 4

Test

In questo capitolo presentiamo i risultati dei test relativi al sistema di apprendimento introdotto. I test sono stati pensati per essere effettuati sul Nao, sia lavorando in simulazione sfruttando Gazebo che con il robot reale. Per l'installazione dei driver del Nao e il suo utilizzo attraverso ROS si prenda visione della tesi di laurea di Davide Zanin [18] e del materiale al link `http://www.ros.org/wiki/nao/Tutorials/Getting-Started`.

Nel test il robot doveva apprendere come giocare a pallacanestro: a partire dalle sole dimostrazioni non corrette il robot deve riuscire a centrare il canestro facendo rimanere la pallina al suo interno. Il movimento compiuto dal Nao consiste nel lanciare la pallina muovendo i giunti del braccio destro: nel primo caso abbiamo mosso solamente il giunto della spalla destra (shoulder pitch) in avanti; nel secondo caso abbiamo realizzato il movimento similmente a un movimento naturale, muovendo in combinazione la spalla destra (pitch) con il gomito destro (roll).

4.1 Traiettorie ideali

Come primi test, ancor prima di lavorare con il Nao, abbiamo provato il sistema su traiettorie ideali, auto generate, di un solo giunto. Ideali nel senso che l'andamento della posizione e della velocità potesse rispecchiare quello che si avrebbe in una situazione reale non sporcata da rumore: nel nostro caso abbiamo usato un andamento parabolico per la velocità, con valori nulli nei punti iniziali e finali, e per la posizione un andamento rappresentabile da una polinomio di terzo grado, essendo la velocità derivata della posizione. I datapoint sono stati ottenuti campionando le traiettorie con lo stesso intervallo di tempo Δt.

Nei test abbiamo considerato traiettorie di due tipi:

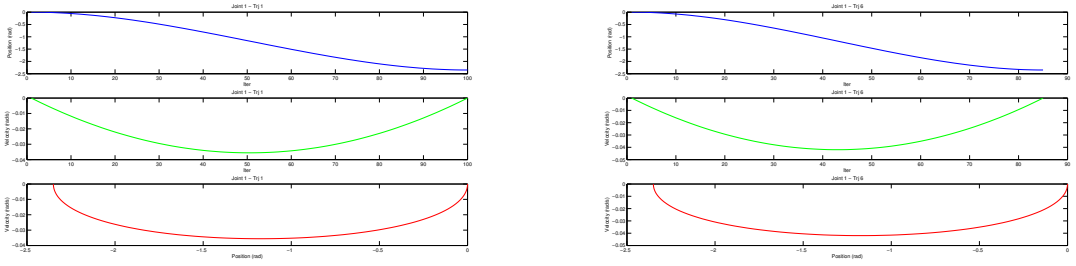
- traiettorie con posizioni iniziali e finali comuni, raggiunte con velocità diverse;
- traiettorie con posizioni iniziali e finali diverse, raggiunte nello stesso tempo.

4.1.1 Posizioni costanti

Supponiamo di utilizzare un singolo giunto che si muove dalla posizione iniziale, di 0 rad, alla posizione finale di -2.35 rad. Campioniamo i datapoint con $\Delta t = 1$

secondo. Consideriamo un dataset di 10 traiettorie iniziali suddivise in traiettorie di due tipi:

- lente, con velocità minima maggiore di 0.04 rad/s e di 100 datapoint;
- veloci, con velocità minima minore di 0.04 rad/s e di 85 datapoint.



Traiettoria lenta Traiettoria veloce

Figura 4.1: Tipi di traiettorie caratterizzate dalla stessa posizione iniziale e finale. In blu la traiettoria della posizione nel tempo, in verde la velocità nel tempo, in rosso la traiettoria posizione-velocità.

Traiettorie generate

Generiamo 15 traiettorie a partire dal dataset iniziale. I risultati sono contenuti in tabella 4.1 e nelle figure 4.2 e 4.3. Come si può notare le nuove traiettorie, rappresentate dalle linee rosse, esplorano lo spazio posizione-velocità delimitato dalle traiettorie iniziali, in blu tratteggiato. Le traiettorie generate seguono inoltre un andamento parabolico che si avvicina molto a quello delle iniziali.

Attempt	*Input Traj.*	K_{best}	*Datapoint*	*Final* ξ	$\dot{\xi}_{min}$	*Stop criteria(*)*
1	10	9	102	-2.40	-0.038	Ths
2	11	8	108	-2.37	-0.039	Vel
3	12	8	108	-2.37	-0.039	Vel
4	13	8	107	-2.37	-0.039	Vel
5	14	8	107	-2.37	-0.039	Vel
6	15	8	107	-2.37	-0.039	Vel
7	16	9	107	-2.37	-0.039	Vel
8	17	9	107	-2.37	-0.038	Vel
9	18	8	107	-2.37	-0.039	Vel
10	19	8	107	-2.37	-0.039	Vel
11	20	9	107	-2.37	-0.039	Vel
12	21	9	107	-2.37	-0.039	Vel
13	22	9	107	-2.37	-0.039	Vel
14	23	9	107	-2.37	-0.039	Vel
15	24	9	107	-2.37	-0.039	Vel

Tabella 4.1: Risultati dei 15 tentativi generati. (*)I valori disponibili per il criterio di arresto sono Ths (threshold), nel caso di superamento della soglia minima o massima della posizione del giunto, Vel (velocity), nel caso di velocità nulla e Time, nel caso di superamento del limite temporale. Vel si verifica quando $|\dot{\xi}| \leq 0.0002$, mentre le soglie di massimo e minimo sono stabilite a 0 e -2.40 rad.

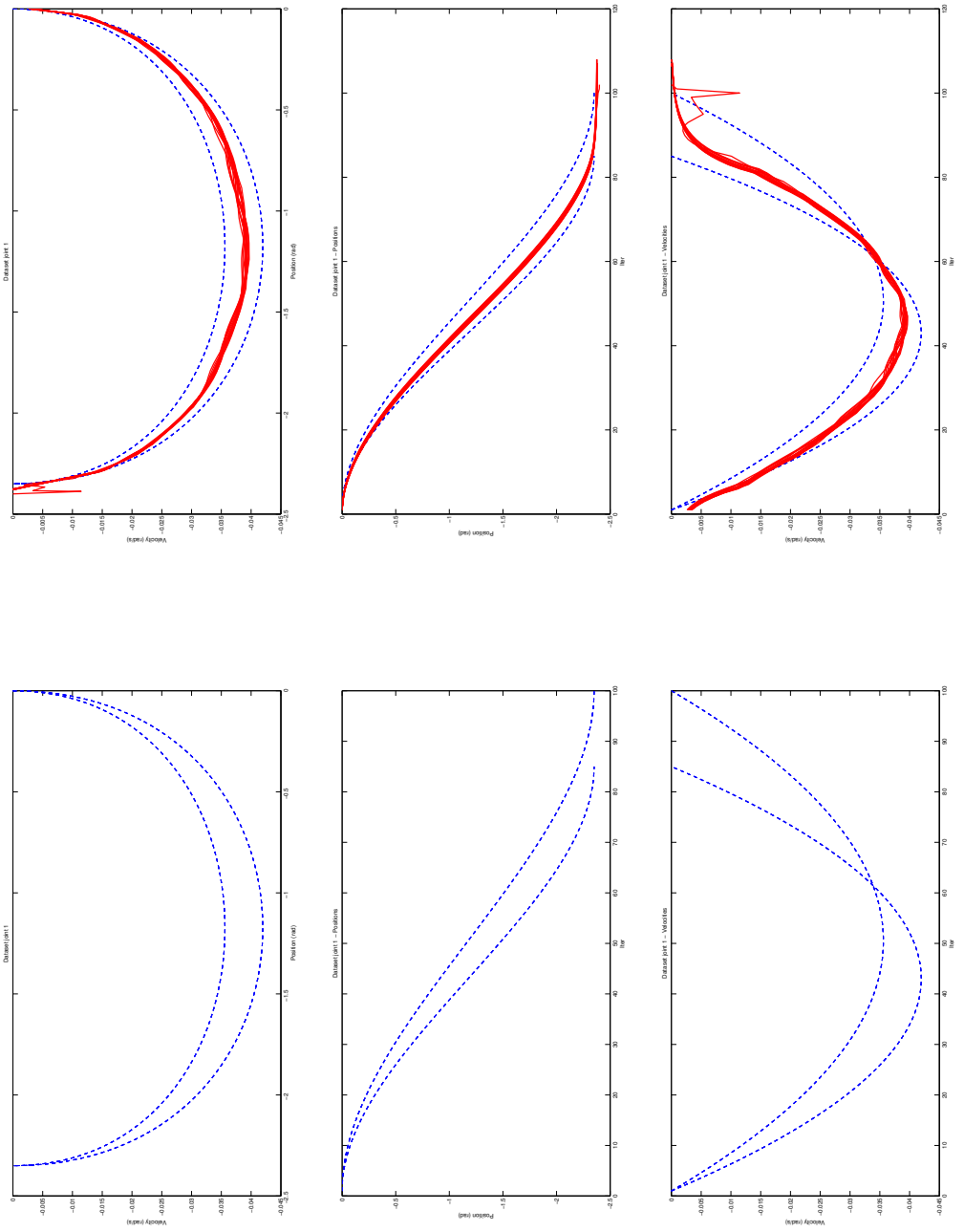


Figura 4.2: Confronto tra il dataset iniziale, contenente le 10 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM.

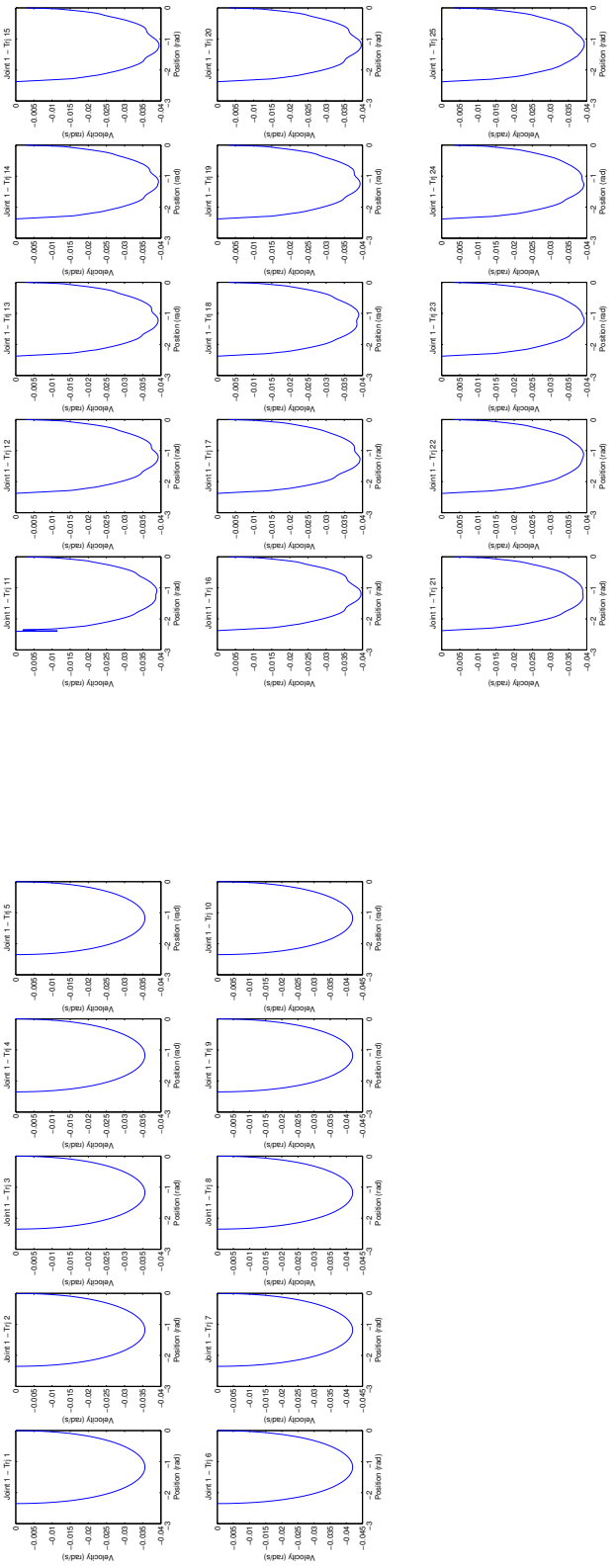


Figura 4.3: Confronto tra il dataset iniziale e il dataset generato, suddiviso in traiettorie singole nello spazio posizione-velocità.

4.1.2 Posizioni diverse

Andiamo ora ad osservare i risultati nel caso di traiettorie che cominciano nella stessa posizione, 0 rad, ma finiscono in posizioni diverse, con tempo di percorrenza uguale. Anche in questo caso utilizziamo traiettorie iniziali di due tipologie, vedi figura 4.4:

- quelle la cui posizione termina nel valore -2.35 rad;
- quelle che terminano in -2.5 rad.

Campioniamo le traiettorie in 100 datapoint ciascuna con intervallo $\Delta t = 1$ secondo.

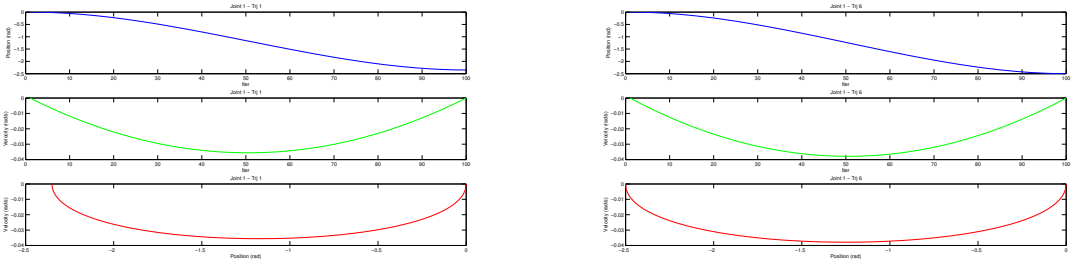
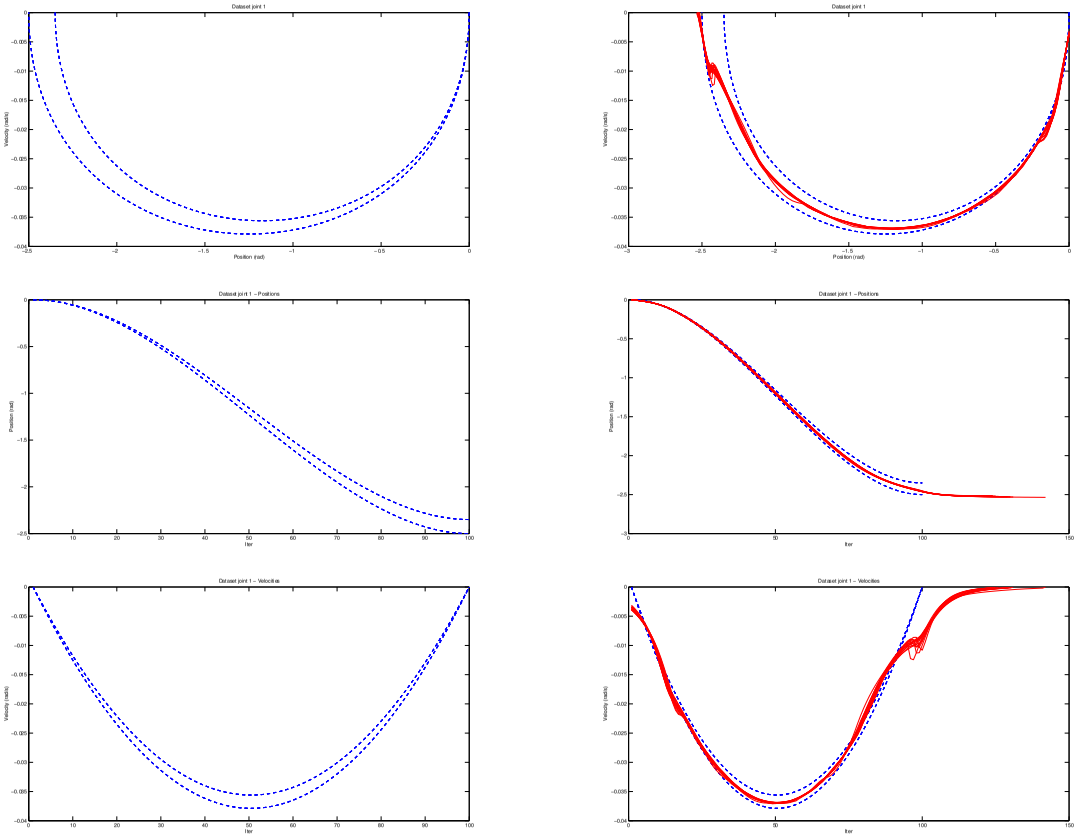


Figura 4.4: Tipi di traiettorie caratterizzate dalla stessa posizione iniziale ma posizioni finale diverse, -2.35 rad a sinistra e -2.5 rad a destra. In blu la traiettoria della posizione nel tempo, in verde la velocità nel tempo, in rosso la traiettoria posizione-velocità.

Traiettorie generate

Generiamo 15 traiettorie a partire dal dataset iniziale. I risultati sono contenuti nelle figure 4.5 e 4.6. Anche in questo caso si può notare che le nuove traiettorie esplorano lo spazio posizione-velocità delimitato dalle traiettorie iniziali. In particolare l'esplorazione si mantiene fino alla posizione di -2.35 rad, oltre la quale il nuovo andamento segue quello della traiettoria più lunga. Questo aspetto è dovuto ai dati di partenza: se osserviamo lo spazio posizione-velocità relativo alle traiettorie iniziali possiamo notare una regione ampia tra le posizioni di 0 e -2.35 rad, contenente i datapoint di entrambe le traiettorie, e una regione più sottile, tra le posizioni di -2.35 e -2.5 rad, costituita solamente dai datapoint della traiettoria più lunga. Dunque nella creazione del GMM otteniamo un modello influenzato dai dati di entrambe le traiettorie inizialmente e dai dati della traiettoria più lunga nelle posizioni che differiscono.



Attempt	*Input Traj.*	K_{best}	*Datapoint*	*Final* ξ	$\dot{\xi}_{min}$	*Stop criteria*
1	10	10	124	-2.51	-0.036	Vel
2	11	11	124	-2.51	-0.036	Vel
3	12	10	124	-2.51	-0.036	Vel
4	13	13	125	-2.51	-0.036	Vel
5	14	13	124	-2.52	-0.036	Vel
6	15	11	126	-2.52	-0.036	Vel
7	16	12	125	-2.52	-0.036	Vel
8	17	9	129	-2.52	-0.036	Vel
9	18	9	129	-2.52	-0.036	Vel
10	19	11	129	-2.52	-0.037	Vel
11	20	9	131	-2.53	-0.036	Vel
12	21	9	130	-2.52	-0.036	Vel
13	22	9	130	-2.53	-0.037	Vel
14	23	9	130	-2.52	-0.036	Vel
15	24	9	142	-2.53	-0.039	Vel

Figura 4.5: Confronto tra il dataset iniziale, contenente le 10 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM. In tabella le informazioni relative ai 15 tentativi generati. Vel si verifica quando $|\dot{\xi}| \leq 0.0002$, mentre le soglie di massimo e minimo sono stabilite a 0 e -2.60 rad.

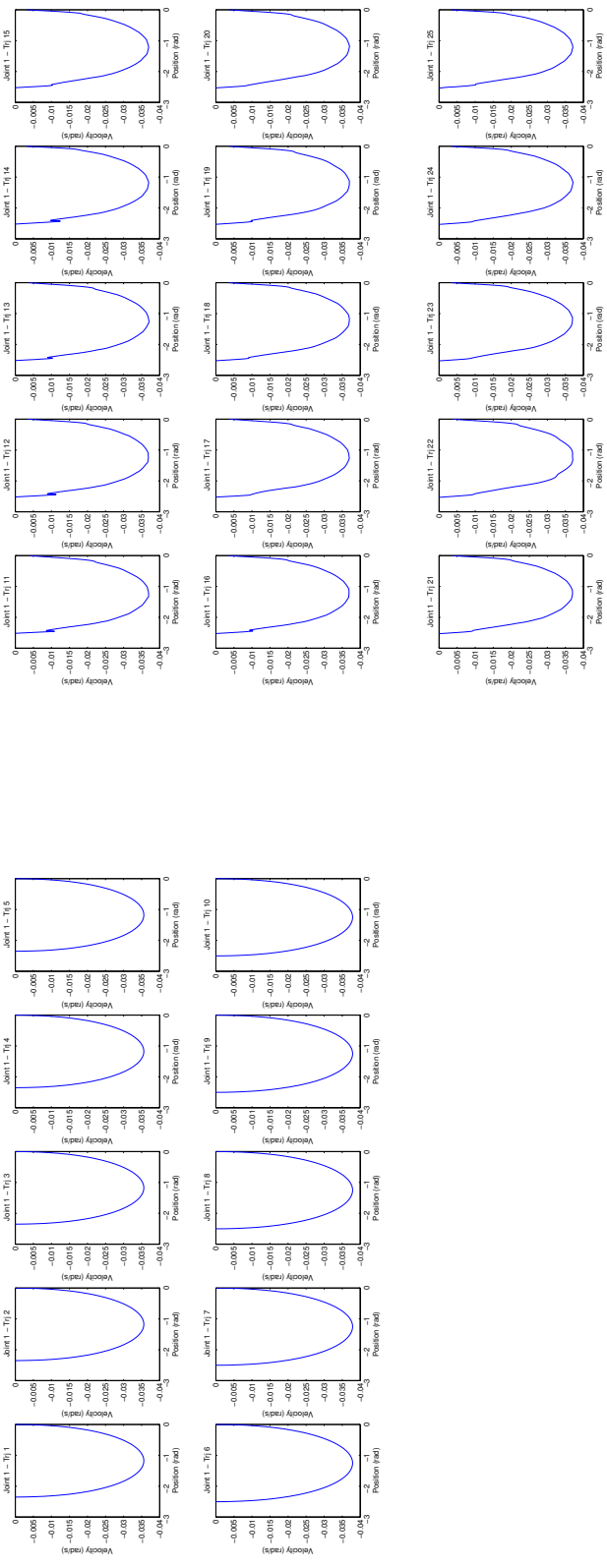


Figura 4.6: Confronto tra il dataset iniziale, a sinistra, e il dataset generato a destra, suddiviso in traiettorie singole nello spazio posizione-velocità.

4.2 Nao test

I test pratici sono stati realizzati tramite il Nao: il task scelto consiste nel fare canestro utilizzando i giunti del braccio. Inizialmente abbiamo tentato di registrare le traiettorie iniziali operando il robot a motori inattivi e catturando attraverso ROS i valori dei giunti (topic *joint_states*). Dalle traiettorie ottenute abbiamo notato che la frequenza media di pubblicazione dei giunti del Nao in ROS è pari a 16-17 Hz, molto più bassa rispetto ai 500 Hz del robot manipolare con cui sono stati eseguiti i test in [1]. Questo ci ha costretto a scegliere un dataset costituito da molte traiettorie sbagliate. Poiché ci siamo accorti che alcune delle traiettorie non potevano essere eseguite a causa della velocità troppo alta non sostenibile dal Nao, abbiamo optato per catturare le traiettorie in modo diverso. ROS consente di manipolare il robot tramite tecniche diverse[1]:

- si può stabilire la posizione finale dei giunti e la velocità con cui il robot deve muoversi (messaggi *JointAngleWithSpeed*);
- si può costruire una traiettoria delle posizioni dei giunti da muovere (messaggi *JointAngleTrajectory*).

Utilizziamo la prima tecnica per recuperare le traiettorie sbagliate: stabiliamo inizialmente la posizione iniziale e finale di movimento del giunto e variamo la velocità di esecuzione del movimento. Consideriamo le velocità che comportano traiettorie sbagliate, che quindi non fanno canestro ma si avvicinano a questo. Utilizziamo la seconda tecnica per riprodurre successivamente sia le traiettorie iniziali che quelle generate dal sistema di apprendimento. Prima le traiettorie vengono riprodotte in simulazione, in Gazebo, in modo da avere un primo riscontro grafico, e successivamente vengono eseguite dal robot reale (vedi figura 4.7).

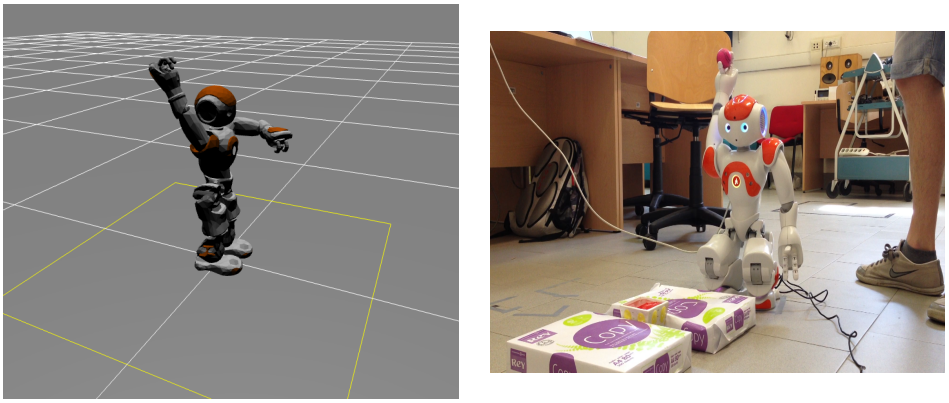


Figura 4.7: A sinistra il Nao simulato in Gazebo, a destra il robot reale.

4.2.1 Basket con un giunto

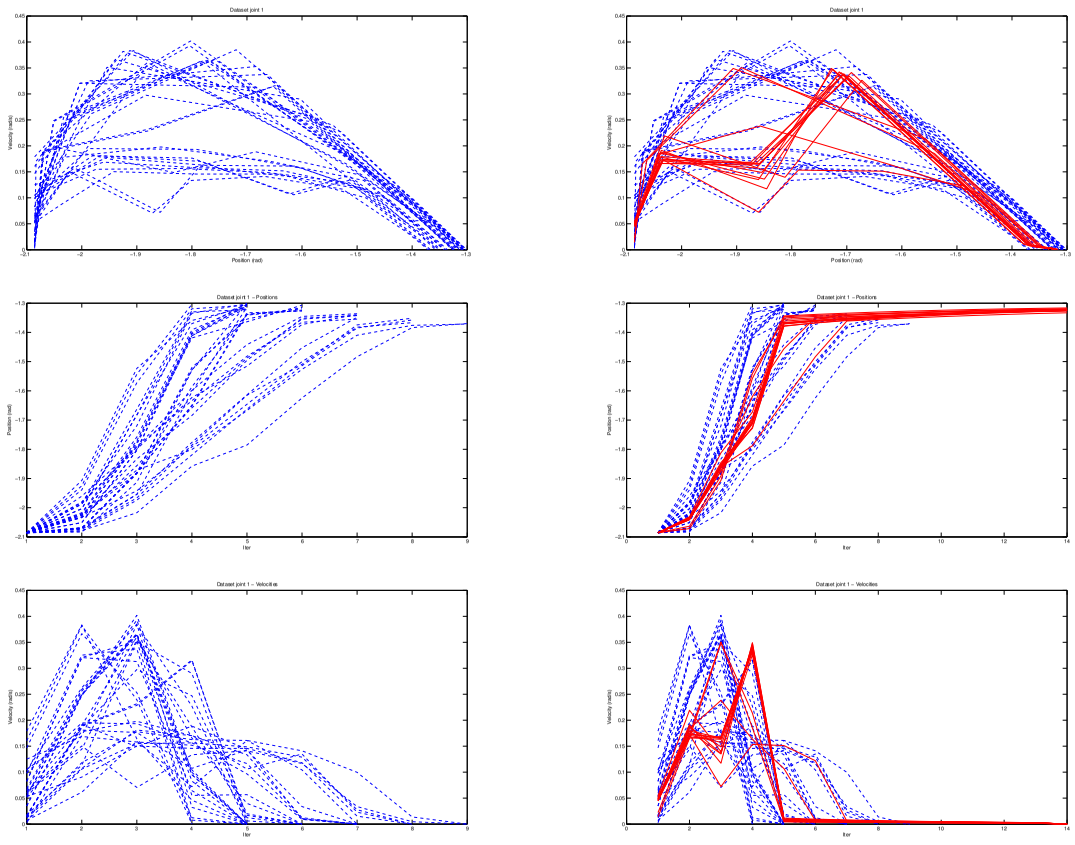
Vediamo i risultati nel caso di movimento di un solo giunto: il pitch della spalla destra del Nao. Posizioniamo il canestro ad una distanza di circa 30 cm dal robot e

[1] vedi `http://www.ros.org/wiki/nao_msgs`

utilizziamo il braccio simulando il lancio di una catapulta, partendo dalla posizione iniziale di -2.0857 rad e arrivando alla posizione finale di -1.30 rad. Consideriamo un dataset di 31 traiettorie non corrette, che fanno terminare la pallina prima e dopo il canestro oppure che lo toccano solamente. Applichiamo il sistema di apprendimento generando 15 nuove traiettorie, delle quali eseguiamo le prime 10 attraverso il Nao ad una frequenza pari a 16 Hz. Per verificare di ottenere il corretto comportamento effettuiamo lo stesso lancio più volte.

Risultati

In figura 4.8 abbiamo il confronto tra il dataset iniziale e il dataset finale contenente le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità, posizione-tempo e velocità-tempo. Possiamo notare come le traiettorie generate, in rosso, esplorano lo spazio tra le traiettorie iniziali, in blu tratteggiato. Nelle figure 4.9 e 4.10 abbiamo rispettivamente le traiettorie iniziali e le traiettorie generate suddivise singolarmente all'interno dello spazio posizione-velocità. Otteniamo generalmente andamenti di due tipi, similmente a quelli delle traiettorie di partenza: uno parabolico che si avvicina al caso ideale, mentre l'altro caratterizzato inizialmente da una crescita della velocità, seguita da una leggera decrescita, per poi crescere nuovamente, formando una sorta di M nello spazio posizione-velocità. Effettuando i lanci a canestro, otteniamo 4 volte canestro su 10 tentativi: in particolare il primo canestro avviene già al secondo tentativo.



TRAIETTORIE INIZIALI

Traj.	*Init. Vel.(*)*	*Datapoint*	*Init. ξ*	*Final ξ*	ξ_{max} *(rad/s)*	*Result(**)*
1	0.20	9	-2.0857	-1.37	0.20	P
2	0.21	9	-2.0857	-1.36	0.21	P
3	0.22	8	-2.0857	-1.36	0.22	P
4	0.23	8	-2.0857	-1.35	0.23	P
5	0.24	7	-2.0857	-1.35	0.24	P
6	0.25	7	-2.0857	-1.34	0.25	P
7	0.26	8	-2.0857	-1.35	0.26	P
8	0.27	7	-2.0857	-1.33	0.27	P
9	0.28	7	-2.0587	-1.33	0.28	P
10	0.29	7	-2.0857	-1.34	0.29	TP
11	0.30	7	-2.0857	-1.34	0.30	TP
12	0.31	6	-2.0857	-1.31	0.31	TP
13	0.32	6	-2.0857	-1.32	0.32	TP
14	0.33	6	-2.0857	-1.32	0.33	TD
15	0.34	6	-2.0857	-1.31	0.34	TD
16	0.50	6	-2.0857	-1.30	0.50	TD
17	0.51	5	-2.0857	-1.31	0.51	TD
18	0.52	5	-2.0857	-1.30	0.52	TD
19	0.53	6	-2.0587	-1.32	0.53	TD
20	0.54	5	-2.0857	-1.30	0.54	TD
21	0.55	5	-2.0857	-1.31	0.55	TD
22	0.56	4	-2.0857	-1.32	0.56	TD
23	0.57	5	-2.0857	-1.30	0.57	TD
24	0.58	4	-2.0857	-1.30	0.58	D
25	0.59	5	-2.0857	-1.30	0.59	TD
26	0.60	5	-2.0857	-1.30	0.60	TD
27	0.61	5	-2.0857	-1.30	0.61	TD
28	0.62	5	-2.0857	-1.32	0.62	D
29	0.63	5	-2.0857	-1.30	0.63	D
30	0.64	5	-2.0857	-1.31	0.64	TD
31	0.65	5	-2.0857	-1.31	0.64	D

TRAIETTORIE GENERATE

Traj.	*Datapoint*	*Init. ξ*	*Fin. ξ*	ξ_{max} *(rad/s)*	*Result(**)*
1	14	-2.0857	-1.30	0.38	D
2	**14**	**-2.0857**	**-1.30**	**0.22**	**C**
3	14	-2.0857	-1.31	0.17	TP
4	**14**	**-2.0857**	**-1.30**	**0.19**	**C**
5	14	-2.0857	-1.30	0.30	TP
6	14	-2.0857	-1.30	0.17	TP
7	14	-2.0857	-1.30	0.18	TP
8	14	-2.0857	-1.30	0.22	TP
9	**14**	**-2.0587**	**-1.30**	**0.20**	**C**
10	**14**	**-2.0857**	**-1.31**	**0.19**	**C**

Figura 4.8: Confronto tra il dataset iniziale, contenente le 31 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM. Nella tabella in alto a destra le informazioni relative alle traiettorie iniziali, mentre nella tabella sottostante le informazioni relative ai 10 tentativi generati ed eseguiti. (*)I valori di velocità iniziale corrispondono alla percentuale rispetto alla velocità massima supportata dal giunto. In arancio sono evidenziate le traiettorie corrette. (**)Valori possibili per la colonna risultato: P(prima), la pallina termina prima del canestro; TP(toccato prima), la pallina rimbalza sul bordo del canestro più vicino al robot; C(canestro); TD(toccato dopo), la pallina rimbalza sul bordo più distante; D(dopo), la pallina termina oltre il canestro; TL(toccato lateralmente), la pallina rimbalza su uno dei due bordi laterali.

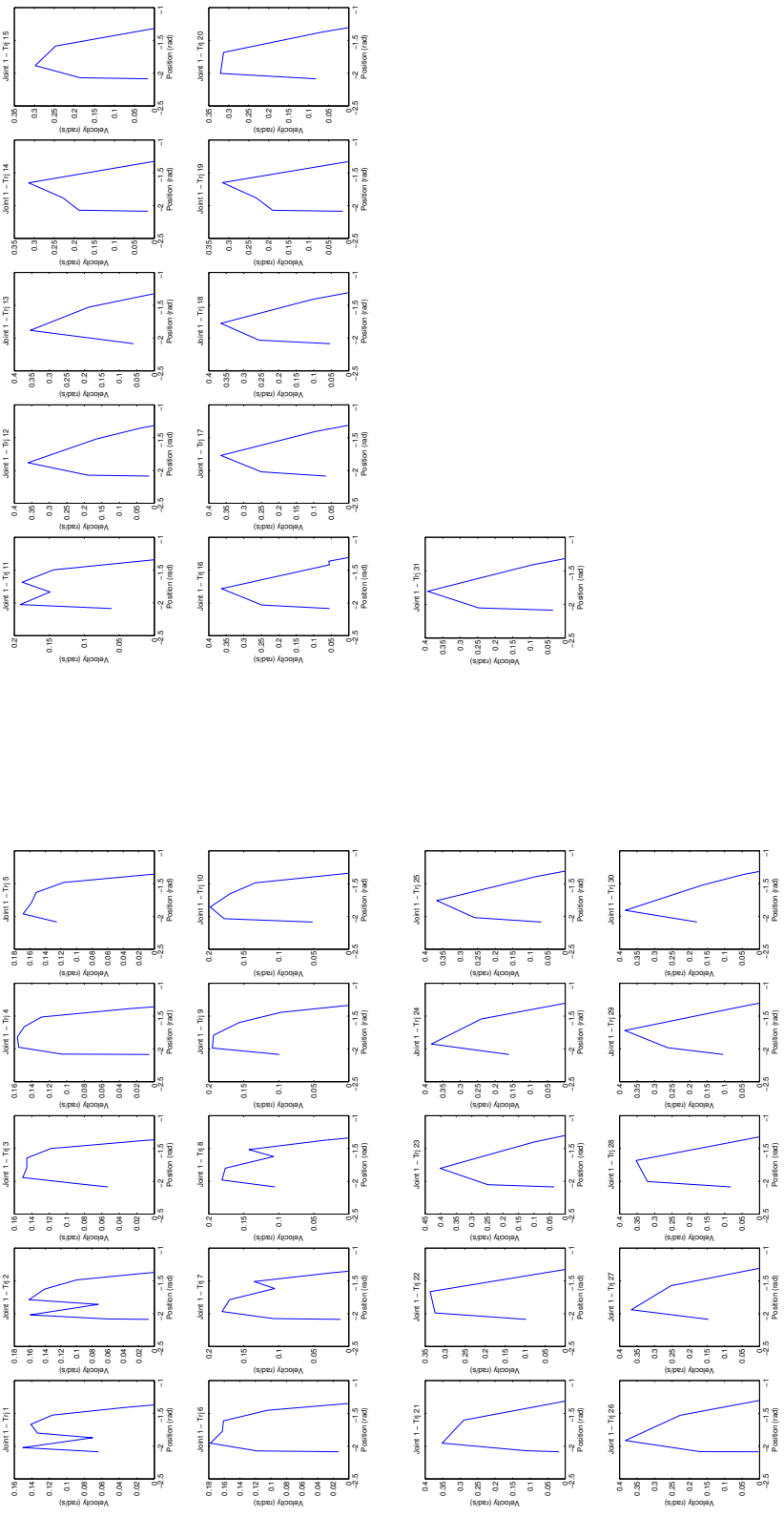


Figura 4.9: Le 31 traiettorie iniziali, raffigurate singolarmente nello spazio posizione-velocità.

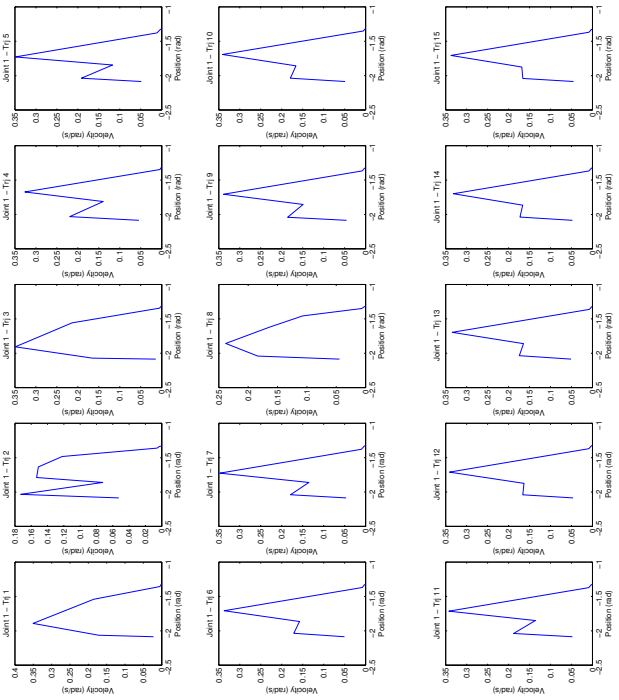


Figura 4.10: Le 15 traiettorie generate dal DMM, raffigurate singolarmente nello spazio posizione-velocità.

4.2.2 Basket con due giunti

Vediamo i risultati nel caso di movimento di due giunti: il pitch della spalla destra e il roll del gomito destro del Nao. Simuliamo dunque il movimento in modo più naturale: a partire dal braccio piegato all'indietro abbassiamo la spalla, dalla posizione iniziale di -2.0857 rad alla finale di -1.52 rad, mentre ruotiamo completamente il gomito, dalla posizione di 1.54 rad a 0.05 rad, in modo da tendere il braccio verso l'alto. Inizialmente creiamo un dataset di 30 traiettorie, che consideriamo non corrette: provando ad eseguirle, abbiamo riscontrato alcuni problemi legati al motore del giunto del gomito che non ci hanno permesso di completare correttamente i test. Dunque abbiamo solamente un riscontro grafico di quanto prodotto dal sistema.

Risultati

Nelle due figure 4.11 e 4.12, presentiamo il confronto tra le traiettorie iniziali e le traiettorie generate dai due modelli, all'interno dello spazio posizione-velocità, posizione-tempo e velocità-tempo, rispettivamente per il giunto della spalla e del gomito. Il comportamento esplorativo del DMM è mantenuto anche in questa situazione: le traiettorie generate, in rosso, spaziano all'interno delle spazio individuato dalle traiettorie di partenza, in blu. Anche in questo caso, dalle figure 4.13, 4.14 e 4.15, possiamo osservare che otteniamo per le traiettorie generate andamenti parabolici, soprattutto nel caso della spalla, e oscillatori nel caso del gomito, similmente a quanto accade nelle traiettorie di partenza.

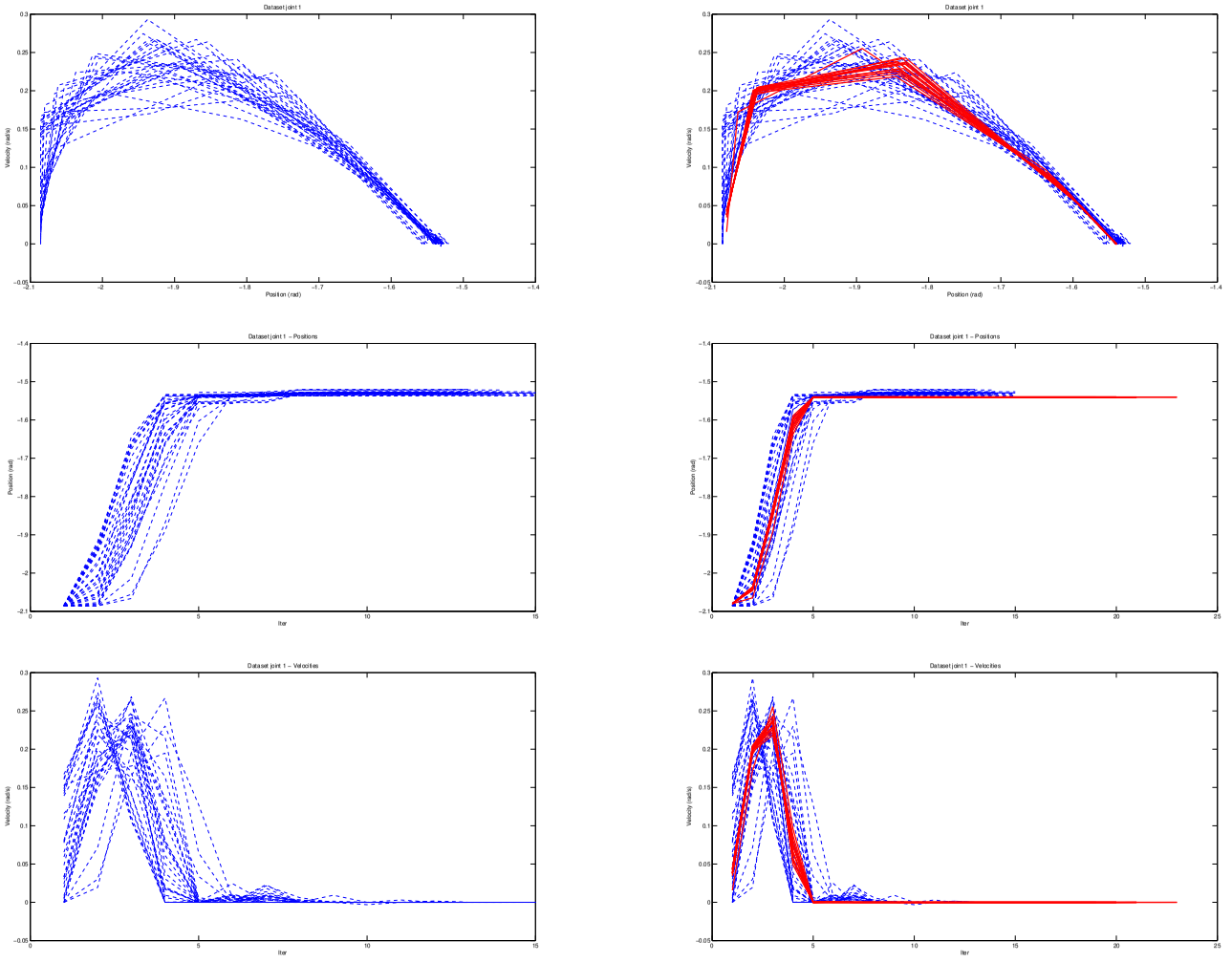


Figura 4.11: Giunto di pitch della spalla [-2.085;-1.52]. Confronto tra il dataset iniziale, contenente le 30 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM.

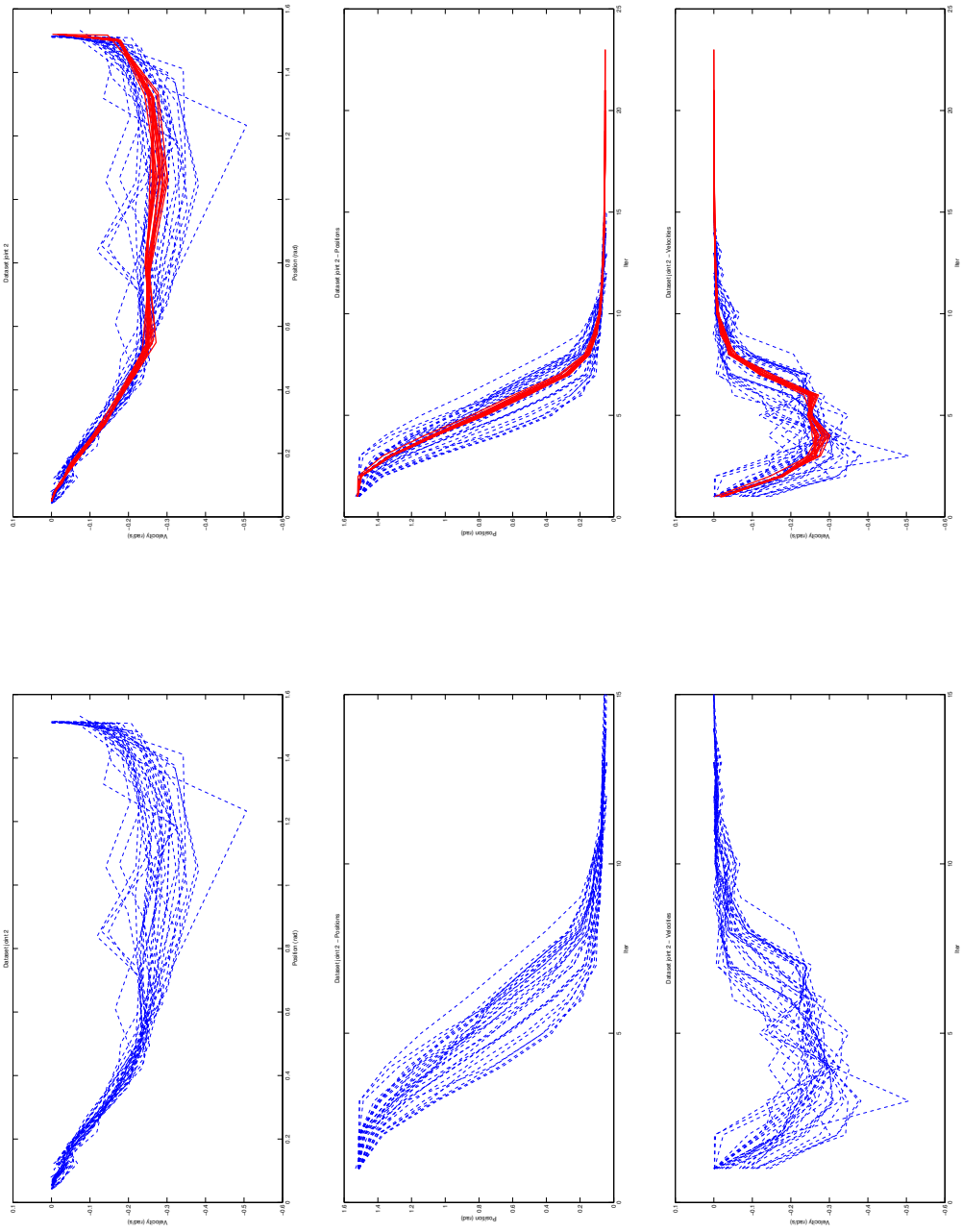


Figura 4.12: Giunto di roll del gomito [0.05;1.54]. Confronto tra il dataset iniziale, contenente le 30 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM.

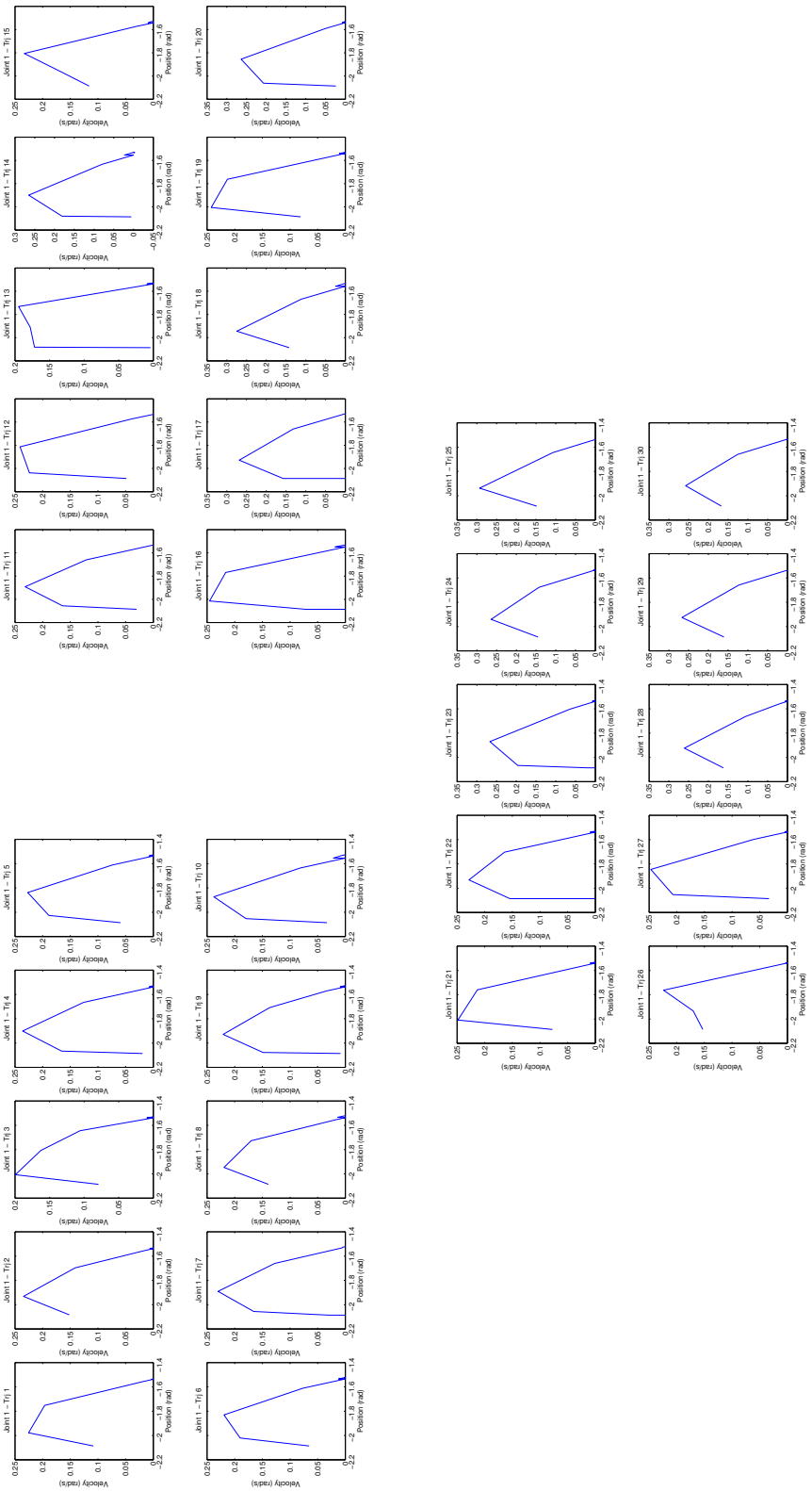


Figura 4.13: Le 30 traiettorie iniziali riguardanti il giunto della spalla, raffigurate singolarmente nello spazio posizione-velocità.

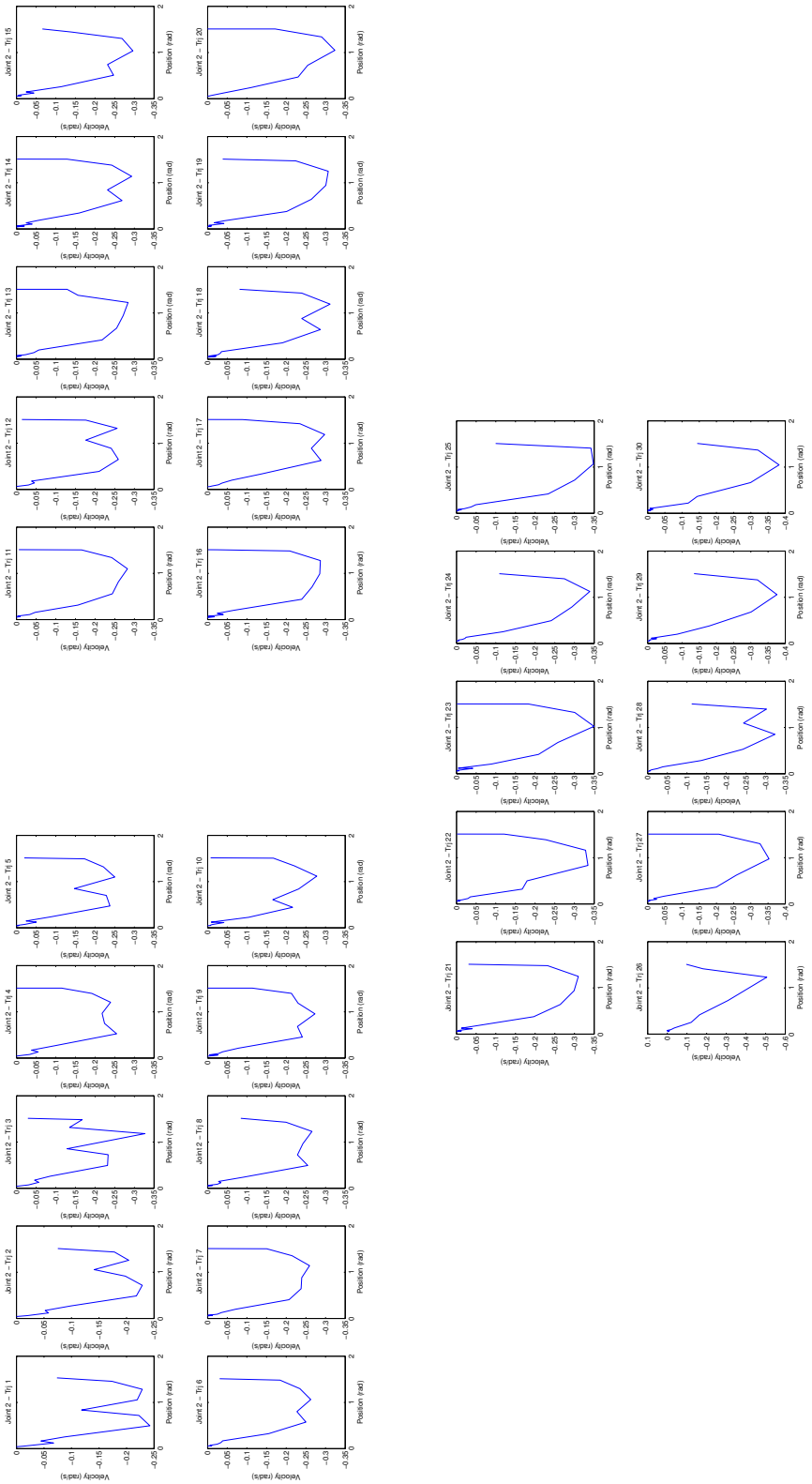


Figura 4.14: Le 30 traiettorie iniziali riguardanti il giunto del gomito, raffigurate singolarmente nello spazio posizione-velocità.

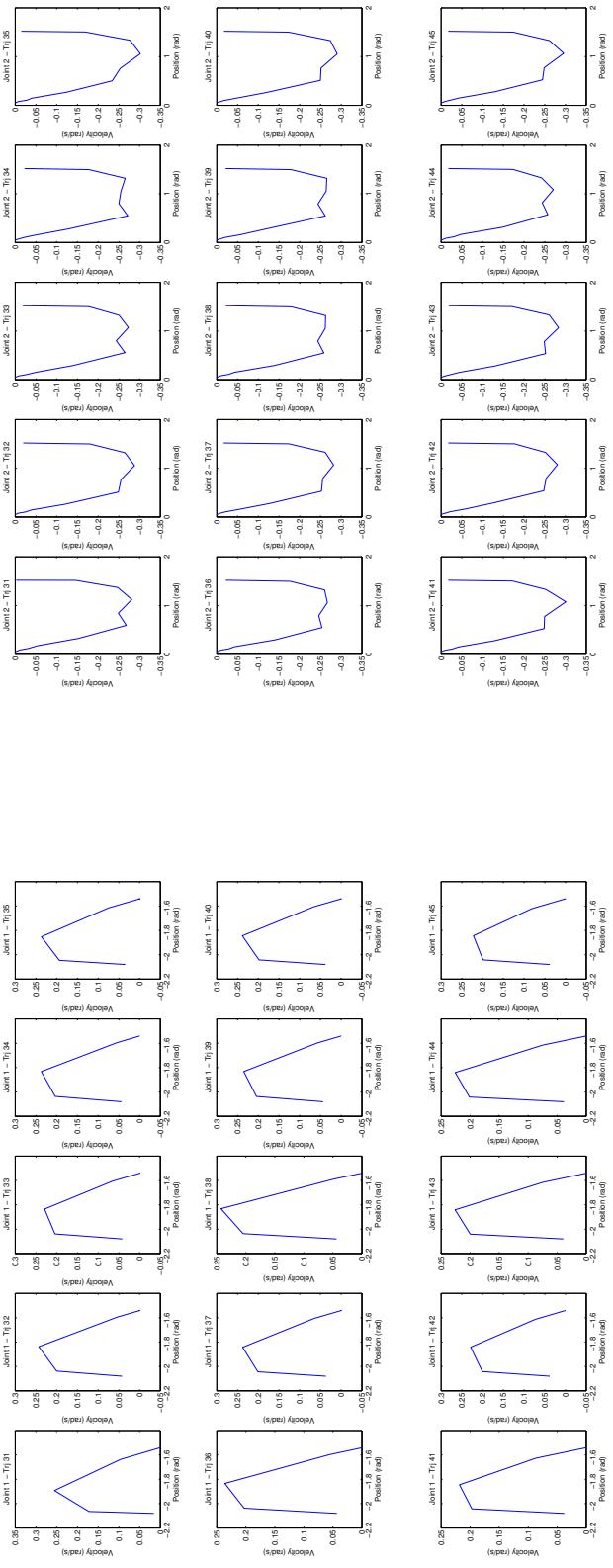


Figura 4.15: Le 15 traiettorie generate, suddivise rispettivamente per il giunto della spalla a sinistra e per il giunto del gomito a destra.

Capitolo 5

DMM e GMM nei movimenti umani

In questo capitolo analizziamo i due modelli Donut Mixture e Gaussian Mixture, nell'ambito dell'apprendimento da movimenti umani. Il nostro obiettivo consiste principalmente nell'istruire un robot a compiere una determinata azione, appresa a partire da dimostrazioni realizzate correttamente da diversi utenti umani. Sebbene il nostro lavoro appare come un approccio standard al RLfD, tuttavia si differenzia nei seguenti aspetti:

- l'apprendimento viene effettuato da utenti diversi;
- le dimostrazioni, anche se corrette, possono essere diverse tra loro;
- le dimostrazioni corrispondono a movimenti naturali.

Nel nostro approccio inizialmente si stabilisce il task da eseguire: ogni persona lo interpreta e lo svolge nel proprio stile, nella maniera corretta e più naturale possibile. Il movimento viene catturato e, successivamente, elaborato per poter essere svolto dal robot stesso: in particolare, dobbiamo tener conto del fatto che il robot possiede una capacità di movimento limitata rispetto all'utente stesso, in quanto dispone di un numero di gradi di libertà inferiore. Riconducendo i movimenti dalle persone al robot, la variabilità intrinseca delle dimostrazioni umane iniziali si traduce in una alterazione della correttezza del movimento eseguito dal robot. Se dal lato umano le dimostrazioni sono tutte corrette, dal lato robot questo aspetto non è mantenuto: situazione ideale per usufruire del modello esplorativo Donut Mixture in modo da ottenere la corretta esecuzione del movimento prestabilito.

Dal punto di vista pratico, abbiamo considerato come azione da eseguire, anche in questo caso, il lancio a canestro attraverso il movimento degli arti del braccio umano. Abbiamo scelto così un'azione semplice da svolgere, che permettesse di muovere solamente i giunti della parte superiore del robot, il Nao, e che potesse offrire un riscontro della correttezza o meno delle esecuzioni: consideriamo un lancio corretto nel caso di avvenuto canestro. Per le dimostrazioni iniziali abbiamo utilizzato le registrazioni di 12 persone, effettuate con il sensore RGB-D Kinect[1] e contenute

[1] http://www.xbox.com/it-IT/Kinect

all'interno di un dataset esistente, utilizzato per il riconoscimento di azioni (vedi figura 5.1).

Figura 5.1: registrazioni contenute nel dataset di riconoscimento di azioni.

Per ogni persona sono stati registrati 3 lanci, per un totale di 36 dimostrazioni iniziali. Dalle registrazioni abbiamo mappato i movimenti umani nel robot, estrapolando gli angoli dei giunti interessati ed elaborando i dati in modo da ottenere traiettorie iniziali pulite e che consentono di realizzare un movimento adeguato e valutabile. Infine abbiamo applicato ai dati ottenuti i due modelli DMM e GMR, valutandone il comportamento in termini di traiettorie generate e di convergenza verso la traiettoria corretta.

5.1 Mappatura dei giunti umani sul robot

Per creare le traiettorie iniziali con cui allenare i modelli di apprendimento, si devono estrapolare gli angoli dei giunti dai movimenti umani registrati con il sensore Kinect. Come passaggio preliminare dobbiamo mappare i giunti umani su quelli del Nao, in particolare quelli delle braccia, come le spalle, i gomiti e i polsi. Il sensore Kinect, sfruttando le librerie *OpenNI*[2], consente di catturare i movimenti e di tracciare lo scheletro umano, restituendo per ogni frame le posizioni dei giunti del corpo umano. Attraverso il pacchetto ROS *tf*[3] possiamo ottenere le trasformazioni tra le coordinate dei frame, in particolare le operazioni di orientazione e traslazione che consentono di passare da un frame all'altro, ad esempio tra il giunto della spalla e quello del gomito. Lo scopo è di mettersi in ascolto sulle trasformazioni, in modo da conoscere le posizioni dei giunti umani, e ottenere, attraverso operazioni matematiche di cinematica inversa, i valori degli angoli tra i giunti, che vengono successivamente mappati nei valori corrispondenti per il robot. Per facilitare i calcoli delle posizioni, in particolare dei vettori tra i giunti, si possono considerare le posizioni rispetto ad un sistema di riferimento fisso, come ad esempio quello associato ai frame della camera del sensore Kinect oppure al torso.

5.1.1 Angoli di yaw dei polsi

L'angolo di yaw del polso corrisponde all'angolo di rotazione della mano attorno ad un asse parallelo all'avambraccio. Poiché il sistema di riferimento del polso non

[2] `http://www.openni.org/`
[3] `http://www.ros.org/wiki/tf`

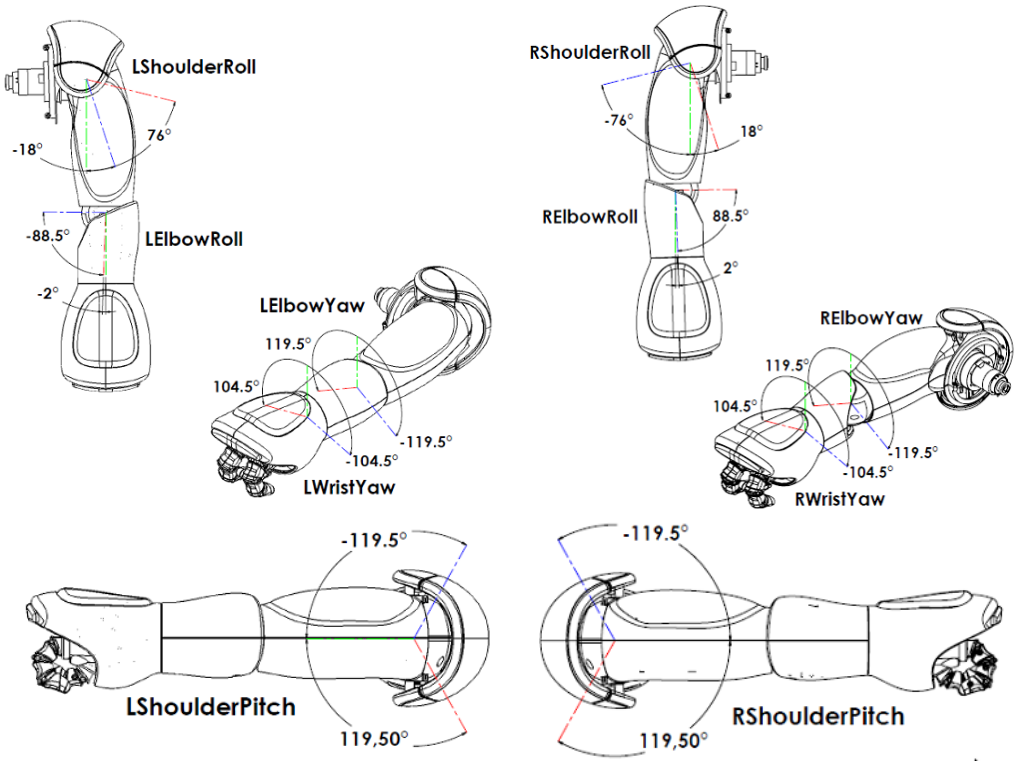


Figura 5.2: giunti degli arti superiori del Nao.

LEFT ARM	
Joint Name	*Range (rad)*
LShoulderRoll	[-0.3142;1.3265]
LShoulderPitch	[-2.0857;2.0857]
LElbowRoll	[-1.5446;-0.0349]
LElbowYaw	[-2.0857;2.0857]
LWristYaw	[-1.8238;1.8238]

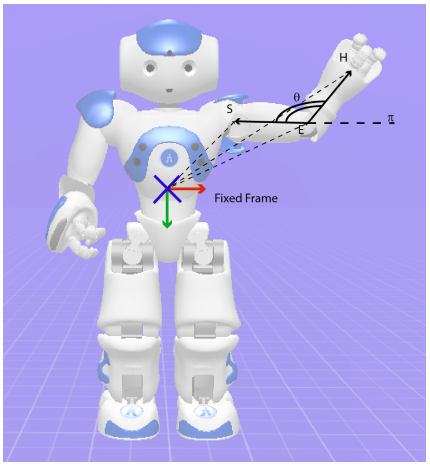
RIGHT ARM	
Joint Name	*Range (rad)*
RShoulderRoll	[-1.3265;0.3142]
RShoulderPitch	[-2.0857;2.0857]
RElbowRoll	[0.0349;1.5446]
RElbowYaw	[-2.0857;2.0857]
RWristYaw	[-1.8238;1.8238]

Figura 5.3: valori possibili per i giunti del braccio sinistro e destro del Nao.

varia nel tempo, non è possibile valutarne l'angolo di rotazione e quindi mappare il movimento.

5.1.2 Angoli di roll dei gomiti

L'angolo di roll del gomito corrisponde all'angolo di apertura o chiusura dell'avambraccio rispetto alla parte alta del braccio: per il calcolo si considera l'angolo tra i due vettori centrati sul gomito, il vettore gomito-spalla e il vettore gomito-mano, ottenuto dal loro prodotto scalare.



$\overrightarrow{ES}$, vettore gomito-spalla
$\overrightarrow{EH}$, vettore gomito-mano

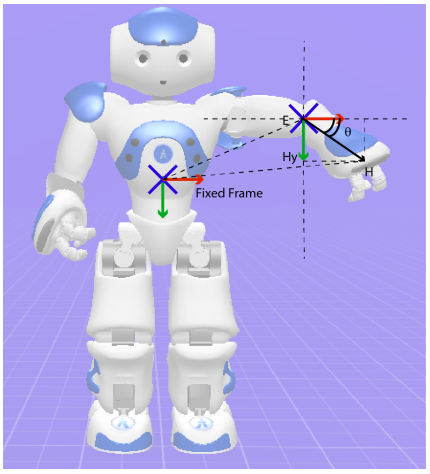
$$< \overrightarrow{ES}, \overrightarrow{EH} >= |\overrightarrow{ES}||\overrightarrow{EH}| \cos\theta$$
$$\theta = \arccos\left(\frac{<\overrightarrow{ES}, \overrightarrow{EH}>}{|\overrightarrow{ES}||\overrightarrow{EH}|}\right)$$

Angolo di roll:

$$\alpha_{roll} = \begin{cases} -\pi + \theta & \text{se gomito sinistro} \\ \pi - \theta & \text{se gomito destro} \end{cases} \tag{5.1}$$

5.1.3 Angoli di yaw dei gomiti

L'angolo di yaw del gomito corrisponde all'angolo di rotazione dell'avambraccio centrato sul gomito, rispetto alla parte alta del braccio. Per il calcolo si considera il sistema di riferimento fisso traslato sul gomito e si valuta l'angolo formato dal vettore gomito-mano rispetto all'asse verticale passante per il gomito.



$\overrightarrow{EH}$, vettore gomito-mano
$\overrightarrow{EH_y}$, proiezione del vettore sull'asse y

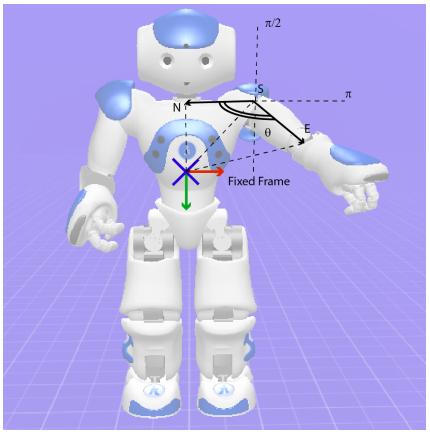
$$\theta = \arcsin(\overrightarrow{EH_y})$$

Angolo di yaw:

$$\gamma_{yaw} = \begin{cases} \theta & \text{se gomito sinistro} \\ -\theta & \text{se gomito destro} \end{cases} \tag{5.2}$$

5.1.4 Angoli di roll delle spalle

L'angolo di roll della spalla corrisponde all'angolo di apertura o chiusura laterale del braccio rispetto al busto. Per il calcolo si considera l'angolo ottenuto dal prodotto scalare tra i due vettori centrati sulla spalla, il vettore spalla-collo e il vettore spalla-gomito.



$\overrightarrow{SN}$, vettore spalla-collo
$\overrightarrow{SE}$, vettore spalla-gomito

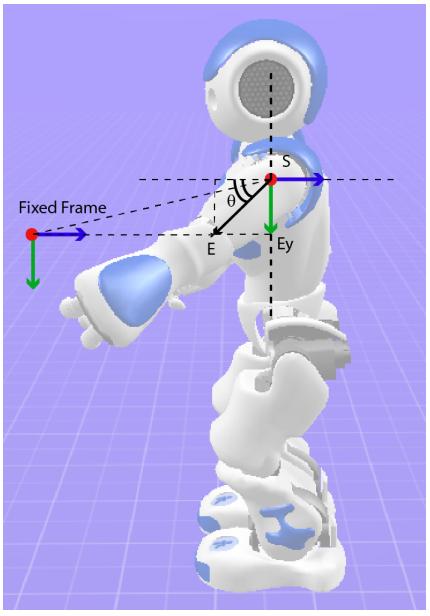
$$< \overrightarrow{SN}, \overrightarrow{SE} >= |\overrightarrow{SN}||\overrightarrow{SE}| \cos\theta$$
$$\theta = \arccos\left(\frac{<\overrightarrow{SN}, \overrightarrow{SE}>}{|\overrightarrow{SN}||\overrightarrow{SE}|}\right)$$

Angolo di roll:

$$\alpha_{roll} = \begin{cases} \theta - \frac{\pi}{2} & \text{se spalla sinistra} \\ \frac{\pi}{2} - \theta & \text{se spalla destra} \end{cases} \tag{5.3}$$

5.1.5 Angoli di pitch delle spalle

L'angolo di pitch della spalla corrisponde all'angolo di alzata o abbassamento frontale del braccio rispetto al busto. Per il calcolo si considera il sistema di riferimento fisso traslato sulla spalla e si valuta l'angolo formato dal vettore spalla-gomito, proiettato nel piano laterale, con l'asse verticale passante per la spalla.



$\overrightarrow{SE}$, vettore spalla-gomito
$\overrightarrow{SE_y}$, proiezione del vettore sull'asse y
$\overrightarrow{SE_{yz}}$, proiezione del vettore sul piano yz

$$\theta = \arcsin\left(\frac{\overrightarrow{SE_y}}{|\overrightarrow{SE_{yz}}|}\right)$$

Angolo di pitch:

$$\beta_{pitch} = \begin{cases} \theta & \text{se } z < 0 \\ \pi - \theta & \text{se } z > 0, y > 0 \\ -\pi - \theta & \text{se } z > 0, y < 0 \end{cases} \tag{5.4}$$

5.1.6 Proporzioni

Le formule precedenti, 5.3 e 5.4, di roll e pitch della spalla, funzionano correttamente nel caso in cui l'utente esegua movimenti della spalla che rientrano nei movimenti possibili per il robot. Ad esempio se si alza il braccio lateralmente (roll puro), il movimento viene eseguito in modo corretto fino a che il braccio non è allo stesso livello della spalla. Nel caso in cui il braccio dovesse oltrepassare la spalla, nel movimento del robot si produrrebbe uno scatto del braccio dovuto ad una rotazione improvvisa di 180 gradi, causata dal passaggio dell'angolo di pitch della spalla da -90 gradi a 90 gradi. Quindi in questo caso il movimento umano di alzata laterale non può essere mappato correttamente: per ovviare a questo problema si sostituiscono le formule precedenti con le formule 5.5 e 5.6, basate sulle proporzioni relative alle proiezioni del vettore spalla-gomito rispetto agli assi del sistema di riferimento fisso, traslato sulla spalla.

$\overrightarrow{SE}$, vettore spalla-gomito
$\overrightarrow{SE_x}$, proiezione del vettore sull'asse x
$\overrightarrow{SE_{xy}}$, proiezione del vettore sul piano xy

Angolo di roll della spalla:

$$\alpha_{roll} = \frac{\overrightarrow{SE_x}}{|\overrightarrow{SE_{xy}}|} * 1.30 \qquad (5.5)$$

$\overrightarrow{SE}$, vettore spalla-gomito
$\overrightarrow{SE_y}$, proiezione del vettore sull'asse y
$\overrightarrow{SE_{yz}}$, proiezione del vettore sul piano yz

Angolo di pitch della spalla:

$$\beta_{pitch} = \frac{\overrightarrow{SE_y}}{|\overrightarrow{SE_{yz}}|} * \frac{\pi}{2} \qquad (5.6)$$

Tuttavia per il nostro obiettivo utilizziamo le formule 5.3 e 5.4.

5.2 Elaborazione delle traiettorie

Una volta mappati gli angoli tra i giunti umani e del robot, si analizzano ed elaborano i valori ottenuti in modo da renderli eseguibili dal robot e soprattutto valutabili, in termini di correttezza o meno del movimento. In questa sezione viene descritta la procedura utilizzata per ottenere le traiettorie su cui applicare i modelli di apprendimento.

Inizialmente si analizzano i dati, assieme ai video dei lanci registrati, in modo da capire come sono stati effettuati i movimenti. Dai video si può notare come il movimento di lancio per fare canestro coinvolge sostanzialmente tutti i giunti del braccio, vedi figura 5.4.

Per prima cosa l'utente porta i giunti nella posizione da cui lanciare: il braccio viene alzato frontalmente (pitch spalla) e lateralmente (roll spalla) fino ad altezza della spalla, mentre l'avambraccio viene chiuso (roll gomito) e portato verso l'alto (yaw gomito). Successivamente viene effettuato il lancio: il braccio si alza leggermente (pitch spalla) e contemporaneamente l'avambraccio viene aperto (roll gomito), mentre il polso della mano viene piegato in avanti per lanciare la palla. Questa tipologia di movimento è la più comune all'interno del dataset iniziale, anche se alcuni lanci vengono effettuati diversamente: ad esempio nel lancio la posizione di partenza del braccio è superiore a quella della spalla e il lancio consiste solamente in

Figura 5.4: esempi di lanci effettuati correttamente da 5 persone diverse. Per ogni persona è visualizzata la posizione iniziale (immagine a sinistra), la posizione di partenza del lancio (immagine centrale) e la posizione finale del lancio (immagine a destra).

un movimento a catapulta del braccio verso il basso. Dunque la prima operazione sui dati consiste nello scartare manualmente questi movimenti.

A questo punto si valutano le modalità di esecuzione dell'azione tramite il robot, considerando il numero di giunti da muovere e soprattutto che il robot gode di meno gradi di libertà dell'utente umano. In effetti, come si può comprendere dalla descrizione precedente, il lancio non dipende solamente dallo spostamento compiuto dal gomito, ma anche dall'inclinazione del polso. Il robot a differenza dell'utente non possiede questo tipo di movimento, ma può muovere solamente il polso con una rotazione lungo l'asse parallelo all'avambraccio.

Abbiamo deciso di riprodurre il comportamento attraverso il robot muovendo in un caso un solo giunto, la spalla (pitch), mentre nell'altro due giunti, spalla (pitch) e gomito (roll): nel primo caso, lo spostamento compiuto dal gomito umano viene mappato nella spalla del robot, essendo questo il movimento predominante del task; nel secondo, i movimenti umani di spalla e gomito vengono mappati nei rispettivi giunti del robot. Un problema comune alle due configurazioni riguarda il movimento del gomito: i valori degli angoli di apertura o chiusura dell'avambraccio possono oltrepassare i valori limite stabiliti per il movimento dello stesso giunto nel robot. Ad esempio, vedi la seconda e quinta persona di figura 5.4, l'utente umano è in grado di piegare il gomito fino a portare l'avambraccio quasi a toccare la parte alta del braccio, mentre il robot arresta la chiusura fin tanto che l'avambraccio non forma un angolo retto con il braccio. Un ulteriore problema invece riguarda il rumore introdotto nei dati dal sensore Kinect e dallo skeleton tracker: ad esempio i dati contengono numerosi valori duplicati oppure nello spostamento di un giunto, con andamento di sola crescita o decrescita, i valori restituiti tendono ad oscillare, in disaccordo con il movimento stesso.

Dunque si elaborano i dati iniziali nel seguente modo:

1. si selezionano i valori dei giunti interessati al movimento;
2. si eliminano i valori duplicati;
3. si estrapola automaticamente il solo movimento di lancio, stabilendo per ogni giunto la posizione iniziale e finale su cui filtrare i valori;
4. si controlla per ogni giunto che il movimento abbia un andamento lineare, crescente o decrescente, andando ad eliminare o correggere manualmente eventuali valori oscillatori.

Per ovviare al problema relativo al superamento delle soglie, del minor numero di giunti e per realizzare movimenti valutabili, sono state applicate le seguenti operazioni:

- per ogni giunto si stabilisce un intervallo di valori su cui proporzionare gli angoli;
- si traslano i valori rispetto a una posizione di partenza.

In questo modo si ottengono traiettorie pulite, che non oscillano, che cercano di utilizzare il maggior numero di dati a disposizione e che iniziano e terminano nelle medesime posizioni.

In particolare, nel caso di utilizzo di un solo giunto del robot, si realizza il movimento della sola spalla elaborando i valori filtrati del gomito:

- si traslano i valori rispetto alla posizione di partenza di -2.05 rad;
- si proporzionano i nuovi valori all'interno dell'intervallo [-2.05;-1.35] rad.

Otteniamo dunque lo spostamento del braccio dall'alto verso il basso, similmente al comportamento della catapulta.

Per il movimento a due giunti, pitch della spalla e roll del gomito, si opera così:

- si proporzionano i valori del gomito umano rispetto all'intervallo completo dei possibili valori di roll del gomito del robot, [-1.54; -0.05] rad nel caso del gomito sinistro e [1.54;0.05] rad nel gomito destro;
- si traslano i valori della spalla umana rispetto alla posizione di -0.90 rad;
- si proporzionano i valori della spalla umana nell'intervallo [-0.90;-1.20] rad.

In questa situazione otteniamo un movimento che si avvicina al movimento naturale dell'utente: la spalla tende ad alzarsi leggermente, mentre il gomito apre l'avambraccio. In entrambi i casi gli intervalli dei valori sono stati scelti per consentire al robot di eseguire i lanci valutandone la correttezza.

5.3 Applicazione dei modelli

Ottenute le traiettorie, queste vengono inizialmente eseguite per verificare quali sono corrette, caso di lancio a canestro, e quali non lo sono. Se i lanci effettuati inizialmente dagli utenti sono corretti, non è detto che questa caratteristica si mantenga durante l'esecuzione attraverso il robot delle nuove traiettorie, ottenute dall'elaborazione dei dati iniziali. Nel caso avessimo ancora una correttezza completa, potremo ricondurci al caso di apprendimento da dimostrazioni corrette e quindi potremo applicare sui dati la Gaussian Mixture Regression in modo da riprodurre il task correttamente. Contrariamente, se avessimo solo traiettorie incorrette, potremo ricondurci al caso di RLfF e quindi sfruttare il modello Donut Mixture per esplorare i dati e ottenere la traiettoria corretta. Dunque sulla base delle nuove traiettorie, lo scopo è di istruire il robot ad eseguire il comportamento correttamente, applicando i due sistemi di apprendimento e analizzando il loro comportamento.

Prima di descrivere i risultati relativi alla situazione ad uno e a due giunti, illustriamo brevemente il funzionamento della Gaussian Mixture Regression (GMR).

5.3.1 Gaussian Mixture Regression

A partire dalle informazioni probabilistiche del Gaussian Mixture Model associato, la Gaussian Mixture Regression è lo strumento per la ricostruzione di una forma generale per i segnali in ingresso: la regressione viene utilizzata per ottenere traiettorie generalizzate su un insieme di traiettorie osservate[5]. La GMR basa il

proprio funzionamento principalmente sul teorema del condizionamento gaussiano e sulla proprietà di combinazione lineare delle funzioni gaussiane.

Da un insieme di osservazioni $\epsilon = \{\epsilon_t, \epsilon_s\}$, con ϵ_s vettore spaziale o delle posizioni D-dimensionale al tempo ϵ_t, la distribuzione di probabilità congiunta $p(\epsilon_t, \epsilon_s)$ viene prima modellata da un GMM Θ. La regressione, sulla base dei parametri (medie, covarianze, priors) delle K componenti gaussiane di Θ, ottiene la versione generalizzata delle traiettorie stimando l'aspettazione condizionata $E[p(\epsilon_s|\epsilon_t)]$ di ϵ_s dato ϵ_t. Calcolando l'aspettazione condizionata ad ogni passo temporale si estrae l'intera traiettoria.

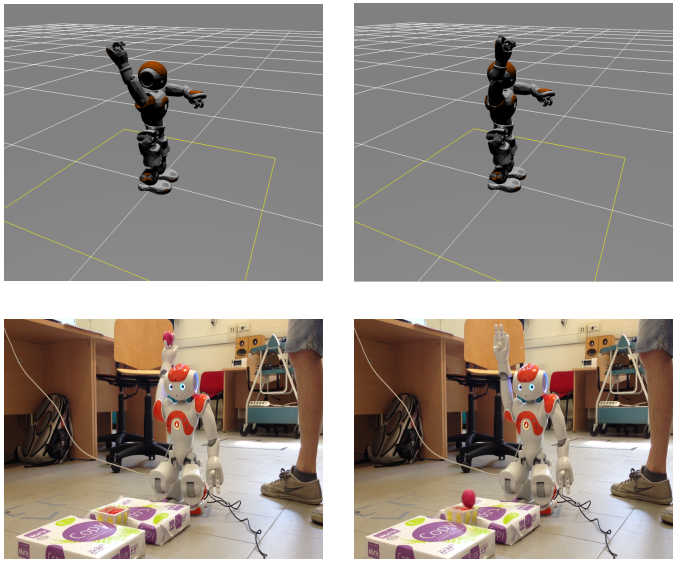
Per una descrizione approfondita sia di GMM che di GMR rimando alla visione di quanto proposto in [5] e della tesi di laurea di Nicola Chessa [19].

5.3.2 Un giunto

In questa situazione, riduciamo il movimento umano, che complessivamente comporta l'utilizzo di quasi tutti i giunti del braccio, allo spostamento di un solo giunto del Nao, come descritto in 5.2. Tra i giunti umani mossi consideriamo quello il cui spostamento caratterizza l'azione: andiamo quindi a mappare il giunto di roll del gomito sinistro dell'utente sul giunto di pitch della spalla destra del Nao. La scelta di utilizzare la spalla e l'arto opposto del robot rispetto a quello umano è dovuta fondamentalmente a guasti meccanici legati ad alcuni giunti del Nao. Elaborando le 36 registrazioni iniziali, si ottiene un dataset di 34 traiettorie, con andamenti diversi tra loro ma stesse posizioni iniziali, -2.05 rad, e posizioni finali, -1.35 rad. In questo modo generiamo un movimento del braccio a catapulta, dall'alto verso il basso, vedi immagini di figura 5.5. Disponiamo il canestro ad una distanza di 40 cm dal Nao ed eseguiamo le traiettorie iniziali, con una frequenza di campionamento pari a 16 Hz. Dai risultati ottenuti e riportati nella tabella di figura 5.5, possiamo notare come 11 lanci su 34 vanno a canestro, mentre i restanti 23 corrispondono a traiettorie non corrette. Ci troviamo in una situazione intermedia per l'apprendimento, poiché disponiamo sia di traiettorie corrette che incorrette. Andiamo dunque a valutare il comportamento dei due modelli in tre circostanze diverse:

1. su tutte le traiettorie;
2. solamente sulle traiettorie non corrette, riconducendoci al puro RLfF;
3. sulle sole traiettorie corrette, riconducendoci al puro RLfD.

Per ogni situazione, generiamo 15 nuove traiettorie attraverso il DMM e le confrontiamo con la traiettoria ottenuta dall'utilizzo sugli stessi dati del GMR. Ricordiamo che il DMM viene applicato su un dataset contenente coppie di posizione e velocità, mentre il GMR costruisce la propria traiettoria a partire da valori di tempo e posizione, senza considerare i valori di velocità, ottenuti successivamente come differenza tra posizioni successive e attuali.



TRAIETTORIE INIZIALI

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	5	0.204	TD
2	7	0.191	C
3	6	0.193	C
4	8	0.16	C
5	10	0.141	C
6	11	0.266	TP
7	6	0.239	TD
8	9	0.202	TD
9	9	0.152	P
10	12	0.198	C
11	6	0.24	TD
12	5	0.273	TD
13	9	0.239	TD
14	7	0.154	TP
15	10	0.152	P
16	7	0.171	P
17	14	0.128	P
18	9	0.181	TD
19	10	0.188	C
20	10	0.165	C
21	7	0.188	C
22	5	0.231	TD
23	9	0.183	TD
24	7	0.204	TD
25	8	0.206	C
26	9	0.188	TD
27	8	0.197	TD
28	9	0.167	C
29	13	0.121	P
30	13	0.163	C
31	7	0.264	TD
32	8	0.28	D
33	11	0.241	TD
34	15	0.112	P

Figura 5.5: A sinistra le immagini relative alla simulazione in Gazebo e alle esecuzioni sul robot reale del lancio con un giunto. A destra la tabella contenente le informazioni, numero di datapoint, velocità massima raggiunta e risultato, riguardo le esecuzioni delle 34 traiettorie iniziali attraverso il robot reale. (*)Valori possibili per la colonna risultato: P(prima), la pallina termina prima del canestro; TP(toccato prima), la pallina rimbalza sul bordo del canestro più vicino al robot; C(canestro); TD(toccato dopo), la pallina rimbalza sul bordo più distante; D(dopo), la pallina termina oltre il canestro; TL(toccato lateralmente), la pallina rimbalza su uno dei due bordi laterali.

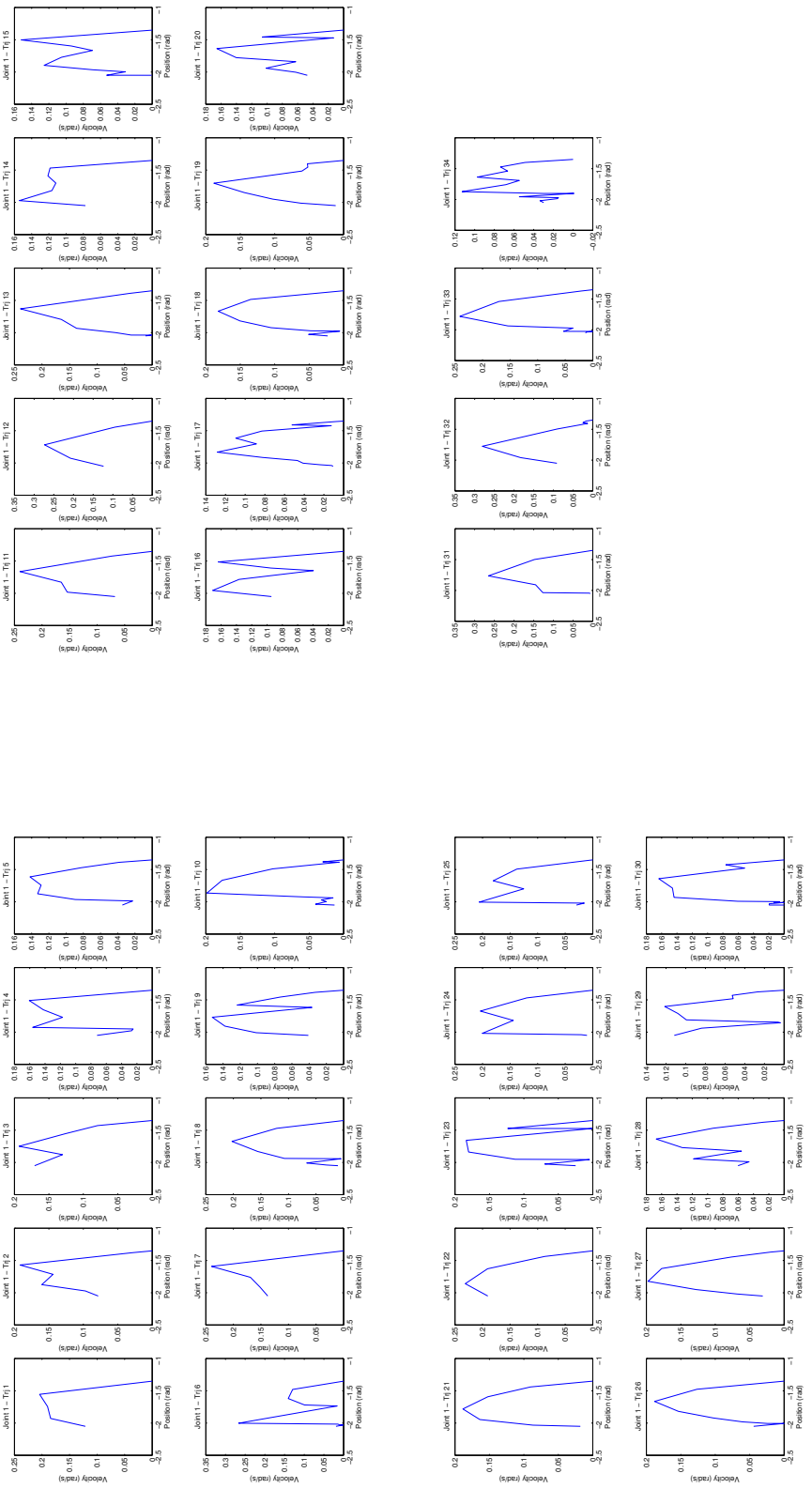


Figura 5.6: Dataset iniziale, contenente le 34 traiettorie, rappresentate nello spazio posizione-velocità.

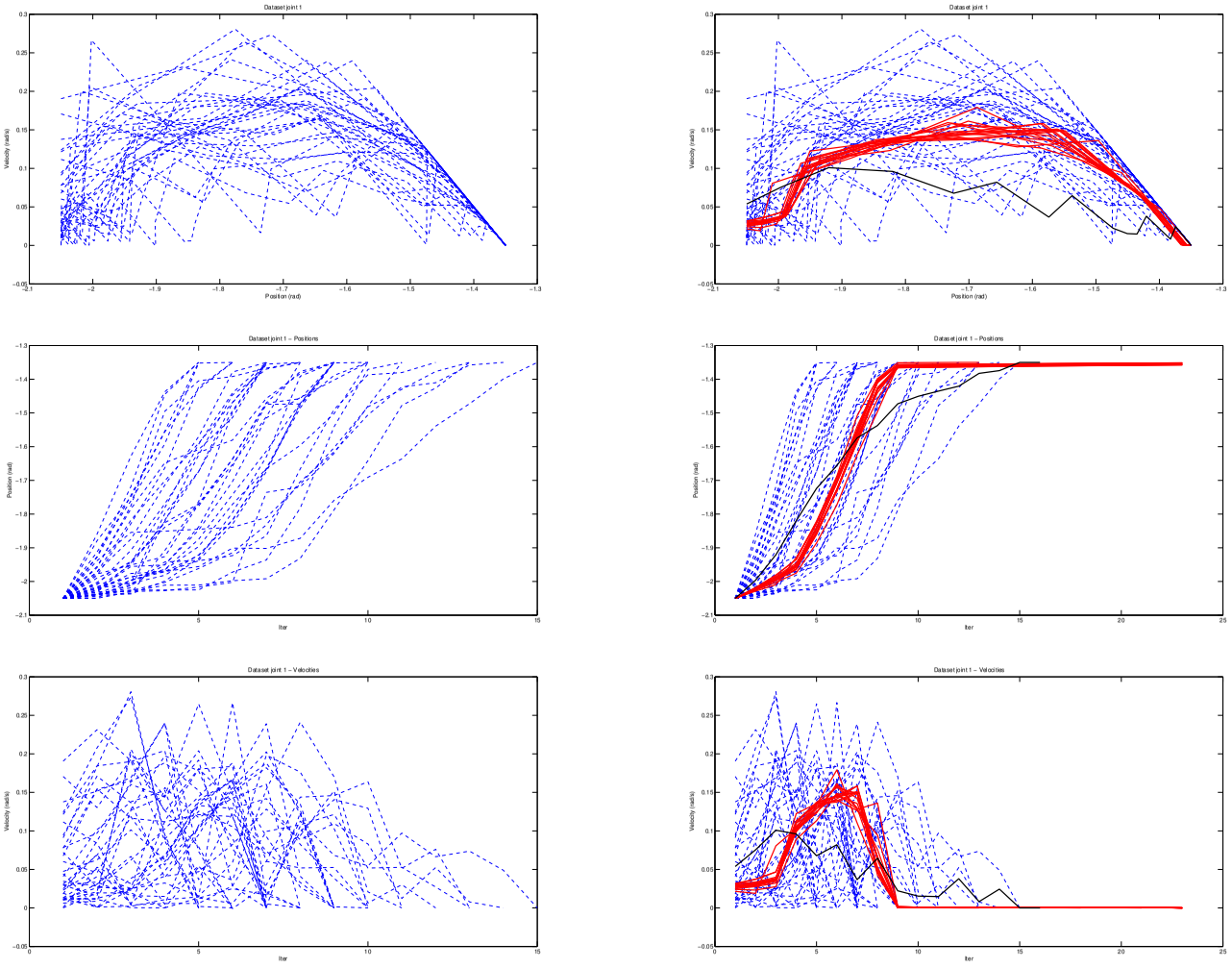
Caso 1: tutte le traiettorie

Come si può notare in figura 5.6, disponiamo di traiettorie iniziali caratterizzate da pochi datapoint, in media 8 per traiettoria, e con andamenti diversi tra loro: poche sono le traiettorie con andamento posizione-velocità puramente parabolico, mentre predomina un andamento oscillatorio dovuto al variare della velocità. Il numero limitato di punti per traiettoria è dovuto alla velocità di esecuzione del movimento, tipicamente compiuto in uno o due secondi. Eseguendo le traiettorie, otteniamo 11 lanci dentro il canestro, mentre i restanti 23 non corretti si suddividono in 8 lanci che terminano prima del canestro (2 di questi lo toccano) e 15 che terminano oltre il canestro (14 di questi lo toccano).

In figura 5.7 sono rappresentati il dataset iniziale e le traiettorie generate dall'applicazione dei due modelli: in particolare, in rosso abbiamo i risultati del DMM, mentre in nero quello del GMR. Nelle prime due figure, in alto, i dati sono raffigurati all'interno dello spazio posizione-velocità, che andiamo a scomporre nella componente di posizione, nelle due figure intermedie, e nella componente di velocità, nelle figure in basso.

Possiamo dunque notare come il modello DMM consente di ottenere traiettorie che spaziano all'interno dello spazio posizione-velocità determinato dalle traiettorie iniziali. Il GMR consente invece di ottenere una traiettoria media per quanto riguarda la posizione, caratterizzata da una pendenza minore rispetto alle traiettorie di posizione ottenute dal DMM, causa dell'abbassamento e dell'oscillazione dei valori di velocità. Contrariamente, come possiamo osservare in figura 5.8, le traiettorie generate dal DMM possiedono un andamento che si avvicina all'andamento ideale parabolico.

Eseguendo le traiettorie generate dal DMM otteniamo lanci che terminano spesso prima del canestro: dalla prima tabella in figura 5.7 notiamo che su 15 tentativi 4 vanno a canestro, mentre i restanti 11 terminano prima toccandolo sempre. I primi lanci corretti vengono ottenuti già al quinto e al sesto tentativo, per poi ripetersi successivamente al tredicesimo e quattordicesimo tentativo. Notiamo tuttavia la bontà dei tentativi non corretti che si avvicinano sempre all'esecuzione corretta del task. Eseguendo invece la traiettoria generata dal GMR si ottiene un movimento del braccio lento e a scatti, che fa terminare la pallina prima del canestro.



DMM - TRAIETTORIE GENERATE

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	21	0.148	TP
2	18	0.148	TP
3	23	0.140	TP
4	23	0.158	TP
5	23	0.156	C
6	23	0.158	C
7	23	0.145	TP
8	13	0.179	TP
9	23	0.161	TP
10	23	0.15	TP
11	23	0.149	TP
12	23	0.159	TP
13	23	0.151	C
14	23	0.15	TP
15	23	0.158	C

GMR - TRAIETTORIA GENERATA

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	16	0.10	P

Figura 5.7: Confronto tra il dataset iniziale, contenente tutte le 34 traiettorie corrette e non corrette, e le traiettorie generate dal modello Donut Mixture e dalla Gaussian Mixture Regression, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM e in nero la traiettoria generata dal GMR. Sulla destra le tabelle riguardanti i risultati dell'esecuzione delle traiettorie generate dai due modelli attraverso il robot reale. In arancio sono evidenziate le traiettorie corrette. (*)Valori possibili per la colonna risultato: P(prima), la pallina termina prima del canestro; TP(toccato prima), la pallina rimbalza sul bordo del canestro più vicino al robot; C(canestro); TD(toccato dopo), la pallina rimbalza sul bordo più distante; D(dopo), la pallina termina oltre il canestro; TL(toccato lateralmente), la pallina rimbalza su uno dei due bordi laterali.

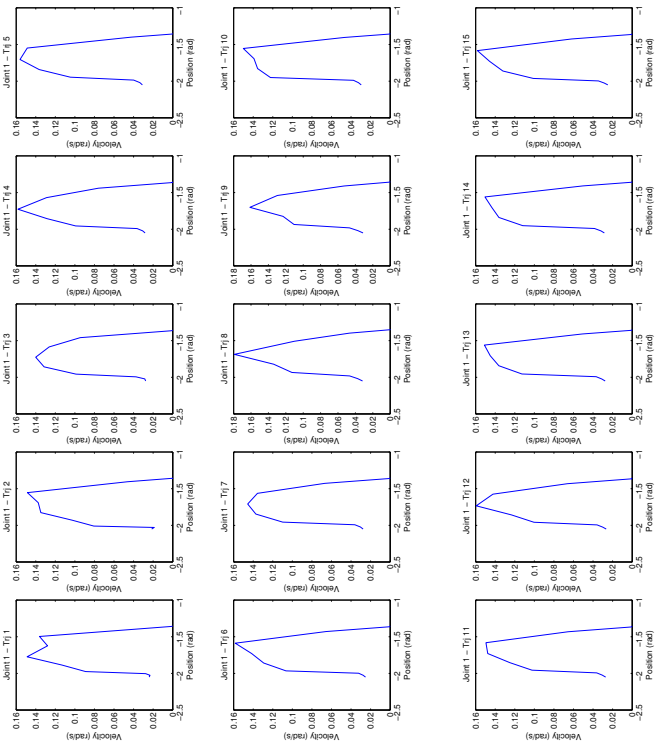
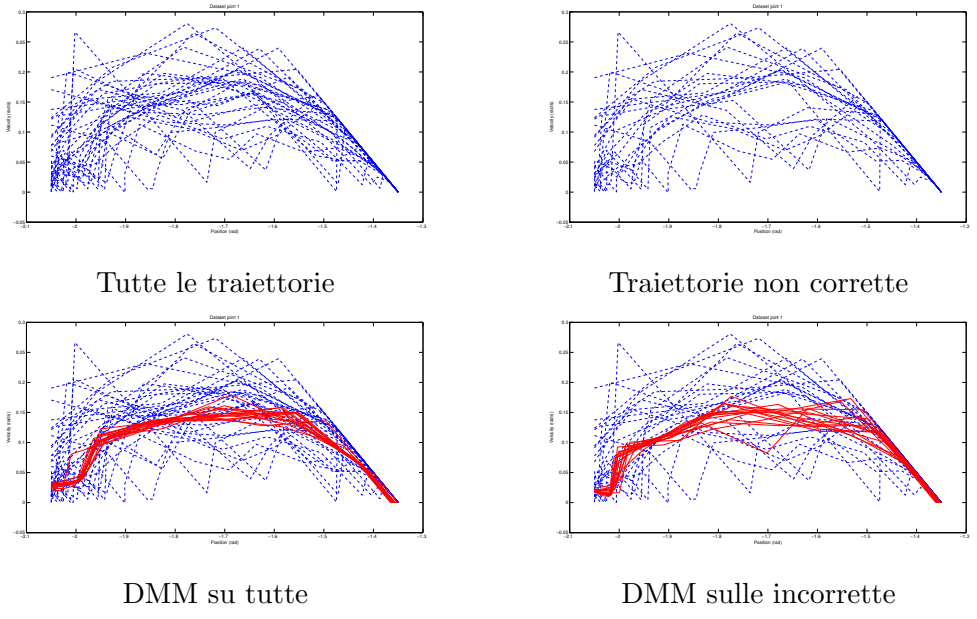


Figura 5.8: I 15 tentativi generati dall'applicazione del DMM, raffigurati nello spazio posizione-velocità.

Caso 2: traiettorie non corrette

In questa situazione filtriamo il dataset precedente, eliminando le traiettorie corrette: otteniamo un nuovo dataset di 23 traiettorie non corrette, riconducendoci all'apprendimento da sole dimostrazioni errate. Ciò che ci aspettiamo è un miglioramento del comportamento del modello Donut Mixture rispetto al caso precedente. In figura 5.10 abbiamo nuovamente il confronto tra il dataset iniziale e le traiettorie generate dai due modelli: si mantiene il comportamento esplorativo del DMM e di aspettazione del GMR. Se andiamo a valutare le differenze tra il precedente e il nuovo dataset e, rispettivamente, i risultati del DMM in entrambi, in figura 5.9, possiamo notare che:

- l'eliminazione delle traiettorie corrette comporta la presenza di una regione centrale libera nello spazio posizione-velocità;
- le traiettorie generate esplorano la nuova zona;
- la variabilità delle traiettorie generate aumenta, esplorando maggiormente lo spazio tra le traiettorie iniziali nelle zone centrali e finali dello spazio posizione-velocità.



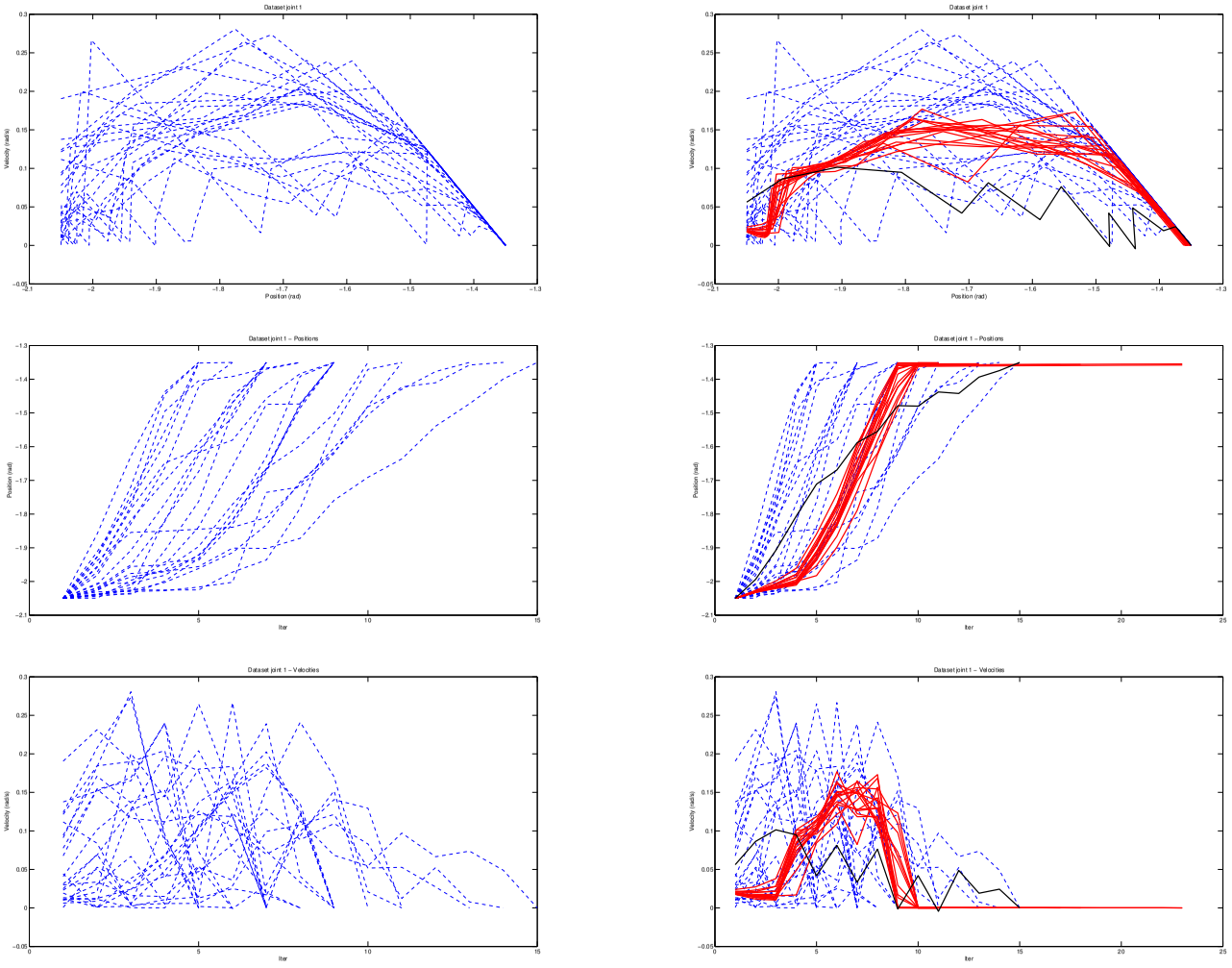
Tutte le traiettorie | Traiettorie non corrette

DMM su tutte | DMM sulle incorrette

Figura 5.9: In alto il confronto tra il dataset costituito da tutte le traiettorie e quello contenente le sole traiettorie incorrette. In basso il confronto tra il DMM applicato ai due dataset precedenti. In blu tratteggiato le traiettorie iniziali, in rosso le traiettorie generate dal modello.

Eseguendo le nuove traiettorie generate dal DMM possiamo notare ciò che ci aspettavamo: su 15 tentativi otteniamo 11 canestri e i restanti 4 lanci tuttavia si avvicinano al risultato corretto, toccando sempre il canestro (vedi prima tabella in figura 5.10). Diversamente dal caso precedente, il primo canestro si verifica già alla prima esecuzione del DMM sui dati incorretti. Otteniamo dunque un netto miglioramento del modello rispetto a prima: in questo caso i dati corretti non influenzano

il sistema, che è in grado di spaziare maggiormente all'interno delle regioni liberate dalle traiettorie corrette, in particolare tra l'intervallo di velocità compreso tra 0.15 e 0.18 rad/s. L'esecuzione del tentativo generato dal GMR non cambia rispetto alla situazione precedente: otteniamo ancora una traiettoria lenta che produce un movimento a scatti del braccio, facendo cadere la pallina molto prima del canestro.



DMM - TRAIETTORIE GENERATE

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	10	0.161	C
2	11	0.156	TP
3	11	0.168	C
4	11	0.163	C
5	11	0.153	C
6	11	0.134	TP
7	10	0.156	C
8	10	0.147	TP
9	11	0.143	C
10	17	0.151	C
11	11	0.164	C
12	13	0.15	C
13	23	0.173	TD
14	23	0.164	C
15	18	0.176	C

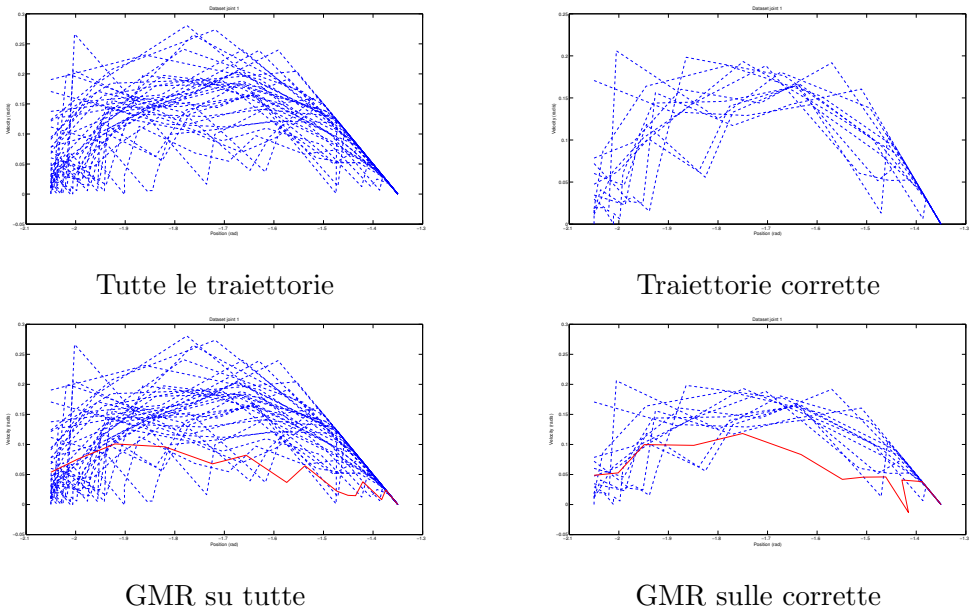
GMR - TRAIETTORIA GENERATA

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	15	0.101	P

Figura 5.10: Confronto tra il dataset iniziale, contenente le 23 traiettorie non corrette, e le traiettorie generate dal modello Donut Mixture e dalla Gaussian Mixture Regression, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM e in nero la traiettoria generata dal GMR. Sulla destra le tabelle riguardanti i risultati dell'esecuzione dei modelli attraverso il robot reale. In arancio sono evidenziate le traiettorie corrette. (*)Valori possibili per la colonna risultato: P(prima), la pallina termina prima del canestro; TP(toccato prima), la pallina rimbalza sul bordo del canestro più vicino al robot; C(canestro); TD(toccato dopo), la pallina rimbalza sul bordo più distante; D(dopo), la pallina termina oltre il canestro; TL(toccato lateralmente), la pallina rimbalza su uno dei due bordi laterali.

Caso 3: traiettorie corrette

Come ultimo caso valutiamo il comportamento dei due modelli sulle sole traiettorie corrette: abbiamo dunque un nuovo dataset di 9 traiettorie. Anche se disponiamo di un numero davvero limitato di dati, ci aspettiamo un miglioramento del GMR, essendo nella situazione ideale al RLfD. In figura 5.12 abbiamo nuovamente il confronto tra il dataset iniziale e le nuove traiettorie generate dai modelli. Differentemente dalle due situazioni precedenti, la variabilità delle traiettorie iniziali sia all'interno dello spazio posizione-velocità che per la sola componente di posizione risulta inferiore. In particolare possiamo osservare che l'intervallo di tempo necessario per le traiettorie a raggiungere la posizione finale si riduce rispetto al caso di utilizzo di tutte le traiettorie. Questo aspetto contribuisce alla creazione attraverso il GMR di una traiettoria con velocità maggiore e meno oscillante (vedi figura 5.11).

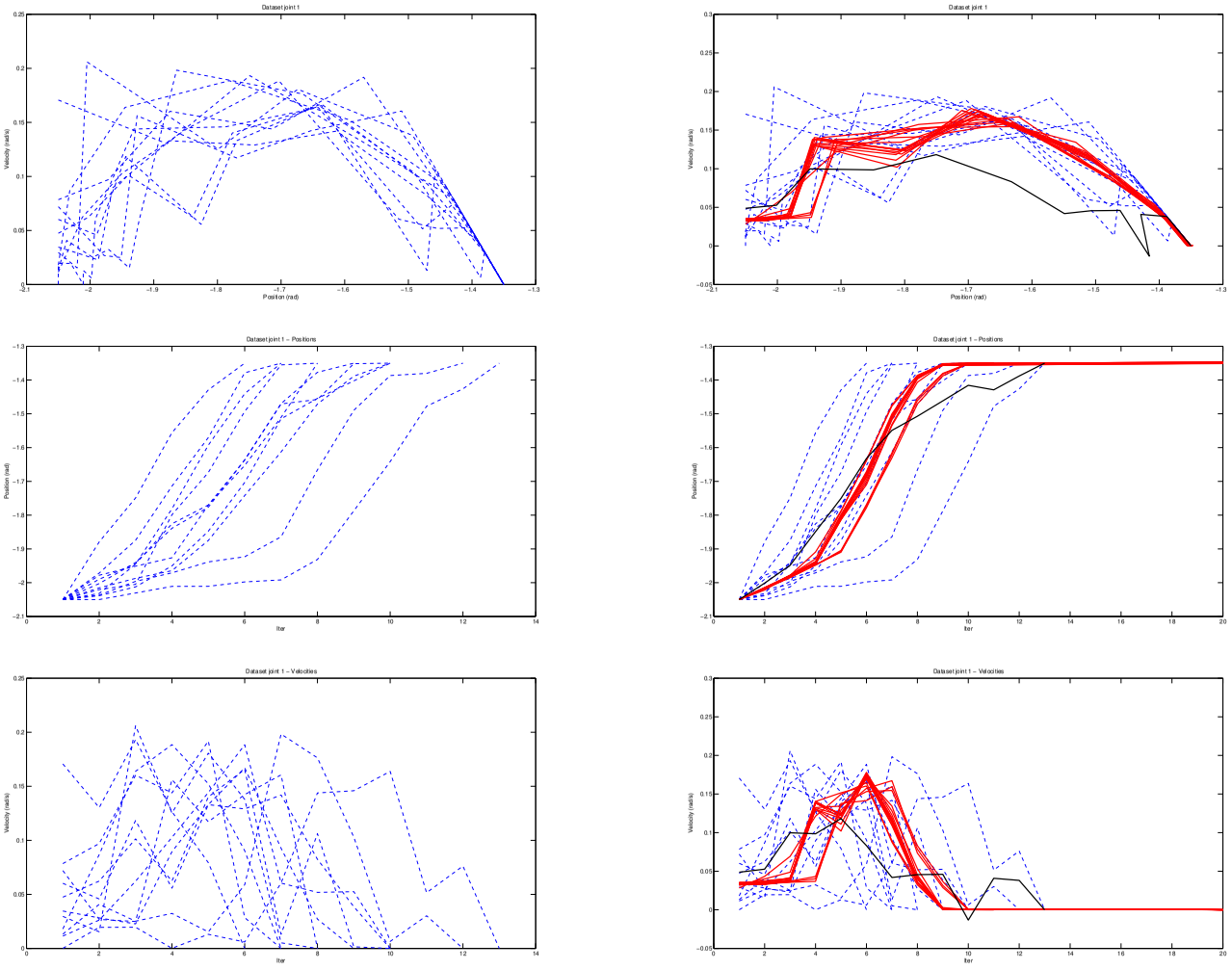


Tutte le traiettorie | Traiettorie corrette

GMR su tutte | GMR sulle corrette

Figura 5.11: In alto il confronto tra il dataset costituito da tutte le traiettorie e quello contenente le sole traiettorie corrette. In basso il confronto tra il GMR applicato ai due dataset precedenti. In blu tratteggiato le traiettorie iniziali, in rosso le traiettorie generate dal modello.

Per il DMM il comportamento è leggermente diverso: utilizzando traiettorie a bassa variabilità, lo spazio ristretto su cui esplorare comporta la generazione di nuove traiettorie vicine tra loro, anch'esse a variabilità ridotta. Eseguendo le traiettorie generate otteniamo su 15 tentativi 13 canestri, mentre i restanti due lanci toccano lateralmente il canestro, vedi risultati in figura 5.12. L'esplorazione interna allo spazio generato dalle traiettorie corrette comporta la generazione di traiettorie anch'esse corrette. Per quanto riguarda invece il comportamento del GMR, otteniamo una traiettoria leggermente più veloce rispetto alle precedenti: da 0.10 rad/s passiamo a 0.118 rad/s, ma il risultato dell'esecuzione non cambia, poiché la pallina cade sempre prima del canestro. Tuttavia il miglioramento sostanziale legato al GMR rispetto ai due casi precedenti riguarda la diminuzione dell'oscillazione.



DMM - TRAIETTORIE GENERATE

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	11	0.16	TL
2	11	0.169	C
3	20	0.171	C
4	20	0.175	C
5	20	0.173	C
6	20	0.154	C
7	20	0.159	C
8	20	0.164	C
9	20	0.171	C
10	20	0.174	C
11	20	0.167	TL
12	20	0.172	C
13	20	0.171	C
14	20	0.178	C
15	20	0.176	C

GMR - TRAIETTORIA GENERATA

Trai.	*Datapoint*	ξ_{max} *(rad/s)*	*Risultato(*)*
1	13	0.118	P

Figura 5.12: Confronto tra il dataset iniziale, contenente le 9 traiettorie corrette, e le traiettorie generate dal modello Donut Mixture e dalla Gaussian Mixture Regression, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM e in nero la traiettoria generata dal GMR. Sulla destra le tabelle riguardanti i risultati dell'esecuzione dei modelli attraverso il robot reale. In arancio sono evidenziate le traiettorie corrette. (*)Valori possibili per la colonna risultato: P(prima), la pallina termina prima del canestro; TP(toccato prima), la pallina rimbalza sul bordo del canestro più vicino al robot; C(canestro); TD(toccato dopo), la pallina rimbalza sul bordo più distante; D(dopo), la pallina termina oltre il canestro; TL(toccato lateralmente), la pallina rimbalza su uno dei due bordi laterali.

5.3.3 Due giunti

Nel caso a due giunti, riduciamo il movimento umano allo spostamento del giunto di pitch della spalla e del giunto di roll del gomito. Anche in questa situazione recuperiamo i dati relativi alla spalla e al gomito sinistro dell'utente e li mappiamo sui rispettivi giunti del robot, però del braccio opposto (vedi figura 5.13). Elaborando le 36 registrazioni iniziali, si ottiene un dataset di 22 traiettorie (vedi tabella a sinistra in figura 5.14).

A causa di problemi relativi al Nao non abbiamo potuto eseguire le traiettorie e quindi testare il comportamento. Possiamo solamente analizzare i risultati graficamente, considerando tutte le traiettorie iniziali come non corrette.

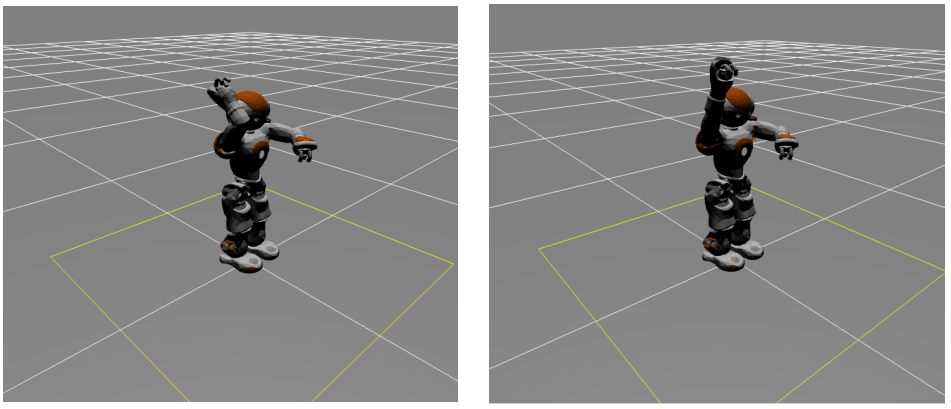


Figura 5.13: Posizioni iniziali e finali dei giunti di spalla e gomito durante il lancio a canestro.

Nelle due figure 5.15 e 5.16, relative al giunto della spalla e al giunto del gomito, presentiamo il confronto tra le traiettorie iniziali e le traiettorie generate dai due modelli, all'interno dello spazio posizione-velocità, posizione-tempo e velocità-tempo. Il comportamento dei modelli è lo stesso del caso ad un giunto: il DMM genera traiettorie esplorative all'interno dello spazio individuato dalle traiettorie di partenza e il GMR genera una traiettoria media nello spazio posizione-tempo, con valori di velocità oscillanti. Possiamo notare come per entrambi i giunti la velocità media della traiettoria ottenuta dal GMR sia inferiore rispetto alle velocità delle altre traiettorie: nel caso della spalla questo aspetto risulta meno evidente in quanto le dimostrazioni iniziali variano poco, a causa del movimento ristretto compiuto dalla spalla (da -0.90 a -1.20 rad); mentre è più evidente nel caso del gomito, in cui il movimento compiuto è completo (da -1.54 a -0.05 rad) e le dimostrazioni variano maggiormente.

TRAIETTORIE INIZIALI

Trai.	*Datapoint*	*Shoulder* $\dot{\xi}_{min}$ *(rad/s)*	*Elbow* $\dot{\xi}_{max}$ *(rad/s)*
1	5	-0.187	0.426
2	7	-0.097	0.402
3	6	-0.098	0.405
4	8	-0.216	0.328
5	6	-0.08	0.503
6	9	-0.112	0.425
7	9	-0.066	0.320
8	6	-0.109	0.504
9	5	-0.105	0.574
10	9	-0.087	0.502
11	9	-0.076	0.382
12	11	-0.053	0.394
13	10	-0.113	0.347
14	7	-0.071	0.396
15	5	-0.118	0.486
16	9	-0.089	0.378
17	7	-0.098	0.428
18	9	-0.097	0.432
19	9	-0.063	0.395
20	9	-0.07	0.414
21	9	-0.098	0.35
22	9	-0.067	0.531

DMM - TRAIETTORIE GENERATE

Traj.	*Datapoint*	*Shoulder* $\dot{\xi}_{min}$ *(rad/s)*	*Elbow* $\dot{\xi}_{max}$ *(rad/s)*
1	9	-0.06	0.396
2	8	-0.062	0.384
3	8	-0.063	0.401
4	10	-0.06	0.377
5	9	-0.063	0.403
6	8	-0.058	0.394
7	9	-0.06	0.344
8	8	-0.063	0.405
9	9	-0.062	0.409
10	9	-0.054	0.398
11	12	-0.059	0.405
12	9	-0.059	0.37
13	8	-0.058	0.399
14	9	-0.056	0.38
15	9	-0.058	0.386

GMR - TRAIETTORIA GENERATA

Traj.	*Datapoint*	*Shoulder* $\dot{\xi}_{min}$ *(rad/s)*	*Elbow* $\dot{\xi}_{max}$ *(rad/s)*
1	11	-0.059	0.251

Figura 5.14: Tabelle contenenti le informazioni, numero di datapoint e velocità massime e minime raggiunte per la spalla e il gomito, relative alle traiettorie iniziali, a sinistra, e le traiettorie generate dai due modelli DMM e GMR, a destra.

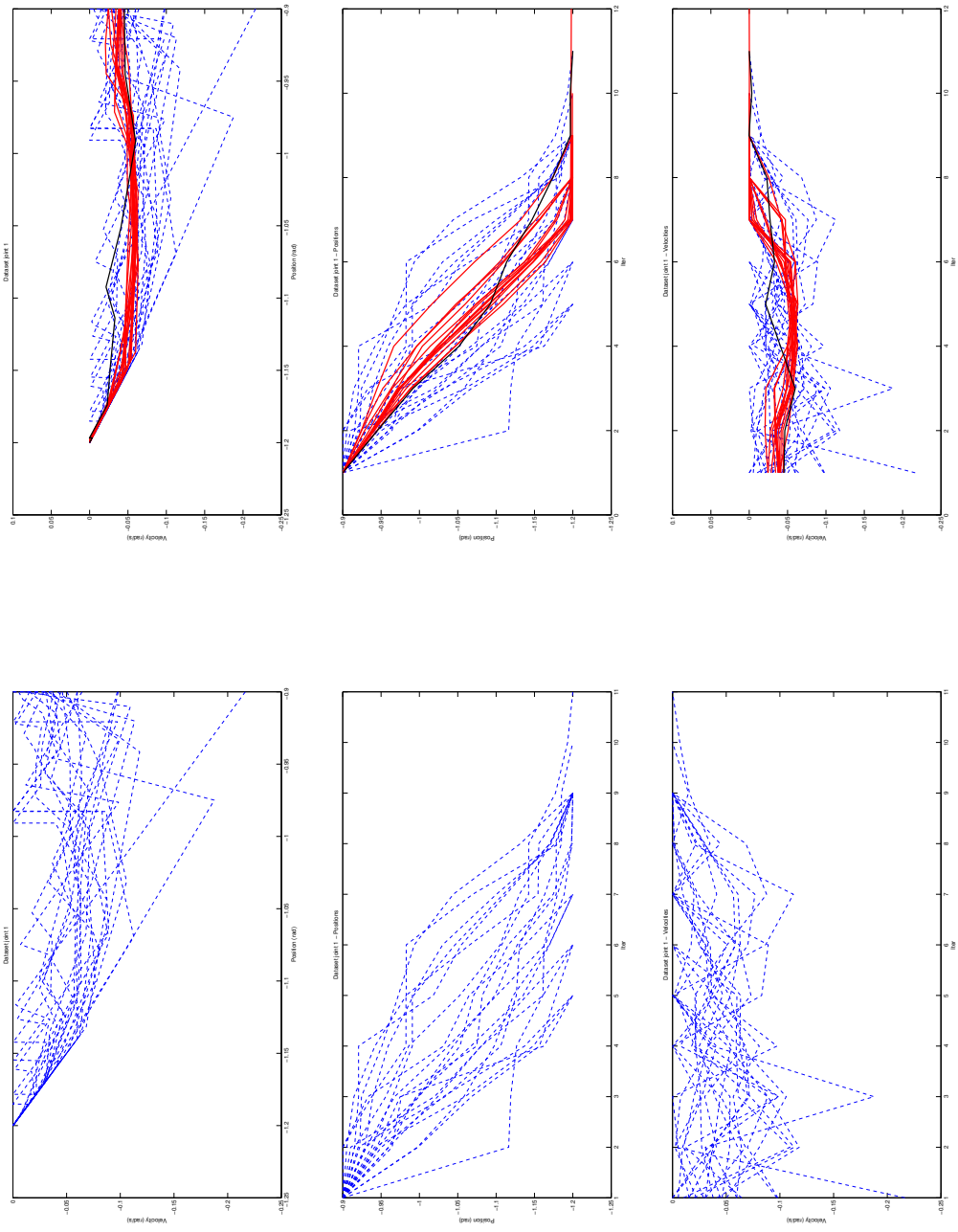


Figura 5.15: Giunto di pitch della spalla [-1.20;-0.90]. Confronto tra il dataset iniziale, contenente le 9 traiettorie corrette, e le traiettorie generate dal modello Donut Mixture e dalla Gaussian Mixture Regression, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM e in nero la traiettoria generata dal GMR.

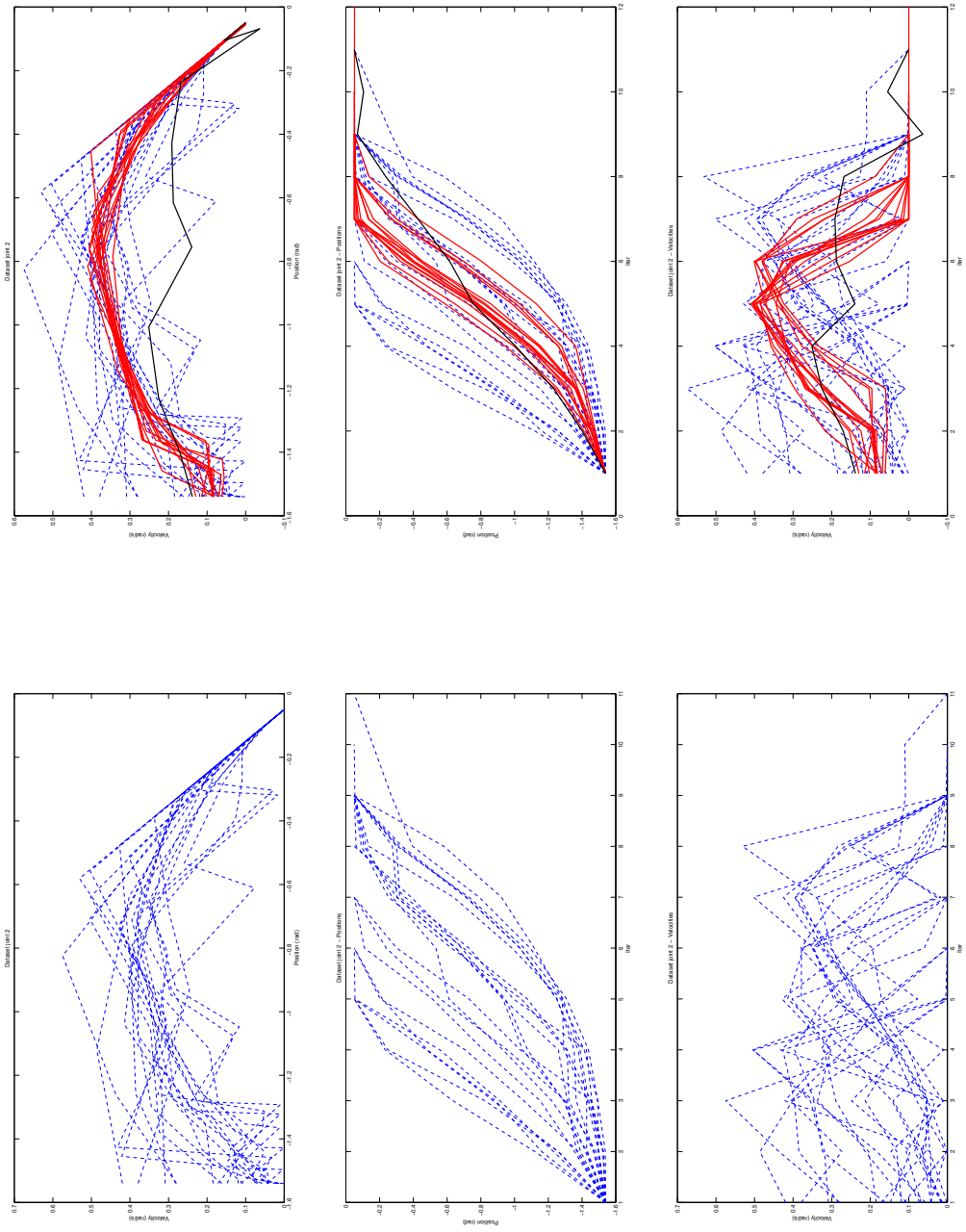


Figura 5.16: Giunto di roll del gomito [-1.54;-0.05]. Confronto tra il dataset iniziale, contenente le 9 traiettorie corrette, e le traiettorie generate dal modello Donut Mixture e dalla Gaussian Mixture Regression, all'interno dello spazio posizione-velocità (in alto), posizione-tempo (al centro) e velocità-tempo (in basso). In blu tratteggiato sono rappresentate le traiettorie di partenza, in rosso le traiettorie generate dal DMM e in nero la traiettoria generata dal GMR.

Capitolo 6

Conclusioni e sviluppi futuri

Con il lavoro descritto in questa tesi, abbiamo analizzato e modificato l'approccio al RLfF, esposto in [2], realizzandone una versione funzionante ma non ottimizzata. Dalla valutazione dei test possiamo concludere che il sistema creato è in grado di apprendere dalle dimostrazioni non corrette, conseguendo il comportamento preposto. Uno degli obiettivi futuri riguarda la conclusione dei test nel caso del basket a due giunti, iniziati ma interrotti per problemi meccanici legati al robot. Inoltre in futuro si potrebbero realizzare nuovi comportamenti, diversi dal giocare a basket, e sempre più complessi, che coinvolgano un numero sempre maggiore di giunti, in modo da testare la robustezza del modello Donut Mixture. Potremo non solo limitarci all'istruire il Nao ma sfruttare anche il robot manipolatore Comau da poco disponibile in laboratorio.

Un altro aspetto da valutare e su cui lavorare riguarda la creazione di un sistema di apprendimento da dimostrazioni non corrette, in grado di applicare le modifiche on-line, quindi di interagire durante la fase di esecuzione con il robot stesso e ricevere un feedback da esso. Inoltre si potrebbe valutare la realizzazione di una funzione di performance utile per valutare il raggiungimento del comportamento preposto: nella versione attuale, in fase di esecuzione verifichiamo le traiettorie generate dal modello in più tentativi e ci arrestiamo non appena otteniamo la correttezza del comportamento per ognuno di questi. La funzione dovrebbe consentire di valutare ogni nuova traiettoria in termini di probabilità di avere un successo o un fallimento in esecuzione.

Per quanto riguarda l'apprendimento da dimostrazioni umane, naturali e tramite utenti diversi, il nostro obiettivo principale consiste nel cercare di istruire il robot a compiere una azione quanto meno simile a quella impartita, usando un numero di giunti inferiore, in cui il movimento risulta maggiormente limitato. Nella realizzazione del comportamento con un singolo giunto, il modello Donut Mixture consente di ottenere nuove traiettorie che imitano i vari movimenti iniziali o li migliorano, andando a canestro o meno: l'andamento delle traiettorie generate risulta meno oscillatorio e più lineare rispetto alle traiettorie iniziali. Il Gaussian Mixture Regression contrariamente non consente di giungere allo stesso comportamento: le traiettorie presentano un andamento con più oscillazioni rispetto alle traiettorie iniziali e l'aspettazione comporta un abbassamento della componente di velocità. In fase di esecuzione dunque otteniamo un movimento lento e a scatti. Tra gli sviluppi fu-

turi, si prevede il completamento dell'apprendimento a due giunti e la realizzazione del comportamento con un numero sempre maggiore di giunti, fino alla completa imitazione del movimento.

Bibliografia

[1] Daniel H Grollman and Aude Billard. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE, 2011.

[2] Daniel H Grollman and Aude G Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331–342, 2012.

[3] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Handbook of robotics chapter 59: Robot programming by demonstration, 2007.

[4] Iris Hendrickx and Antal Van Den Bosch. Hybrid algorithms with instance-based classification. In *Machine Learning: ECML 2005*, pages 158–169. Springer, 2005.

[5] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 255–262. ACM, 2007.

[6] Elisa Tosello. Motion planning per un robot a molti gradi di libertà. Master's thesis, 2012.

[7] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[8] Giovanni Zilli, Luca Bergamaschi, and Manolo Venturin. Metodi di ottimizzazione. *Edizioni Libreria Progetto, Padova*, 2005.

[9] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[10] Mary Ann Branch, Thomas F Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[11] Thomas F Coleman and Arun Verma. A preconditioned conjugate gradient approach to linear equality constrained minimization. *Computational Optimization and Applications*, 20(1):61–72, 2001.

[12] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[13] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[14] Donald Goldfarb. Factorized variable metric methods for unconstrained optimization. *Mathematics of Computation*, 30(136):796–811, 1976.

[15] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[16] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[17] Jorge. Nocedal and Stephen J Wright. *Numerical optimization*, volume 2. Springer New York, 1999.

[18] Davide Zanin. Costruzione e testing in gazebo e v-rep di un modello per il robot umanoide nao. Master's thesis, 2013.

[19] Nicola Chessa. Programming by demonstration per robot manipolatore a n gradi di libertà. Master's thesis, 2013.