Università degli Studi di Padova

facoltà di ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Programming By Demonstration per robot manipolatore a n gradi di libertá

Laureando
Nicola Chessa
Matricola 1014413

Relatore
Emanuele Menegatti

Correlatore
Stefano Michieletto

Anno Accademico 2012–2013

Ai miei genitori,
per avermi sostenuto in questi anni di studio
e permesso di raggiungere questo traguardo.

Nicola Chessa

Abstract

Questa tesi è stata sviluppata presso il laboratorio IAS-Lab di Robotica Autonomia dell'Università degli studi di Padova.

Il progetto si basa sullo studio e sviluppo di un sistema software che applica i principi fondamentali della tecnica Programming By Demonstration ad un robot manipolatore. Gli obiettivi sono quelli di sfruttare tale tecnica per istruire il robot a compiere determinate azioni e valutare la qualità dei movimenti riprodotti. Le informazioni utilizzate per effettuare tali movimenti provengono da un sensore RGB-D e, pertanto, il punto di interesse del lavoro svolto è l'utilizzo di dati 3D per identificare le azioni da far compiere al braccio manipolatore. La successiva fase di riproduzione dei movimenti sul robot è stata implementata utilizzando il modello matematico GMM - Gaussian Mixture Model.

Il progetto è stato sviluppato utilizzando ROS come base di sviluppo per la creazione di un modello tridimensionale che rappresenti il robot preso in esame e per lo sviluppo di un'interfaccia che permetta un controllo agevole del robot stesso.

Indice

1 Introduzione **1**

2 Ros e software **5**
- 2.1 Ros . . . 5
  - 2.1.1 Comunicazione fra processi . . . 6
  - 2.1.2 Comandi all'interno di ROS . . . 8
- 2.2 Visualizzatori/Simulatori . . . 8
  - 2.2.1 Rviz . . . 8
  - 2.2.2 Gazebo . . . 9
- 2.3 Pacchetto TF . . . 10
- 2.4 OpenNI Tracker . . . 10
- 2.5 URDF (Unified Robot Description Format) . . . 14
  - 2.5.1 Link . . . 14
  - 2.5.2 Joint . . . 15

3 Comau Smart-5 Six **17**
- 3.1 Introduzione . . . 17
- 3.2 Caratteristiche Tecniche . . . 18

4 Comau Model **21**
- 4.1 Componenti e loro caratteristiche . . . 21
- 4.2 Descrizione URDF . . . 21
  - 4.2.1 Sistema di riferimento . . . 21
  - 4.2.2 Componenti del modello 3D e descrizione URDF . . . 22
  - 4.2.3 Giunti del modello e descrizione URDF . . . 27
- 4.3 Simulazione Modello . . . 28
  - 4.3.1 rViz . . . 28
  - 4.3.2 Gazebo . . . 29

5 Comau Controller **33**
- 5.1 Funzionalità . . . 33
- 5.2 Mapping . . . 34
  - 5.2.1 Joint1, Joint2, Joint3 . . . 35
  - 5.2.2 Joint4, Joint5, Joint6 . . . 42
- 5.3 Collision Avoidance . . . 43
- 5.4 Architettura SW . . . 45
- 5.5 I/O su file . . . 51
- 5.6 Interfaccia avvio del software . . . 53

6 Robot Learning 55
6.1 Robot Learning by Imitation . . . 55
6.1.1 Architettura Modello . . . 56
6.2 Modello Realizzato . . . 57
6.2.1 Gaussian Mixture Model (GMM) . . . 58
6.2.2 Gaussian Mixture Regression (GMR) . . . 67

7 Task 69
7.1 Descrizione Task . . . 69
7.2 Test . . . 70
7.2.1 Dataset A1 . . . 71
7.2.2 Dataset A2 . . . 74
7.2.3 Dataset B1 . . . 78
7.2.4 Dataset B2 . . . 81
7.3 Risultati . . . 84

8 Conclusioni 87

Bibliografia 89

1 Introduzione

Programming by Demonstration (PbD) è una tecnica di programmazione utilizzata nel campo della robotica il cui scopo è quello di istruire un robot a compiere una determinata azione. La comprensione di questo metodo necessita la conoscenza base di alcuni argomenti presenti in diversi ambiti di ricerca, quali l'interazione uomo-macchina, machine learning e computer vision.

L'utilizzo del Programming by Demonstration ebbe inizio circa 30 anni fa e, soprattutto nell'ultima decade, il suo impiego è aumentato in modo considerevole grazie agli indubbi vantaggi che l'utilizzo di tale metodo comporta. Innanzitutto PbD, denominato anche *Learning by Imitation*, è uno potente strumento di programmazione messo a disposizione agli operatori per la creazione di azioni da far eseguire al robot [5]; attraverso l'utilizzo di questo metodo allo sviluppatore end-user non è richiesta la conoscenza di alcun linguaggio di programmazione, tuttavia deve sapere quali movimenti il robot deve eseguire per il raggiungimento di un preciso *task* (compito); tale conoscenza è richiesta in quanto PbD è un processo che si occupa di imitare le azioni che il robot dovrà fare. L'utente quindi, sfruttando le capacità offerte dal Programming by Demonstration, non si dovrà occupare della scrittura di codice per comandare i movimenti del robot, ma avrà il compito di fargli imparare tali movimenti attraverso una serie di dimostrazioni, date attraverso un'opportuna *interfaccia*.

Questa peculiarità è stata il principale motivo di studio e conseguente sviluppo del paradigma PbD; come conseguenza del non dover più scrivere codice, si presenta un secondo beneficio, ossia quello riguardante i costi dovuti allo sviluppo e aggiornamento del software per il controllo di un robot, che sono ovviamente ridotti.

Un altro importante motivo che ha comportato un notevole interesse nel Programming by Demonstration è lo sviluppo, e la conseguente costruzione, di un sempre maggior numero di robot umanoidi. Un robot umanoide, oltre ad avere un aspetto simile all'uomo, richiede per sua natura un controllo flessibile e versatile dei movimenti; inoltre, il variare dell'ambiente in cui l'umanoide si può trovare, richiede la creazione di un'ampia varietà di task che il robot deve possedere per gestire le diverse situazioni che gli si possono presentare; grazie al PbD, l'aggiunta o la modifica di task può essere fatta facilmente, in modo affidabile e con tempistiche inferiori rispetto ad altre tecniche di programmazione. Il Programming by Demonstration ha avuto, per questi motivi, un maggior sviluppo per i robot umanoidi piuttosto che per quelli industriali, come per esempio i bracci manipolatori.

Il lavoro di questa tesi ha lo scopo di studiare l'utilizzo di questo approccio

proprio nell'ambito in cui è stato poco applicato, ovvero per il controllo di un braccio manipolatore. Il robot preso in esame è lo *Smart-5 Six*, prodotto dalla multinazionale italiana *Comau*. Lo stadio di progettazione e analisi inerente la tecnica di PbD è stato preceduto dallo sviluppo del software per il controllo del robot in un ambiente simulato; per questo scopo è stato utilizzato un sistema di sviluppo per la robotica *RSS (Robotic Software System)* Open Source: *ROS*.
ROS (Robotic Operating System) è un recente RSS nato per aiutare gli sviluppatori di robot nella creazione di applicazioni. ROS consente di astrarre la programmazione hardware dal sistema operativo utilizzato, fornisce driver per controllare diversi dispositivi hardware e mette a disposizione un insieme di librerie e strumenti che agevolano la scrittura di programmi per robot. Inoltre all'interno di ROS sono presenti vari simulatori, fra cui *RViz* e *Gazebo*, che permettono di inserire e controllare agevolmente nuovi modelli di robot. L'utilizzo di questo RSS Open Source (BSD) è inoltre supportato da un'ampia community; questo aspetto è di primaria importanza soprattutto nel campo robotico, essendo un campo in piena evoluzione; la condivisione di algoritmi e codice è quindi di fondamentale importanza, sia per il riutilizzo di funzionalità già implementate, sia per la risoluzione di eventuali problemi.

La progettazione di un modello tridimensionale del robot è stata indispensabile per gestire i movimenti del robot in un ambiente simulato. Il modello creato rispetta fedelmente il robot reale dal punto di vista geometrico, tant'è che sono stati utilizzati i dati ufficiali rilasciati gratuitamente dall'azienda Comau.

Il controllo dei movimenti dei giunti principali del braccio manipolatore avviene utilizzando dati RGB-D, acquisiti attraverso una telecamera RGB in aggiunta ad un sensore infrarosso (*Microsoft Kinect* o *Asus Xtion Live)*. I dati catturati dal sensore contengono le informazioni relative ai movimenti della persona registrata dal dispositivo rimappati su una struttura virtuale che rappresenta (scheletro); l'obiettivo è quindi controllare alcuni giunti del robot attraverso dei semplici movimenti che una persona può effettuare. Per i giunti che intervengono sull'end effector è stato preferito utilizzare un dispositivo di input con cui poter agire con maggiore precisione.

La parte della tesi inerente il *PbD* prende spunto dal lavoro presentato da Guenter et al. [11], nella quale viene descritto il modello matematico utilizzato per far apprendere al robot un determinato task. Rispetto al lavoro presente nel paper sono state apportate alcune modifiche, questo per adattare meglio il modello matematico al task preso in esame per il robot manipolatore utilizzato.

La tesi è così strutturata:

- nel capitolo 2 saranno presentati l'ambiente di lavoro ROS e gli strumenti utilizzati;
- nel capitolo 3 verrà fatta una breve descrizione tecnica del robot Comau Smart-5 Six;
- il capitolo 4 contiene la descrizione della creazione del modello simulato del robot all'interno di ROS;

- nel capitolo 5 verrà presentato il pacchetto software sviluppato per controllare il robot;
- il capitolo 6 è relativo alla parte di learning; verrà fatta un'introduzione teorica dell'argomento, ponendo attenzione al modello matematico utilizzato;
- nel capitolo 7 sarà descritta l'azione da far apprendere al robot e saranno riportati i risultati dei test relativi all'apprendimento del task;
- infine il capitolo 8 contiene le conclusioni e i possibili sviluppi.

2 | Ros e software

In questo capitolo verranno descritti gli strumenti utilizzati durante lo sviluppo del software; in particolar modo verrà fatta un'introduzione a *ROS* e, successivamente, saranno esposti con maggior dettaglio i programmi che questo RSS mette a disposizione.

2.1 Ros

ROS [4], acronimo di *Robot Operating System*, è un framework software per lo sviluppo di applicazioni robotiche rilasciato sotto licenza BSD. Questo sistema operativo open source fornisce un insieme di librerie atte a facilitare la progettazione di software per robot; lo sviluppo di ROS ebbe inizio nel 2007 e, nonostante sia un progetto relativamente recente, l'interesse verso tale framework è in continua ascesa; la sua sempre maggiore diffusione è dovuta principalmente a due motivi: l'architettura e il supporto a diversi robot.

- *Architettura*. è composta da 2 livelli: il primo dei quali è denominato semplicemente *ros* e offre servizi detti a basso livello, quali: a) software per il controllo fisico di un robot; b) astrazione dell'hardware di ogni singolo robot attraverso un'interfaccia comune a tutti i robot; c) driver per la gestione di diversi sensori; d) strumenti per la comunicazione fra diverse applicazioni robotiche, attraverso lo scambio di messaggi; e) gestione automatizzata dei pacchetti. Il secondo, chiamato *ros-pkg*, è un insieme di pacchetti contenente applicazioni che implementano diverse funzionalità relative, per esempio, ad argomenti quali la percezione, la simulazione e alcuni algoritmi di planning.

- *Supporto diversi robot*. All'interno di ROS è possibile controllare diversi robot, grazie ad appositi pacchetti software (*package*) già sviluppati e collaudati. Questi pacchetti, nonostante i robot possano essere fra loro molto diversi, seguono molto spesso una struttura software comune il che comporta il vantaggio di agevolare l'implementazione di nuovi robot.

Nelle applicazioni qui sviluppate sono state sfruttate in particolar modo la gestione automatizzata dei pacchetti e lo scambio di messaggi fra le applicazioni create, che sono alcune delle funzionalità del livello ros. Di rilevante importanza è stato il pacchetto *OpenNI* contenente il driver per la cattura delle informazioni RGB-D provenienti dai dispositivi *Microsoft Kinect* o *Asus Xtion Live*; questo

software, oltre a contenere i driver per la corretta rilevazione dei dispositivi precedentemente elencati, include anche un'applicazione che si occupa del *tracking* di una persona, ovvero della registrazione dei movimenti di alcune parti del corpo del soggetto rilevato.

La creazione del modello simulato di un robot avviene attraverso una descrizione in formato XML detta *URDF (Unified Robot Description Format)* nella quale non vengono inserite solamente le informazioni grafiche del robot, ma anche quelle inerenti la sua cinematica e la sua dinamica. ROS mette a disposizione una serie di tool che aiutano il processo di scrittura. In particolar modo è presente un'applicazione che controlla la corretta sintassi della rappresentazione nel formato URDF del robot.

L'utilizzo del formato *URDF* per la descrizione del modello del robot è stato successivamente sfruttato per visualizzarne i movimenti in fase di simulazione, operazioni per le quali ROS mette a disposizione due strumenti: *Rviz* e *Gazebo*.

Tutte le informazioni qui inserite si riferiscono alla versione *Fuerte* di ROS. In questo contesto non si ritiene necessario documentare le fasi di installazione e configurazione del software ROS, in quanto queste informazioni sono già presenti in modo chiaro nel tutorial[1] sul web. Tuttavia verrà inserito un paragrafo inerente l'installazione del pacchetto *OpenNI*, in quanto il pacchetto presente nel repository ufficiale di ROS presenta alcuni problemi di compatibilità.

2.1.1 Comunicazione fra processi

In questa sezione viene descritto il livello di comunicazione presente nell'architettura di ROS. Tale livello, detto anche *ROS Computational Graph Level*, si occupa del passaggio di informazioni fra i diversi processi di ROS in esecuzione, questi ultimi collegati fra loro attraverso una rete peer-to-peer gestita da ROS stesso.

Per rendere comprensibile il funzionamento di questo livello, è necessario introdurre i seguenti concetti:

1. Node;
2. Topic;
3. Message;
4. Service;
5. Bag.

Node Un nodo è un qualunque processo in grado di svolgere una qualsiasi attività computazionale. I nodi devono essere progettati per cooperare fra loro: ciascun nodo può comunicare con qualunque altro sfruttando i *topics*, che si possono "ascoltare" dopo che è avvenuta la registrazione a un determinato servizio.

Fra i possibili nodi che possono essere eseguiti, ve ne è uno di fondamentale importanza: il nodo *Master*, essenziale per gli altri nodi, in quanto si occupa della gestione delle registrazioni ai diversi servizi e della loro nomenclatura; inoltre ha l'impegno di tener traccia dei *publishers* e dei *subscribers*, ovvero di quei nodi che

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

inviano messaggi ad un certo topic e di quelli che si registrano a tale topic per leggere i messaggi pubblicati.

Topics Sono il mezzo di comunicazione per lo scambio di messaggi fra nodi. Questo tipo di comunicazione è unidirezionale, ciò significa che non è possibile gestire situazioni come quelle di request/response; i topics, tuttavia, permettono un facile scambio di messaggi fra nodi, che possono essere *publishers*, se si occupano di generare nuovi messaggi, oppure *subscribers*, nel caso in cui siano interessati ai dati presenti in un determinato topic. Ciascun topic è fortemente "tipizzato", in quanto al suo interno è possibile pubblicare/leggere un solo tipo di messaggio.

Message Struttura dati rigida composta da un prefissato numero e tipo di campi; ciascun campo può essere un tipo di dato primitivo, come un integer, float, boolean, ecc.; è supportato, inoltre, il tipo di dato array, utile per raggruppare più dati primitivi. I messaggi possono essere utilizzati anche per lo scambio di dati attraverso l'utilizzo dei servizi.

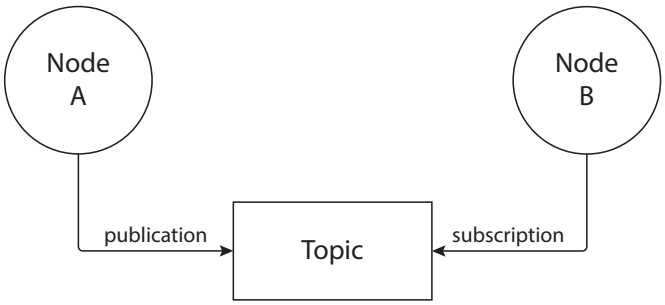


Figura 2.1: Publication/Subscription di un topic

Services Creati per supportare il modello request/response. Utilizzato spesso nei sistemi distribuiti. La procedura di richiesta/risposta avviene utilizzando un servizio definito da una coppia di messaggi: uno per la fase di request e uno per quella di response. Il funzionamento è il seguente: il nodo che fornisce un servizio, lo caratterizza associandogli un nome: il client che richiede il servizio, invia un messaggio di request al nodo e si mette in attesa fino alla ricezione della risposta.

Bag Formato di file presente in ROS per memorizzare i dati presenti nei messaggi. Questi file sono di primaria importanza perché, attraverso un'apposita applicazione, è possibile immagazzinare e riprodurre i messaggi: questa caratteristica è apprezzata soprattutto in fase di sviluppo in quanto permette al programmatore di controllare il corretto funzionamento del proprio software utilizzando in ingresso gli stessi dati. L'operazione di salvataggio dei dati è sfruttata soprattutto per le informazioni generate dai dispositivi hardware: una volta immagazzina-

te infatti, è possibile elaborare tali informazioni anche se i dispositivi non sono online.

2.1.2 Comandi all'interno di ROS

All'interno di ROS sono presenti diversi comandi per lo sviluppo e l'esecuzione delle proprie applicazioni dei quali quelli utilizzati principalmente sono, tuttavia, due:

- `rosmake`: il suo scopo è quello di compilare il codice sorgente di un determinato *package*;
- `roslaunch`: questo comando legge un file in formato *XML* dove sono descritti i nodi da avviare e gli eventuali parametri con cui avviarli.

2.2 Visualizzatori/Simulatori

In questa sezione verranno descritte le caratteristiche delle applicazioni per simulare i movimenti del robot.

2.2.1 Rviz

Software di ROS dedicato alla visualizzazione di modelli tridimensionali, utilizzato per rappresentare i robot ed il loro moto e rilasciato, come l'intera libreria a cui appartiene, sotto licenza BSD. Rviz è un visualizzatore e il suo compito principale è renderizzare oggetti a schermo. Gli aspetti relativi alla fisica del robot non vengono presi in considerazione da questo programma.

Una volta avviato Rviz, il programma mette in primo piano la visualizzazione del mondo tridimensionale utilizzato e i diversi oggetti al momento presenti; oltre alla finestra che si occupa del rendering, Rviz a margine presenta le seguenti sezioni:

- *Displays*: in questa finestra vengono elencati tutti gli oggetti che sono visualizzati nella schermata di rendering con la possibilità di rimuovere o inserire altri display (oggetti). Nella vasta gamma messa a disposizione da Rviz, i display più comunemente usati sono: 1. *GlobalOptions*, in cui vengono descritte le caratteristiche generali del mondo 3D creato; 2. *Grid*, che presenta diverse opzioni per personalizzare la base del mondo 3D; 3. *Robot Model*, che si occupa della visualizzazione del modello del robot preso in esame; 4. *TF*, che si occupa della visualizzazione dell'albero delle trasformate tf; tale albero visualizza come sono posizionati i giunti fra loro, la cui descrizione segue quella indicata nel formato *URDF*, di cui si parlerà successivamente. Per ciascun giunto è possibile visualizzare il sistema di riferimento XYZ a lui associato; per il colore degli assi viene utilizzata la convenzione RGB, in cui l'asse X viene colorato di rosso, l'asse Y di verde e infine l'asse Z

con il colore blu. Nel capitolo 2.3 viene delineato con maggior dettaglio il funzionamento delle trasformate tf.

- *Tool Properties*: il pannello permette di associare una posizione di partenza e una posizione obiettivo ai due rispettivi topic di ROS.

- *Views*: in questa finestra è possibile scegliere il tipo di telecamera con cui visualizzare il mondo tridimensionale creato: sono presenti tre tipi di telecamera: 1. *Orbit*: camera orbitale, permette la rotazione della scena attorno ad un punto focale; 2. *FPS*: first-person camera, permette la rotazione della scena nella stessa maniera in cui una persona ruota la testa; 3. *TopDownOrtho*: visualizzazione ortogonale del mondo lungo l'asse Z rispettivo al frame del robot.

- *Selection*: pannello di fondamentale importanza per la descrizione dei link selezionati: per ciascuno di essi, vengono indicati l'orientamento e la posizione rispetto all'origine del sistema di riferimento fisso del mondo.

2.2.2 Gazebo

Gazebo [2] è un simulatore 3D progettato per essere utilizzato con diversi tipi di robot, sia in ambienti interni che in ambienti esterni. Oltre ai robot il software supporta diversi oggetti e tipi di sensori; il programma, inoltre, ha la capacità di simulare sia i feedback provenienti dai sensori stessi, sia l'interazione fra gli oggetti presenti nel mondo tridimensionale.

Questo simulatore è stato progettato per lo sviluppo e conseguente testing di algoritmi per robot. La parte di simulazione è stata concepita per rappresentare fedelmente i movimenti dei robot reali, facilitando così la fase di testing presente durante lo sviluppo di un'applicazione robotica; grazie alla simulazione non è più necessario effettuare le prove sul robot reale, evitando così il sorgere di possibili problemi, quali gli errori hardware del robot o comportamenti inaspettati del software che rischiano di provocare danni al robot stesso. La bontà della simulazione è garantita grazie all'utilizzo di *ODE (Open Dynamics Engine)* come motore fisico e di *OGRE (Object-Oriented Graphics Rendering Engine)* per il rendering di tutti gli oggetti presenti nella scena tridimensionale.

ODE è un motore fisico che si occupa principalmente della simulazione della dinamica dei corpi rigidi e della funzionalità detta *collision detection*, utilizzata per verificare la presenza di collisioni fra gli oggetti inseriti nella scena. OGRE si occupa solo della visualizzazione della scena: questo motore di rendering ha il vantaggio di essere supportato da diversi software di modellazione grafica; permette quindi la visualizzazione di modelli grafici descritti in diversi formati, fra cui il formato `.dae` (Collada format [1]).

Uno dei vantaggi nell'uso di Gazebo deriva dalla sua architettura modulare: grazie ad essa, infatti, è possibile inserire nuove funzionalità attraverso lo sviluppo di propri *plugin*; questo aspetto facilita, per esempio, la creazione di nuovi mo-

delli di robot o di sensori, tant'è che la quantità di robot che si possono simulare con Gazebo è in continuo aumento.

Un ulteriore beneficio nell'uso di Gazebo deriva dalla documentazione che si può trovare nel sito ufficiale[2] del software e dalla comunità presente nella piattaforma *Gazebo Answer*, dove è possibile visionare un'ampia quantità di risposte relative ai dubbi di altri utenti e, nel caso fosse necessario, pubblicare domande relative ad eventuali difficoltà riscontrate.

2.3 Pacchetto TF

Il pacchetto TF permette di tener traccia nel tempo dei movimenti e, quindi, delle posizioni dei diversi giunti di un robot. Questo software è perciò di rilevante importanza nelle applicazioni robotiche: infatti, un robot possiede tipicamente molti giunti, ciascuno dei quali è caratterizzato da una determinata posizione e rotazione nello spazio.

TF associa un frame a ciascun giunto, il quale a sua volta rappresenta un sistema di riferimento; questi frame sono organizzati all'interno del pacchetto attraverso una struttura dati ad albero: in questo modo, è possibile verificare le relazioni che vi sono fra loro, come per esempio il vettore distanza o la matrice di rotazione, matrice che indica come un frame è ruotato rispetto ad un altro. É possibile controllare queste relazioni in tempo reale e per qualsiasi tipo di frame.

2.4 OpenNI Tracker

OpenNI (Open Natural Interaction) [3] è un framework open source utilizzato per lo sviluppo di librerie o applicazioni software che richiedono l'utilizzo di dispositivi di registrazione di dati RGB-D, quali i sensori 3D. Attraverso questo framework è possibile gestire diversi dispositivi hardware, quali per esempio *Microsoft Kinect* o *Asus Xtion Live*; all'interno di questo software sono presenti i driver che si occupano della gestione dei dispositivi supportati e viene offerto al programmatore un middleware, contenente un insieme di librerie che permettono l'uso dell'hardware attraverso delle semplici funzioni. Oltre al controllo delle fonti RGB-D, OpenNI offre al programmatore delle API che, tra le altre cose, permettono il tracking dello scheletro di una persona.
L'utilizzo di OpenNI comporta quindi i seguenti vantaggi:

- supporto a diversi sensori 3D;
- facilita lo sviluppo per i programmatori;
- libreria flessibile e semplice;
- se necessario, permette la programmazione dei dispositivi a basso livello.

[2] http://www.gazebosim.org/documentation.html

Oltre alle funzionalità descritte, OpenNI è dotata di un ampio supporto anche grazie ad una vasta community[3] e, di conseguenza, l'utente può ricercare le soluzioni ai proprio problemi ed eventualmente chiedere assistenza.

All'interno di Ros Fuerte è presente il plugin *openni_tracker* che si occupa della registrazione dei movimenti dello scheletro di una o più persone. In questa versione, tuttavia, tale plugin non funziona correttamente a causa di alcuni conflitti derivanti dall'utilizzo di versioni più aggiornate della libreria OpenNI. A differenza delle precedenti applicazioni, quindi, verrà descritta la procedura di installazione del software al fine di ottenere una versione efficiente di questo plugin; verrà successivamente descritto il suo funzionamento.

Installazione

Per ottenere il plugin di tracking funzionante, per prima cosa è necessario verificare che nel sistema siano presenti i seguenti pacchetti software:

- `libopenni-nite-dev`
- `libopenni-dev`
- `libopenni-sensor-prime-dev`

Per appurare la presenza di tali pacchetti, aprire un terminale, eseguire il comando `niReg -l` e verificare di ottenere un risultato simile a questo:

```
(...)
OpenNI version is 1.5.2.23
(...)
/usr/lib/libXnVFeatures.so (compiled with OpenNI 1.0.0.22):
    Scene: PrimeSense/XnVSceneAnalyzer/1.3.0.17
    User: PrimeSense/XnVSkeletonGenerator/1.3.0.17

/usr/lib/libXnVHandGenerator.so (compiled with OpenNI 1.0.0.22):
    Gesture: PrimeSense/XnVGestureGenrator/1.3.0.17
    Hands: PrimeSense/XnVHandTracker/1.3.0.17
```

Successivamente, scaricare dal sito ufficiale[4] di OpenNI il file:
`NITE-Bin-Linux-x86-v1.5.2.21.tar.zip`
contenente le nuove librerie già compilate. Quindi scompattare l'archivio scaricato, entrare nella directory ed eseguire i seguenti comandi:

```
sudo ./uninstall.sh
sudo ./install.sh
```

Per constatare che tutto sia andato a buon fine, rieseguire il comando `niReg -l` e controllare che l'output sia simile al seguente:

```
(...)
OpenNI version is 1.5.2.23
```

[3] http://community.openni.org/openni
[4] http://www.openni.org/openni-sdk/openni-sdk-history-2/#.UXflRpBdW0x

```
(...)
/usr/lib/libXnVFeatures_1_3_0.so (compiled with OpenNI 1.0.0.22):
    Scene: PrimeSense/XnVSceneAnalyzer/1.3.0.17
    User: PrimeSense/XnVSkeletonGenerator/1.3.0.17

/usr/lib/libXnVFeatures_1_3_1.so (compiled with OpenNI 1.2.0.8):
    Scene: PrimeSense/XnVSceneAnalyzer/1.3.1.8
    User: PrimeSense/XnVSkeletonGenerator/1.3.1.8

/usr/lib/libXnVFeatures_1_4_1.so (compiled with OpenNI 1.3.2.3):
    Scene: PrimeSense/XnVSceneAnalyzer/1.4.1.2
    User: PrimeSense/XnVSkeletonGenerator/1.4.1.2

/usr/lib/libXnVFeatures_1_4_2.so (compiled with OpenNI 1.3.4.6):
    Scene: PrimeSense/XnVSceneAnalyzer/1.4.2.5
    User: PrimeSense/XnVSkeletonGenerator/1.4.2.5

/usr/lib/libXnVFeatures_1_5_2.so (compiled with OpenNI 1.5.4.0):
    Scene: PrimeSense/XnVSceneAnalyzer/1.5.2.21
    User: PrimeSense/XnVSkeletonGenerator/1.5.2.21

/usr/lib/libXnVHandGenerator_1_3_0.so(compiled with OpenNI 1.0.0.22):
    Gesture: PrimeSense/XnVGestureGenrator/1.3.0.17
    Hands: PrimeSense/XnVHandTracker/1.3.0.17

/usr/lib/libXnVHandGenerator_1_3_1.so (compiled with OpenNI 1.2.0.8):
    Gesture: PrimeSense/XnVGestureGenrator/1.3.1.8
    Hands: PrimeSense/XnVHandTracker/1.3.1.8

/usr/lib/libXnVHandGenerator_1_4_1.so (compiled with OpenNI 1.3.2.3):
    Gesture: PrimeSense/XnVGestureGenrator/1.4.1.2
    Hands: PrimeSense/XnVHandTracker/1.4.1.2

/usr/lib/libXnVHandGenerator_1_4_2.so (compiled with OpenNI 1.3.4.6):
    Gesture: PrimeSense/XnVGestureGenrator/1.4.2.5
    Hands: PrimeSense/XnVHandTracker/1.4.2.5

/usr/lib/libXnVHandGenerator_1_5_2.so (compiled with OpenNI 1.5.4.0):
    Gesture: PrimeSense/XnVGestureGenerator/1.5.2.21
    Hands: PrimeSense/XnVHandTracker/1.5.2.21
```

Nota: questa soluzione è presente in ROS Answers[5]. Al momento della scrittura della tesi tale problema persiste, tuttavia è consigliabile provare a far eseguire il plugin prima di iniziare questa procedura.

Utilizzo

L'utilizzo di ROS OpenNI Skeleton Tracker è piuttosto semplice: prima di avviare il plugin, è necessario che una persona sia posizionata in piedi di fronte alla telecamera RGB-D; all'avvio del plugin, una volta che la persona viene registrata, è possibile tener traccia dei movimenti delle sue parti del corpo.

[5] http://answers.ros.org/question/42654/openninite-incompatible-in-fuertegroovy-on-precise/

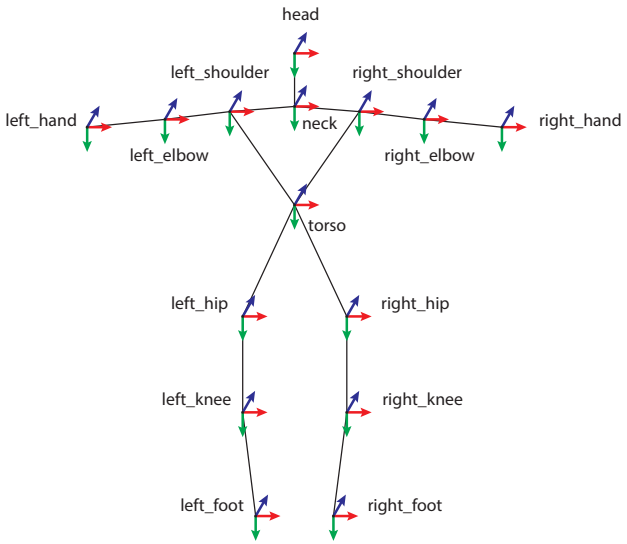
Di seguito viene riportato l'elenco delle parti del corpo rilevate dal plugin:

Tabella 2.1: Etichette OpenNI Skeleton Tracker.

Etichetta	Descrizione
/skeleton_1_head	testa
/skeleton_1_left_elbow	gomito sinistro
/skeleton_1_left_foot	piede sinistro
/skeleton_1_left_hand	mano sinistra
/skeleton_1_left_hip	anca sinistra
/skeleton_1_left_knee	ginocchio sinistro
/skeleton_1_left_shoulder	spalla sinistra
/skeleton_1_neck	collo
/skeleton_1_right_elbow	gomito destro
/skeleton_1_right_foot	piede destro
/skeleton_1_right_hand	mano destra
/skeleton_1_right_hip	anca destra
/skeleton_1_right_knee	ginocchio destro
/skeleton_1_right_shoulder	spalla destra
/skeleton_1_torso	busto

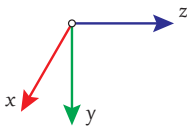
Per visualizzare i movimenti della persona in tempo reale, è stato utilizzato il visualizzatore *rviz* (2.2.1). All'interno di questa applicazione, è possibile sfruttare il display *TF* per tener traccia delle relazioni che vi sono fra i diversi frame, ovvero fra le parti del corpo della persona registrata. La disposizione delle etichette del pacchetto *Skeleton Tracker* è quella di figura 2.2.

Figura 2.2: Etichette Skeleton Tracker



Oltre ai frame associati alla persona, è presente un ulteriore frame che descrive la posizione della telecamera RGB-D nello spazio; questo frame è chiamato `/openni_rgb_optical_frame` e il suo sistema di riferimento è quello in figura 2.3;

Figura 2.3: Sistema di riferimento telecamera RGB-D



2.5 URDF (UNIFIED ROBOT DESCRIPTION FORMAT)

All'interno di ROS il modello di ciascun robot deve essere descritto tramite un file di tipo XML detto *URDF (Unified Robot Description Model)*. Questo formato è utilizzato per descrivere diverse caratteristiche di un robot, fra le quali:

- aspetto grafico, cioè come si presenta il robot;
- dinamica e cinematica;
- modello relativo alle collisioni.

Il formato prevede una descrizione del robot utilizzando una struttura ad albero i cui nodi principali sono i *link*, utilizzati per descrivere le parti rigide del robot e i *joint*, utilizzati per collegare fra loro i link.

2.5.1 LINK

I *link* descrivono i componenti rigidi di un robot: a ciascuno di essi è associato un nome e, all'interno della sua dichiarazione, è possibile inserire le descrizioni relative all'aspetto, alle caratteristiche fisiche come l'inerzia e al modello per la gestione delle collisioni.

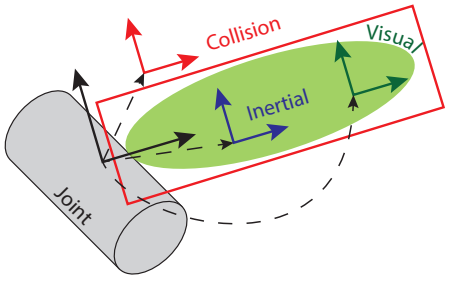


Figura 2.4: Esempio di un link

É possibile specificare un link attraverso il tag `<link>` e, al suo interno, è inoltre possibile precisare le sue caratteristiche usando i seguenti tag XML:

- `<visual>` indica l'aspetto grafico del link preso in esame; la geometria del link può essere specificata o attraverso una delle primitive grafiche messe a disposizione dal formato URDF stesso (cilindro, parallelepipedo, sfera), oppure utilizzando una semplice mesh.[6] É possibile utilizzare il tag `<origin>` per indicare la posizione del sistema di riferimento utilizzato per la grafica rispetto a quello del link stesso.

- `<inertial>` esprime le caratteristiche relative alla fisica del link, come la massa e l'inerzia: la prima è descritta da un numero scalare, mentre la seconda deve essere specificata attraverso una matrice 3×3. Come per `<visual>`, anche in questo tag è possibile precisare l'origine rispetto al sistema di riferimento del link, aggiungendo il tag `<origin>`.

- `<collision>` descrive il modello relativo alla collisione del link considerato: il modello viene indicato con il tag `geometry`, al cui interno deve essere fatta o una stima delle dimensioni del link (per esempio attraverso una bounding box), oppure può essere indicata la mesh del link stesso. É possibile aggiungere il tag `<origin>`, il cuo significato è analogo a quello descritto precedentemente per gli altri tag.

Per la descrizione del link sono disponibili altre opzioni, che si possono trovare nella sezione[7] del sito di ROS dedicata agli URDF.

2.5.2 Joint

Nel formato URDF i *joint* sono utilizzati per delineare come sono collegati fra loro i link attraverso una relazione di tipo *padre-figlio* e per specificare la cinematica e la dinamica di un giunto del robot. Come per i *link*, anche i *joint* sono identificati univocamente da un nome.
I giunti possono essere di diversi tipi ciascuno dei quali deve essere specificato usufruendo dell'opzione `type` prevista dal tag `<joint>`. Un giunto può essere:

1. *revolute*: giunto il cui movimento prevede la rotazione attorno ad un suo asse; il valore del giunto deve essere compreso in un intervallo limitato inferiormente e superiormente;

2. *continuous*: giunto simile a revolute, ma che non ha alcun limite di rotazione;

3. *prismatic*: giunto che scorre lungo uno dei suoi assi in un intervallo limitato;

4. *fixed*: giunto fisso, bloccato in ciascun grado di libertà;

5. *floating*: il giunto può variare di posizione in tutti e 6 i gradi di libertà;

[6] formati di mesh supportati: `dae, stl`
[7] http://www.ros.org/wiki/urdf/XML/link

6. *planar*: il giunto permette il movimento in un piano perpendicolare all'asse.

Oltre al tipo di giunto, all'interno del tag `<joint>`, devono essere inseriti i seguenti tag figli:

- `<origin>` utilizzato per specificare la posizione del link figlio rispetto a quello padre; il giunto è posizionato nell'origine del link figlio;
- `<parent>` nome del link padre;
- `<child>` nome del link figlio;
- `<axis>` utilizzato per specificare l'asse del giunto. Questo asse sarà quello di rotazione se il giunto è *revolute/prismatic*, di traslazione se invece è di tipo *prismatic*; viene invece utilizzata la normale per i giunti di tipo *planar*. L'asse deve essere specificato utilizzando un vettore normalizzato (x, y, z).
- `<limit>` consente di specificare l'intervallo di valori che il giunto può assumere; l'intervallo deve essere indicato con le opzioni `lower` e `upper`; inoltre, devono essere indicate due opzioni: a) `effort`: specifica la massima forza applicabile al giunto; b) `velocity`: consente di indicare la velocità massima applicabile al giunto. Questo tag può essere utilizzato se il giunto è di tipo *revolute* o *prismatic*.

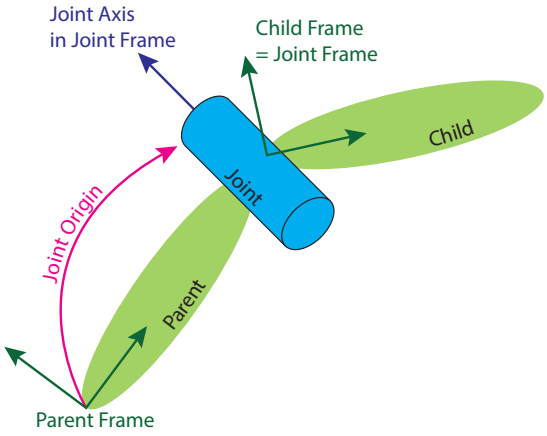


Figura 2.5: Esempio di un joint

3 | Comau Smart-5 Six

In questo capitolo verrà introdotto il robot *Comau Smart-5 Six* [8]. Sarà fatta una breve descrizione tecnica, sviluppata ponendo maggiore enfasi sugli aspetti utilizzati in questo progetto e tralasciando caratteristiche in questo ambito secondarie, come per esempio la meccanica del robot.

Tutti i dettagli tecnici sono liberamente visualizzabili nei documenti ufficiali [7] rilasciati dall'azienda *Comau* e liberamente scaricabili dal sito ufficiale[1].

3.1 Introduzione

Il Comau Smart-5 Six appartiene al progetto *Smart* dell'azienda, il quale presenta un insieme di celle robotizzate e soluzioni integrate di processo per tutte le categorie di carico caratterizzate da dimensioni ridotte, elevato volume di lavoro ed elevata precisione di movimento.

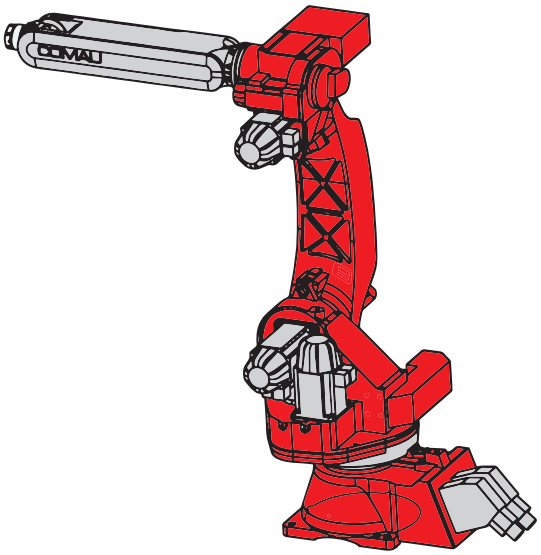


Figura 3.1: Comau Smart-5 Six

Questo robot fa parte della famiglia *Smart-5 Six* sviluppata dalla Comau per applicazioni di manipolazione leggera e saldatura ad arco; fra le caratteristiche più interessanti, sono da evidenziare:

[1] http://www.comau.com/ita/download/Pages/Download.aspx

- predisposizione al montaggio di numerosi dispositivi opzionali;
- grande capacità di orientamento del polso in spazi ristretti, grazie alle sue ridotte dimensioni;
- elevata ripetibilità.

Per quanto concerne la gestione del robot, gli interventi manutentivi sono minimi, intuitivi e non richiedono attrezzature particolari; inoltre, l'intercambiabilità fra robot della stessa versione è garantita facendo si che possano essere rapidamente sostituiti senza richiedere pesanti interventi correttivi sul programma.

3.2 Caratteristiche Tecniche

Il Comau Smart-5 Six è un robot antropomorfo con 6 gradi di libertà; di seguito sono riportate le principali caratteristiche fisiche/tecniche del robot: sono elencate solo quelle ritenute utili ai fini del progetto e facilmente reperibili nel sito dell'azienda *Comau*.

In tabella 3.1 sono riportate le principali caratteristiche tecniche.

Tabella 3.1: Caratteristiche Tecniche Comau Smart-5 Six

Descrizione	Valore
Struttura / nº assi	Antropomorfo / 6
Carico al poldo	6 kg
Carico Supplementare su avambraccio	10 kg
Coppia Asse 4	11,7 Nm
Coppia Asse 5	11,7 Nm
Coppia Asse 6	5,8 Nm

Nella tabella 3.2 è riportata la *corsa* per ciascun asse, ovvero il massimo intervallo di valori che un giunto può assumere; i valori riportati sono validi nel caso in cui il robot sia installato a terra e nell'ipotesi in cui non sia presente alcun ostacolo nello spazio dei movimenti del robot.

Tabella 3.2: Corsa assi Comau Smart-5 Six

Nome Asse	Limite inferiore	Limite superiore
Asse 1	-170º	+170º
Asse 2	-85º	+155º
Asse 3	-170º	0º
Asse 4	-210º	+210º

Tabella 3.2: continua nella prossima pagina

Tabella 3.2: continua dalla pagina precedente

Descrizione	Limite inferiore	Limite superiore
Asse 5	-130°	+130°
Asse 6	-2700°	+2700°

Infine nella tabella 3.3, è indicata la velocità di rotazione massima che ciascun giunto può assumere.

Tabella 3.3: Velocità assi Comau Smart-5 Six

Nome Asse	Velocità
Asse 1	140°/s
Asse 2	160°/s
Asse 3	170°/s
Asse 4	450°/s
Asse 5	375°/s
Asse 6	550°/s

Il Comau Smart-5 Six non può essere fissato direttamente al pavimento a causa delle notevoli sollecitazioni scaricate a terra dal robot stesso: per il suo fissaggio è necessario l'utilizzo di una piastra in acciaio interposta fra il pavimento e il robot. Con questa configurazione, l'area operativa di sicurezza del robot è quelle evidenziata in rosso nella figura 3.2. L'installazione del robot può essere fatta, oltre che a terra, anche su un piano inclinato: in questo caso bisogna però porre attenzione ai limiti nei movimenti dei giunti, che ovviamente non corrispondono più a quelli indicati nella tabella 3.2.

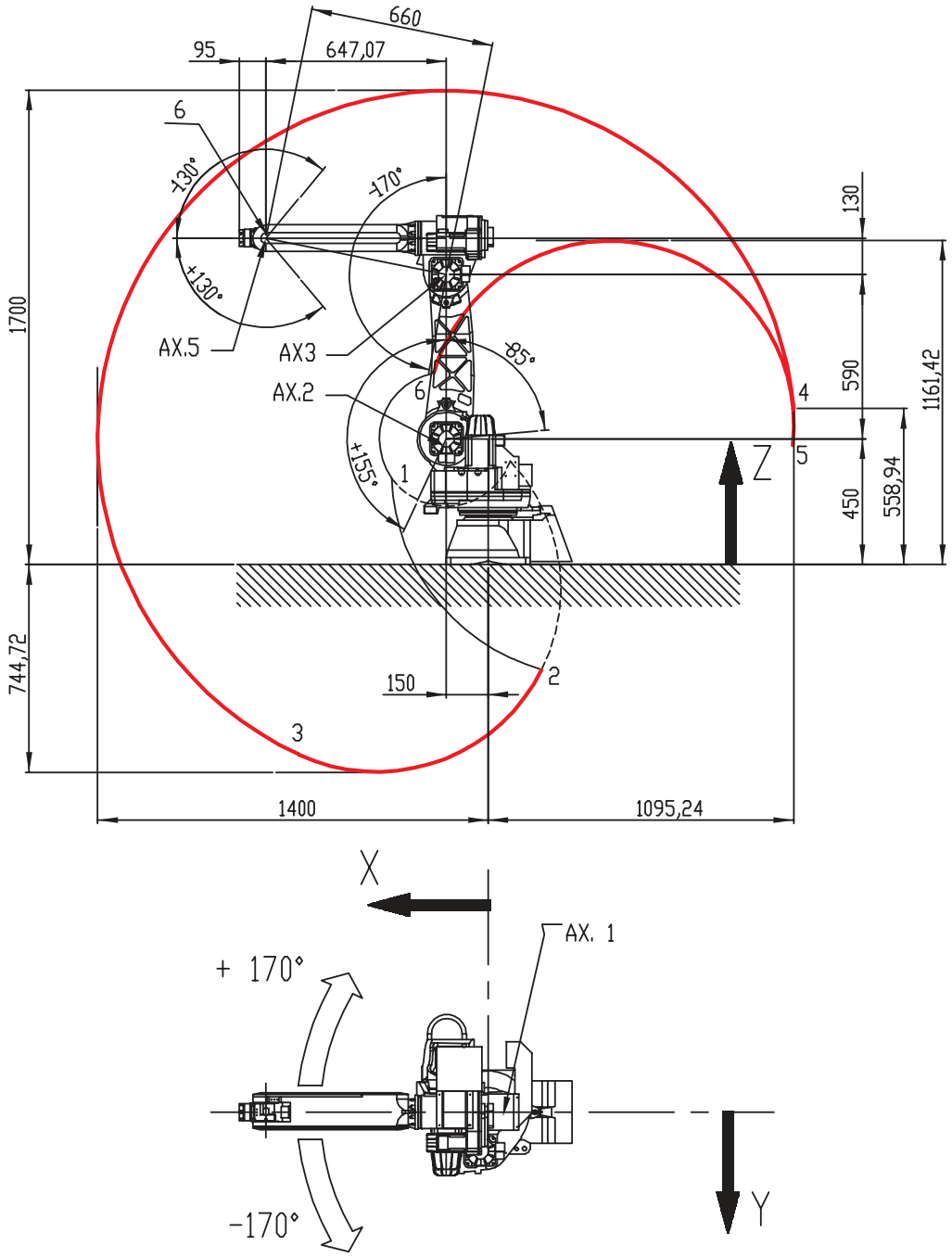


Figura 3.2: Area Operativa (linea rossa) Comau Smart-5 Six. Sono inoltre presenti le dimensioni geometriche del robot e le relazioni fra alcuni suoi componenti. (disegno prelevato dal documento *Smart5 SiX Technical Specifications*, disponibile nella sezione download del sito Comau.)

4 | Comau Model

In questo capitolo sarà descritto il modello tridimensionale del Comau Smart-5 Six e come è stato inserito nei software *rViz* e *Gazebo*. Come si potrà verificare, il modello è composto da 7 componenti: le loro caratteristiche geometriche sono fedeli a quelle riportate nel capitolo 3 e, quindi, a quelle reali mentre le informazioni relative alla massa, alla dinamica e cinematica non sono quelle reali in quanto tali dati non sono liberamente disponibili. Per aggirare tale lacuna, questi valori sono stati calcolati attraverso l'utilizzo del software *3D SolidWorks*[1]

4.1 Componenti e loro caratteristiche

Il Comau Smart-5 Sixè stato suddiviso in 7 componenti, scelta fatta per i seguenti motivi:

1. poiché il robot manipolatore ha 6 gradi di libertà, è stata ritenuto sufficiente associare a ciascun grado di libertà la relativa componente che può muovere;

2. nonostante alcuni componenti non siano strettamente legati ai movimenti del robot (come per esempio i motori), ulteriori suddivisioni sono state scartate in quanto avrebbero complicato il modello del robot senza poter trarre alcun beneficio reale in fase di simulazione.

4.2 Descrizione URDF

In questa sezione verranno elencate tutte le singole componenti del modello tridimensionale e, per ciascuna di esse, saranno documentate tutte le caratteristiche fisiche e geometriche.

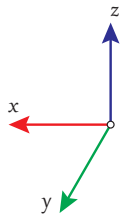
4.2.1 Sistema di riferimento

Prima di iniziare con la descrizione di ogni componente, è doveroso citare il sistema di riferimento utilizzato: la scelta di tale sistema è stata fatta seguendo li linee guida riportate in figura 3.2 e, quindi, gli assi risultano essere quelli dell'immagine 4.1.

Tutte le descrizioni geometriche, dinamiche e cinematiche devono perciò essere considerate rispettando questo sistema di riferimento. Il modello creato, dal

[1] http://www.solidworks.it/

Figura 4.1: Sistema di riferimento Comau Smart-5 Six



punto di vista geometrico, è in scala 1 : 1000 con quello reale: tale scelta è stata fatta per gestire più facilmente il robot simulato all'interno di *rViz* e *Gazebo*.

4.2.2 Componenti del modello 3D e descrizione URDF

Come descritto precedentemente, il modello tridimensionale del robot simulato è composto da 7 componenti; i nomi assegnati alle singole parti seguono la numerazione degli assi associati a ciascuno pezzo (tabella 3.2) e sono quindi:

1. `Comau_Base;`
2. `Comau_axes1;`
3. `Comau_axes2;`
4. `Comau_axes3;`
5. `Comau_axes4;`
6. `Comau_axes5;`
7. `Comau_axes6;`

Comau_Base Il primo pezzo del Comau Smart-5 Six è la base e si tratta di un pezzo fisso senza alcuna capacità di movimento.

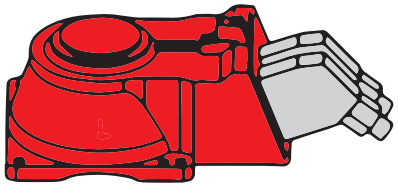


Figura 4.2: Comau_Base

Comau_Base	
Origine (m) (x, y, z)	$\begin{pmatrix} -0.10030 & 0.00262 & 0.08516 \end{pmatrix}$
Massa (Kg)	125.01405
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 0.81984 & -0.08140 & -0.08589 \\ -0.08140 & 2.56616 & 0.00694 \\ -0.08589 & 0.00694 & 2.84953 \end{pmatrix}$

Comau_axes1 Questo pezzo ha la possibilità di ruotare attorno al proprio asse z. Le sue caratteristiche sono:

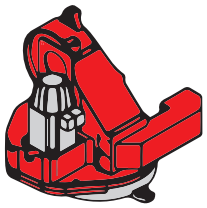


Figura 4.3: Comau_axes1

Comau_axes1	
Origine (m) (x, y, z)	$\begin{pmatrix} 0.03242 & 0.01778 & 0.12409 \end{pmatrix}$
Massa (Kg)	151.43713
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 2.46806 & 0.21183 & 0.33903 \\ 0.21183 & 2.51812 & -0.00049 \\ 0.33903 & -0.00049 & 2.54467 \end{pmatrix}$

Comau_axes2 Questo componente ha la possibilità di ruotare attorno al proprio asse y. Le sue caratteristiche sono:

Figura 4.4: Comau_axes2

Comau_axes2	
Origine (m) (x, y, z)	$\begin{pmatrix} -0.00776 & 0.01741 & 0.18315 \end{pmatrix}$
Massa (Kg)	49.71935
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 3.14612 & 0.04427 & -0.04419 \\ 0.04427 & 2.31232 & -0.80972 \\ -0.04419 & -0.80972 & 1.04069 \end{pmatrix}$

Comau_axes3 Il giunto 3 ha la possibilità di ruotare attorno al proprio asse y. Le sue caratteristiche sono:

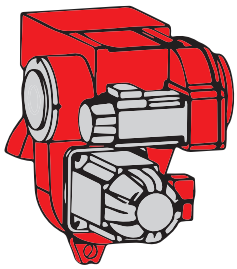


Figura 4.5: Comau_axes3

Comau_axes3			
Origine (m) (x, y, z)	$\begin{pmatrix} -0.02847 & -0.02463 & 0.09430 \end{pmatrix}$		
Massa (Kg)		82.66447	
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 1.25225 & 0.03071 & -0.11594 \\ 0.03071 & 0.75277 & -0.24169 \\ -0.11594 & -0.24169 & 1.01901 \end{pmatrix}$		

Comau_axes4 Il giunto 4 ha la possibilità di ruotare attorno al proprio asse x. Le sue caratteristiche sono:

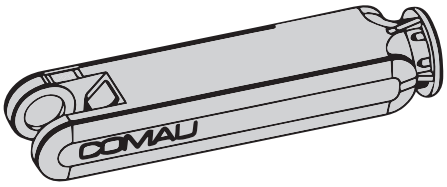


Figura 4.6: Comau_axes4

Comau_axes4			
Origine (m) (x, y, z)	$\begin{pmatrix} 0.26341 & -0.00244 & -0.00002 \end{pmatrix}$		
Massa (Kg)		50.29031	
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 0.10718 & -0.00597 & -0.00015 \\ -0.00597 & 1.29410 & 0.00004 \\ -0.00015 & 0.00004 & 1.32930 \end{pmatrix}$		

Comau_axes5 Questo componente ha la possibilità di ruotare attorno al proprio asse y. Le sue caratteristiche sono:

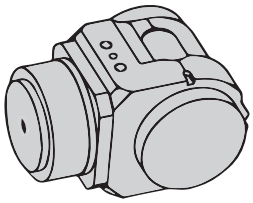


Figura 4.7: Comau_axes5

Comau_axes5	
Origine (m) (x, y, z)	$\begin{pmatrix} 0.02065 & 0.00074 & -0.00007 \end{pmatrix}$
Massa (Kg)	3.65233
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 0.00274 & -0.00006 & 0.00000 \\ -0.00006 & 0.00429 & 0.00000 \\ 0.00000 & 0.00000 & 0.00473 \end{pmatrix}$

Comau_axes6 L'ultimo giunto ha la possibilità di ruotare attorno al proprio asse x. Le sue caratteristiche sono:

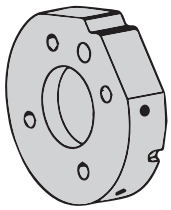


Figura 4.8: Comau_axes6

Comau_axes6	
Origine (m) (x, y, z)	$\begin{pmatrix} -0.00746 & -0.00003 & -0.00067 \end{pmatrix}$
Massa (Kg)	0.26849
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 0.00012 & 0.00000 & 0.00000 \\ 0.00000 & 0.00006 & 0.00000 \\ 0.00000 & 0.00000 & 0.00007 \end{pmatrix}$

Ciascun componente è descritto attraverso il formato *URDF*: per tale descrizione è sufficiente inserire i dati elencati utilizzando l'opportuna sintassi prevista dal formato stesso. Oltre alle specifiche tecniche, è doveroso inserire anche il modello grafico del singolo pezzo, motivo per il quale è stata ricavata, per ogni componente, una mesh in formato *Collada* (`.dae`) dal modello grafico del Comau

Smart-5 Six (modello grafico scaricabile gratuitamente dal sito[2] della *Comau*). La scelta di tale formato è motivata dal fatto che entrambi i programmi di *ROS* utilizzati (*Gazebo* e *rViz*) supportano tutte le features previste dallo stesso, come per esempio i colori dei poligoni che compongono la mesh.

Nel codice sottostante, è descritto il *link* `Comau_axes_1` nel formato *URDF*; tutti gli altri *link* seguono la medesima sintassi variando ovviamente solo i valori.

Descrizione URDF `Comau_Axes1`

```
<link name="axes_1">
   <visual>
      <geometry>
      <mesh
         filename="package://comau_model/meshes/dae/axes_1.dae"
         scale="0.001 0.001 0.001"
         />
      </geometry>
   </visual>

   <inertial>
      <origin xyz="0.03242 0.01778 0.12409" rpy="0 0 0"/>
      <mass value="151.43713"/>
      <inertia
         ixx="2.46806" ixy="0.21183" ixz="0.33903"
         iyy="2.51812" iyz="-0.00049"
         izz="2.54467"/>
   </inertial>

   <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
      <mesh
         filename="package://comau_model/meshes/dae/axes_1.dae"
         scale="0.001 0.001 0.001"
         />
      </geometry>
   </collision>
</link>
```

[2] http://www.comau.com/ita/download/Pages/Download.aspx

4.2.3 Giunti del modello e descrizione URDF

In questa sezione verrà descritto il modello *URDF* e le dipendenze geometriche presenti fra i diversi componenti del robot, ovvero i vari giunti. Come descritto in dettaglio nella sezione 2.5.2, ciascun giunto è identificato univocamente dal nome e, per ognuno di essi, deve essere specificata la relazione che vi è fra i link padre e figlio, l'origine del giunto rispetto al link padre, il tipo di giunto e le relative caratteristiche di traslazione o rotazione.

Facendo seguito a quanto esposto precedentemente, il Comau Smart-5 Six è stato suddiviso in 7 pezzi; di conseguenza, per la composizione del modello simulato è necessaria la creazione di $7 - 1 = 6$ giunti. Tutti i giunti sono di tipo *revolute*, poiché ciascun componente ha la sola capacità di ruotare attorno ad uno specifico asse e per un intervallo di valori limitato (vedere tabella 3.2). I nomi dei 6 giunti sono:

1. `joint1`;
2. `joint2`;
3. `joint3`;
4. `joint4`;
5. `joint5`;
6. `joint6`;

Per ciascuno di essi deve essere specificata una relazione fra il link padre e il link figlio: ciò implica che la descrizione finale dei giunti abbia una struttura simile a quella di un albero; in questo caso, dato che ciascun nodo padre ha solamente un unico figlio, la struttura è più somigliante ad una lista. Le relazioni presenti fra i vari giunti sono quelle indicate in tabella 4.1: nella colonna Rot. è specificato l'asse intorno a cui il giunto può ruotare.

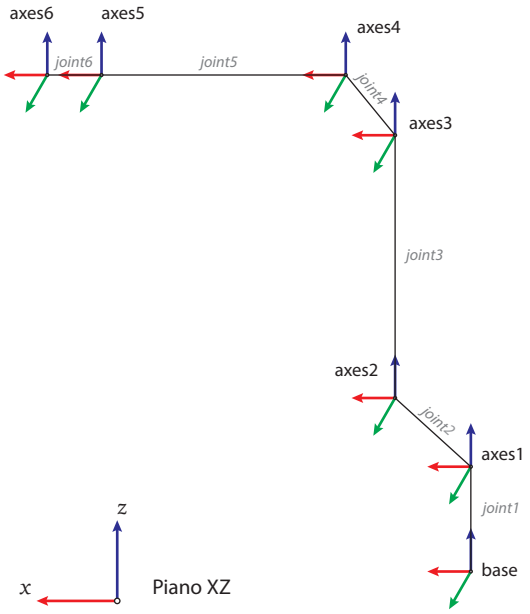
Joint	Padre	Figlio	Rot.	Origine (x, y, z)
`joint1`	`Comau_Base`	`Comau_axes1`	z	$(0 \quad 0 \quad 0.20140)$
`joint2`	`Comau_axes1`	`Comau_axes2`	y	$(0.150 \quad -0.090 \quad 0.2486)$
`joint3`	`Comau_axes2`	`Comau_axes3`	y	$(0 \quad 0.107 \quad 0.590)$
`joint4`	`Comau_axes3`	`Comau_axes4`	x	$(0.100 \quad -0.017 \quad 0.130)$
`joint5`	`Comau_axes4`	`Comau_axes5`	y	$(0.54707 \quad 0 \quad 0)$
`joint6`	`Comau_axes5`	`Comau_axes6`	x	$(0.095 \quad 0 \quad 0)$

Tabella 4.1: Giunti: relazioni link padre-figlio e origine

I valori relativi alle coordinate dell'origine di ogni giunto seguono fedelmente i dati del Comau Smart-5 Six (presenti nella figura 3.2), con l'unica accortezza che ciascun valore è stato scalato di un fattore 1000 per rispettare il sistema di riferimento utilizzato (vedere 4.2.1).

Seguendo le relazioni riportate in tabella 4.1, la struttura generata risulta essere quella di figura 4.9.

Figura 4.9: Configurazione giunti Comau Smart-5 Six (visualizzazione grafica sul piano XZ per maggiore chiarezza)



Nel codice sottostante è descritto il giunto `joint_1` nel formato *URDF*; il codice per la descrizione dei *joint* restanti è analogo a quello riportato, l'unica cosa che varia sono i valori per la rappresentazione di ciascun giunto.

Descrizione URDF `joint_1`

```
<joint name="joint_1" type="revolute">
   <parent link="base_comau"/>
   <child link="axes_1"/>
   <origin xyz="0 0 0.20140" rpy="0 0 0"/>
   <axis xyz="0 0 -1"/>
   <limit lower="-2.97" upper="2.97" effort="0" velocity="2.44"/>
</joint>
```

4.3 Simulazione Modello

In questa sezione verrà esposto come il modello del robot, descritto nel formato *URDF*, è raffigurato all'interno del visualizzatore *rViz* e del simulatore *Gazebo*.

4.3.1 rViz

Per visualizzare la mesh tridimensionale del robot in *rViz* è sufficiente aver descritto in modo corretto il robot nel formato *URDF*. Per caricare il modello nel

visualizzatore è stato creato un file `.launch` in modo tale da poter sfruttare il comando `roslaunch` presente all'interno di ROS. Il file di launch è il seguente:

```
1<launch>
2   <arg name="model" value="$(find comau_model)/robot/comau_dae.urdf" />
3   <arg name="gui" default="True" />
4   <arg name="rviz_settings" value="$(find comau_model)/rviz.vcg" />
5
6   <param name="robot_description" textfile="$(arg model)" />
7   <param name="use_gui" value="$(arg gui)"/>
8
9   <node name="joint_state_publisher" pkg="joint_state_publisher" type="
        joint_state_publisher"></node>
10  <node name="robot_state_publisher" pkg="robot_state_publisher" type="
        state_publisher" />
11
12  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rviz_settings)
        "/>
13</launch>
```

Le righe $2 - 4$ sono utilizzate per settare alcuni parametri quali il nome del file *URDF* e il file delle impostazioni per *rViz*. Le righe $6 - 7$ servono per impostare alcuni parametri di avvio dei nodi, i quali vengono poi avviati con le successive righe $9 - 12$.

4.3.2 Gazebo

Gazebo si occupa di simulare la dinamica dei movimenti di un robot, motivo per il quale è stato necessario lo sviluppo di un plugin specifico del Comau Smart-5 Six per questo software: durante il movimento di ciascun giunto del robot, questo plugin deve fornire al simulatore i valori relativi alla velocità e alla forza con cui il giunto si sposta.

Gazebo ha la funzionalità di simulare anche la fisica degli oggetti presenti nell'ambiente simulato: per mantenere costantemente in equilibrio il robot è stato necessario modificare la sua descrizione *URDF*; come accennato nella descrizione tecnica del robot che si può consultare nel documento [7] e scaricabile dal sito della Comau (link presente nel capitolo 3), è previsto il fissaggio al pavimento del braccio manipolatore attraverso l'aggiunta di una piastra in acciaio posta fra il pavimento e il Comau Smart-5 Six.

Nelle seguenti sezioni verranno descritti con maggior dettaglio il funzionamento del plugin e la modifica fatta nella descrizione *URDF*.

Descrizione URDF

Rispetto al modello descritto nel capitolo 4.2.3, sono stati aggiunti un nuovo *link* e un nuovo *joint*: il primo rappresenta la piastra in acciaio utilizzata per il fissaggio del robot al pavimento, il secondo ha lo scopo di unire il link aggiunto al link `Comau_Base` del robot. Il nuovo link è identificato dal nome `base_link` ed è

rappresentato da un parallelepipedo di dimensioni (m) $(w, h, d) = (0.1, 0.08, 0.01)$. I suoi parametri fisici sono:

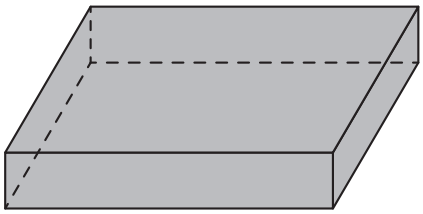


Figura 4.10: base_link

Base_link	
Origine (m) (x, y, z)	$\begin{pmatrix} 0 & 0 & 0.05 \end{pmatrix}$
Massa (Kg)	1000
Matrice Inerzia (Kg·m^2)	$\begin{pmatrix} 0.541666 & 0.00000 & 0.00000 \\ 0.00000 & 0.841666 & 0.00000 \\ 0.00000 & 0.00000 & 1.366666 \end{pmatrix}$

Il valore della massa garantisce l'equilibrio e la stabilità del robot simulato in tutte le possibili posizioni dei giunti. I numeri presenti nella matrice d'inerzia, invece, sono stati calcolati attraverso le formule relative ai momenti di inerzia di un parallelepipedo: i momenti sono stati calcolati rispetto all'asse (x, y, z) posizionato nel centro di massa del parallelepipedo stesso. Le formule sono:

$$I_{xx} = \frac{1}{12} m(h^2 + d^2) = \frac{1000}{12}(0.08^2 + 0.01^2) = 0.541666$$
$$I_{yy} = \frac{1}{12} m(w^2 + d^2) = \frac{1000}{12}(0.1^2 + 0.01^2) = 0.841666$$
$$I_{zz} = \frac{1}{12} m(w^2 + h^2) = \frac{1000}{12}(0.1^2 + 0.08^2) = 1.366666$$

Gli altri valori della matrice sono pari a 0 in quanto l'asse di inerzia è posizionato nel centro di massa e i suoi assi sono paralleli alle facce del parallelepipedo.

Per quanto concerne il *joint*, il suo unico scopo è quello di unire il `base_link` con il link `Comau_Base`; il giunto è, quindi, di tipo *fixed* e non è soggetto ad alcun movimento. I parametri del giunto sono:

Joint	Padre	Figlio	Rot.	Origine (x, y, z)
`joint_fixed`	`base_link`	`Comau_Base`	$\emptyset$	$\begin{pmatrix} 0 & 0 & 0.1 \end{pmatrix}$

Tabella 4.2: Impostazioni joint_fixed

Perciò, la struttura finale del robot simulato varia da quella riportata in figura 4.9 solo per l'aggiunta del `base_link` e del giunto `joint_fixed`.

Plugin

Per inserire il modello all'interno dell'ambiente di *Gazebo*, è stato creato un plugin la cui principale funzione è quella di trasmettere al simulatore i valori della velocità e della forza di ciascun giunto.

Gazebo permette la creazione di 4 tipi di plugin: a) World, b) Model, c) Sensor, d) System. In questo caso si tratta di un Model plugin in quanto l'obiettivo è quello di controllare la fisica dei movimenti del modello del robot. Per la creazione di tale plugin è necessario estendere la classe `ModelPlugin` e implementare il metodo `Load`.

L'implementazione del metodo `Load` è la seguente:

Metodo `Load`

```
1 void ComauGazeboPlugin::Load(
    physics::ModelPtr _parent, sdf::ElementPtr _sdf){
3
    /* ...
5    * load joints in a joint_map
     */
7
    cmd_angles_sub_ = rosnode_->subscribe<sensor_msgs::JointState>(
9      "joint_states", 1, &ComauGazeboPlugin::cmdCallback, this);
    cmd_pgain_sub_ = rosnode_->subscribe<std_msgs::Float64>(
11     "comau/pgain", 1, &ComauGazeboPlugin::cmdCallbackPgain, this);
    cmd_dgain_sub_ = rosnode_->subscribe<std_msgs::Float64>(
13     "comau/dgain", 1, &ComauGazeboPlugin::cmdCallbackDgain, this);

15  updateConnection = event::Events::ConnectWorldUpdateStart(boost::bind(
        &ComauGazeboPlugin::OnUpdate, this));
  }
```

Questo metodo viene chiamato da *Gazebo* all'avvio del plugin e si occupa di caricare tutti i *joints* del robot all'interno di una hash map, la quale utilizza una coppia (intero, stringa) per associare al nome di ciascun giunto un intero, il quale viene successivamente impiegato come indice per un vettore di `physics::JointPtr` al fine di controllare i giunti del robot.

Nelle righe 8-13 sono registrate 3 callback: la prima è utilizzata per inviare i valori effettivi dei giunti attraverso il topic `joint_states`, le altre due, invece, servono per modificare in tempo reale i valori del PID usato per controllare i giunti (con i topic `comau/pgain` e `comau/dgain`).

Nell'ultima riga viene registrato il metodo `OnUpdate`, utilizzato dal simulatore per aggiornare lo stato del robot.

Il metodo `OnUpdate` è impiegato per calcolare in tempo reale la velocità e la forza da impartire a ciascun giunto sfruttando la posizione e la velocità attuale.

Per renderlo possibile, viene adoperato un controllore PID la cui componente proporzionale è presente nella variabile `pgain` mentre quella derivativa si trova in `dgain`.

Il codice del metodo `OnUpdate` è il seguente:

Metodo `Load`

```
void ComauGazeboPlugin::OnUpdate() {
   for (unsigned int i = 0; i < joints_.size(); ++i) {
      physics::JointPtr joint = joints_[i];

      double current_velocity = joint->GetVelocity(0);
      double current_angle = joint->GetAngle(0).GetAsRadian();
      double damping_force = dgain_ * (0 - current_velocity);
      double diff_force = pgain_ * (cmd_[i] - current_angle);
      double effort_command = diff_force + damping_force;

      joint->SetForce(0, effort_command);
      joint->SetVelocity(0,effort_command);
   }
}
```

5 | Comau Controller

In questo capitolo verrà descritto il software sviluppato per controllare il robot attraverso una telecamera RGB-D; inoltre, verranno descritte altre funzionalità secondarie

5.1 Funzionalità

La funzionalità principale implementata nel Comau controller è quella di gestire in tempo reale la posizione dei giunti del robot attraverso i movimenti fatti da una persona; tuttavia non è possibile controllare tutti i giunti attraverso i dati RGB-D, cosa che verrà analizzata più nel dettaglio in un secondo momento. L'operazione di controllo dei giunti attraverso alcune azioni compiute da una persona viene definita *Mapping*; oltre a questa attività, all'interno del software sono state implementate ulteriori funzioni secondarie, tra le quali:

1. salvataggio dei movimenti dei giunti;
2. riproduzione dei movimenti dei giunti;
3. avvio del controller attraverso un'interfaccia testuale unix-style.

Durante lo sviluppo del controller è stata posta molta attenzione all'architettura del software: l'obiettivo principale è stato quello di sviluppare il codice in moduli attraverso l'uso di interfacce. L'utilizzo delle interfacce permette di definire le funzionalità che il software deve avere mantenendo tuttavia separata l'implementazione del corrispettivo compito. Questa caratteristica è stata di rilevante importanza in quanto ha portato ai seguenti vantaggi: sviluppo di moduli dipendenti dalle corrispettive interfacce ma non dalla loro implementazione e sviluppo di più implementazioni della medesima interfaccia (elemento utile in fase di test per valutare la miglior realizzazione). I moduli del software sono i seguenti:

- modulo per l'attività di mapping dei giunti;
- modulo per la gestione delle collisioni (collision avoidance);
- modulo per l'invio dei valori dei giunti al robot simulato;
- modulo per le operazione di I/O su file.
- interfaccia utente per l'avvio del controller;

Prima di procedere con la descrizione dei singoli moduli, saranno analizzate le classi base del controller, essendo di fatto le classi fondamentali dell'intero software. In seguito saranno esposte nel dettaglio tutte le funzionalità del controller e la modalità di realizzazione delle stesse.

5.2 Mapping

Il robot può essere delineato come un sistema di corpi rigidi legati fra loro attraverso un certo numero di giunti; l'operazione di mapping consiste nel controllare l'orientamento e la posizione di questi giunti in tempo reale attraverso determinate azioni compiute dall'utente. In questo progetto alcuni giunti sono gestiti attraverso dei semplici movimenti eseguiti da una persona.

Nell'operazione di mapping è stata utilizzata quella teoria che analizza la relazione tra la configurazione (posizione ed orientamento) di un segmento e gli angoli dei giunti: tale teoria si chiama *Cinematica*. L'analisi di questa relazione può essere fatta in due modi:

- cinematica diretta: metodo che permette di trovare l'orientamento e la posizione dei *link* del robot in funzione dei valori degli angoli dei vari giunti. Questo procedimento è utilizzato nel campo della robotica per vari scopi, quali calcolare il centro di gravità, conoscere la configurazione dei corpi rigidi o per individuare le collisioni con l'ambiente circostante al robot.

- cinematica inversa: è la ricerca dei valori degli angoli dei giunti che corrispondono all'orientazione e alla posizione di un *link* nello spazio. Questa tecnica può essere utilizzata quando si adopera un sistema di visione per il *motion capture*: tale sistema, infatti, fornisce una descrizione geometrica del corpo umano e, con l'impiego della cinematica inversa, è possibile determinare il valore degli angoli fra i vari arti partendo dalla posizione degli arti stessi.

Nel progetto sviluppato sono stati utilizzati sia il concetto di cinematica diretta che di quello cinematica inversa: il primo è stato applicato per evitare di far collidere il robot manipolatore con l'ambiente circostante, mentre il secondo è stato utilizzato per assegnare i valori degli angoli ad alcuni giunti del robot attraverso i movimenti di una persona, movimenti registrati da una telecamera con sensore RGB-D. Non è stato invece possibile controllare tutti i giunti attraverso i movimenti della persona per due motivi:

1. facilità di utilizzo: l'obiettivo principale è stato quello di rendere il più familiare possibile il controllo del robot; di conseguenza, si è cercato di governare gli spostamenti del Comau Smart-5 Six con il movimento naturale del proprio corpo. Non è stato possibile controllare tutti e 6 i giunti del robot con i dati RGB-D in quanto avrebbe richiesto di effettuare movimenti scomodi rendendo così il controllo del robot poco user-friendly;

2. stabilità dei movimenti: il secondo obiettivo nel controllo del robot è stato quello di gestire i suoi movimenti in modo stabile, ovvero evitando scatti improvvisi e tali per cui il robot fosse "fuori controllo". Poiché i movimenti della persona sono catturati dal sensore RGB-D, è chiaro che la robustezza nella gestione del braccio manipolatore dipende anche dalla stabilità dei valori provenienti dal sensore: per questo motivo, il sensore RGB-D è stato utilizzato come fonte di input per i movimenti per garantire valori piuttosto stabili e sicuri.

Verrà ora illustrato come è stato implementato il mapping di ogni singolo giunto. Per i primi 3 giunti (`joint1`, `joint2`, `joint3`), è stata utilizzata la telecamera con dati RGB-D utilizzando la tecnica della cinematica inversa per assegnare in tempo reale il valore ai singoli *joints*. Per quelli restanti (`joint4`, `joint5`, `joint6`), invece, il controllo avviene attraverso una periferica, come esposto nelle pagine seguenti.

5.2.1 Joint1, Joint2, Joint3

Come è stato introdotto precedentemente, a questi giunti è stata applicata la teoria della cinematica inversa, impostando quindi il valore dell'angolo relativo al giunto in base a posizione e orientamento dei *link* nello spazio. In questo frangente, i *link* presi in esame sono quelli relativi ad una persona (descritti in figura 2.2): dalla posizione nello spazio di alcuni dei *link* riportati, si ricava l'angolo che verrà mappato nei valori dei giunti del robot.

Prima di analizzare nel dettaglio come ogni angolo verrà assegnato a ciascuno dei 3 giunti, è doveroso porre particolare attenzione ai sistemi di riferimento utilizzati. Come si può vedere dalla figura 5.1, il sistema di riferimento utilizzato per descrivere la posizione di un *link* di una persona è diverso da quello utilizzato per descrivere la posizione di un *link* del Comau Smart-5 Six: poiché gli angoli che si dovranno mappare devono essere coerenti con il sistema di riferimento del robot, i calcoli mostrati nelle pagine seguenti saranno anch'essi coerenti con lo stesso sistema.

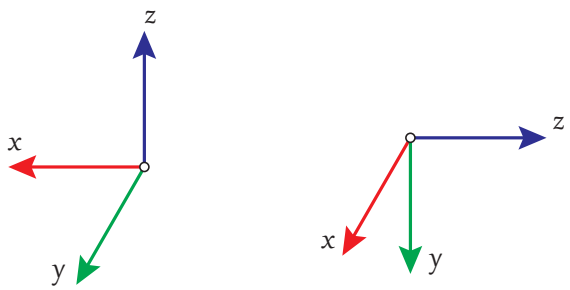


Figura 5.1: Sistemi di riferimento. Comau Smart-5 Six a sinistra, Skeleton a destra

Il passaggio dal sistema di riferimento dello skeleton a quello del robot è realizzato sfruttando queste semplici analogie:

Tabella 5.1: Sistemi di riferimento

Comau Smart-5 Six	Skeleton
x	$-z$
y	x
z	$-y$

Ora verrà descritto in dettaglio come ciascun giunto è stato mappato attraverso i movimenti dei *link* di una persona, movimenti catturati sfruttando le informazioni provenienti dalla telecamera RGB-D.

Joint1

Joint1 permette di effettuare una rotazione attorno al proprio asse z: questo movimento porta il braccio manipolatore a ruotare attorno al suo asse verticale; di conseguenza, il giunto si occupa di posizionare il Comau Smart-5 Six in una direzione prestabilita. La scelta di questa direzione viene rilevata dal sensore RGB-D osservando la posizione della mano destra rispetto al collo della persona interessata.

Siano

$$H = (x_H, y_H, z_H)$$
$$O = (x_O, y_O, z_O)$$

rispettivamente le coordinate (x, y, z) nello spazio relative ai giunti `/skeleton_1_right_hand` e `/skeleton_1_neck`; per trovare la posizione del punto H rispetto a quello O, è necessario innanzitutto trovare il vettore $\vec{v} = (x_v, y_v, z_v)$ che li unisce, considerando come origine il punto O relativo al collo. Il vettore $\vec{v}$ ha quindi come valori:

$$x_v = x_H - x_O$$
$$y_v = y_H - y_O$$
$$z_v = z_H - z_O$$

Utilizzando i valori del vettore $\vec{v}$ è possibile trovare l'angolo richiesto attraverso la proiezione del vettore stesso sul piano XY e calcolare l'angolo presente fra il vettore proiettato in questo piano e l'asse x.

Per trovare tale angolo, le operazioni da fare sono:

$$\rho_{XY} = \sqrt{x_v^2 + y_v^2}$$
$$x_v = \rho_{XY} \cos \alpha$$

Applicando la formula inversa e facendo attenzione al segno di y_v, l'angolo risultante è:

$$\alpha = \begin{cases} \arccos\left(\frac{x_v}{\rho_{XY}}\right) & \text{se } y_v \geq 0, \\ -\arccos\left(\frac{x_v}{\rho_{XY}}\right) & \text{altrimenti.} \end{cases}$$

In figura 5.2 è rappresentato graficamente l'angolo trovato.

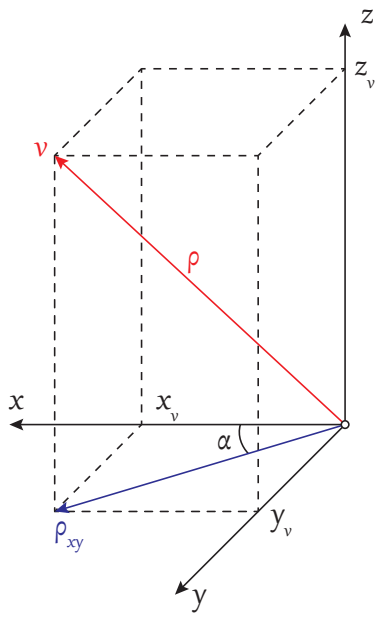


Figura 5.2: Mapping Joint1

Joint2

Questo giunto ha la possibilità di far ruotare i *link* a lui annessi attorno al proprio asse *y*: questa rotazione porta il Comau Smart-5 Six ad effettuare uno spostamento longitudinale lungo la direzione stabilita dal *Joint1*. L'eventuale avanzamento del braccio del robot è stato mappato, quindi, con lo spostamento in avanti o indietro della persona rispetto alla sua posizione iniziale. Per calcolare il possibile avanzamento della persona si procede nel seguente modo:

1. capire in quale direzione è voltata la persona;
2. calcolare l'effettivo avanzamento rispetto alla posizione della persona.

Per stabilire la direzione della persona ci si è basati sulla rilevazione della posizione delle spalle: più precisamente, è stata calcolata la posizione della spalla sinistra rispetto a quella destra e, successivamente, si è trovata la direzione della persona osservando la rotazione delle spalle rispetto alla posizione iniziale della persona stessa.

Una volta rilevata la direzione, calcolare di quanto avanza la persona risulta piuttosto facile: è sufficiente, a questo punto, proiettare il vettore che indica l'avanzamento lungo la direzione precedentemente stabilita e calcolare di quanto la persona si è spostata rispetto alla sua posizione iniziale. Lo spostamento della persona viene rilevato confrontando la posizione del giunto `/skeleton_1_neck` rispetto alla posizione iniziale del giunto stesso.

Verranno ora illustrati nel dettaglio tutti i calcoli effettuati.
Siano:

$$L = (x_L, y_L, z_L)$$
$$R = (x_R, y_R, z_R)$$
$$N = (x_N, y_N, z_N)$$

rispettivamente le posizioni nello spazio dei giunti:

- `/skeleton_1_left_shoulder` (spalla sinistra);
- `/skeleton_1_right_shoulder` (spalla destra);
- `/skeleton_1_neck` (collo).

è necessario per prima cosa trovare la posizione della spalla sinistra rispetto a quella destra; come nel *Joint1*, bisogna trovare il vettore che lega questi due giunti. Sia $\vec{v} = (x_v, y_v, z_v)$ il vettore, le sue coordinate sono:

$$x_v = x_L - x_R$$
$$y_v = y_L - y_R$$
$$z_v = z_L - z_R$$

Successivamente, è necessario proiettare il vettore relativo all'avanzamento della persona nel sistema di riferimento stabilito dall'orientamento della persona. Per effettuare questa operazione è necessario, però, effettuare prima i seguenti passi: innanzi tutto bisogna capire quanto ampio è l'angolo relativo alla rotazione della persona attorno all'asse z rispetto alla sua posizione iniziale.

Sia R_0 il sistema di riferimento iniziale della persona, i cui vettori sono:

$$\vec{x_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \vec{y_0} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \vec{z_0} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.1}$$

il nuovo sistema di riferimento R_1, relativo alla posizione ruotata della persona, avrà di conseguenza i seguenti vettori:

$$\vec{x_1} = \vec{y_1} \times \vec{z_1} \qquad \vec{y_1} = \begin{pmatrix} \frac{x_v}{\|\vec{v}\|} \\ \frac{y_v}{\|\vec{v}\|} \\ 0 \end{pmatrix} \qquad \vec{z_1} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

da cui si possono ricavare i nuovi vettori che risultano essere:

$$\vec{x_1} = \begin{pmatrix} \frac{y_v}{\|\vec{v}\|} \\ -\frac{x_v}{\|\vec{v}\|} \\ 0 \end{pmatrix} \qquad \vec{y_1} = \begin{pmatrix} \frac{x_v}{\|\vec{v}\|} \\ \frac{y_v}{\|\vec{v}\|} \\ 0 \end{pmatrix} \qquad \vec{z_1} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.2}$$

Come si può constatare dal sistema di riferimento 5.2, la componente z non subisce alcuna alterazione poiché la rotazione avviene attorno questo stesso asse;

la componente y invece viene allineata al vettore relativo alla posizione delle spalle e, in base alla regola della mano destra, il vettore x risulta essere il prodotto vettoriale fra il vettore y e quello z.

A questo punto risulta indispensabile capire di quanto il sistema di riferimento R_1 sia ruotato rispetto a quello canonico R_0. Per individuare tale rotazione bisogna trovare la matrice di rotazione M_{R1R0}, con la seguente procedura: a) si trova la matrice di rotazione M_{R0R1}; b) si calcola la sua matrice inversa.

La matrice M_{R0R1} è facile da ottenere poiché si tratta di una semplice differenza di rotazione rispetto alla posizione canonica; ponendo

$$\alpha = \frac{x_v}{\|\vec{v}\|} \qquad \beta = \frac{y_v}{\|\vec{v}\|}$$

la matrice M_{R0R1} è:

$$\begin{pmatrix} \beta & \alpha & 0 \\ -\alpha & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

la matrice inversa M_{R1R0} risulta quindi essere:

$$\begin{pmatrix} \frac{\beta}{\alpha^2+\beta^2} & -\frac{\alpha}{\alpha^2+\beta^2} & 0 \\ \frac{\alpha}{\alpha^2+\beta^2} & \frac{\beta}{\alpha^2+\beta^2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sfruttando questa matrice, è possibile prendere un generico punto nello spazio (con coordinate relative al sistema R_0) e ottenere le corrispondenti nuove coordinate rispetto al sistema di riferimento R_1 (vedere figura 5.3).

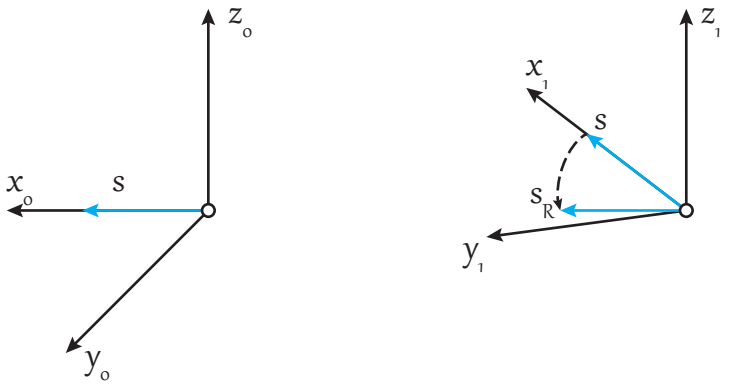


Figura 5.3: Passaggio dal sistema di riferimento R_0 a R_1

Ora è necessario valutare le coordinate del vettore relativo allo spostamento della persona nel sistema di riferimento R_1. Questo vettore è calcolato prendendo in esame la posizione del giunto del collo in due istanti di tempo diversi, quello iniziale t_0 e quello corrente t_c; lo spostamento $\vec{s}$ è facilmente ottenibile calcolando

la differenza fra la posizione iniziale $N_0 = (x_{N,0}, y_{N,0}, z_{N,0})$ del collo e quella attuale $N_c = (x_{N,c}, y_{N,c}, z_{N,c})$; il vettore risulta quindi essere:

$$\vec{s} = \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \begin{pmatrix} x_{N,c} - x_{N,0} \\ y_{N,c} - y_{N,0} \\ z_{N,c} - z_{N,0} \end{pmatrix}$$

Per trovare l'effettiva posizione della persona $\vec{s}$ nel sistema di riferimento R_1, è sufficiente calcolare il seguente prodotto:

$$\vec{s_R} = M_{R1R0} \cdot \vec{s}$$

Per ottenere l'effettivo spostamento della persona lungo la sua direzione, è sufficiente considerare la proiezione del vettore $\vec{s_R}$ lungo l'asse x. A questo punto, è necessario far compiere al robot un angolo tale per cui la posizione del braccio avanzi o indietreggi del valore pari alla coordinata x: a tal scopo, nota la lunghezza del *Comau_Axes_2* (indicata con *arm1_length*), è sufficiente applicare la seguente formula trigonometrica:

$$\alpha = \arccos\left(\frac{x}{\text{arm1\_length}}\right)$$

Prima di inviare l'angolo al giunto, è necessario correggere il valore con un fattore correttivo pari a $\frac{\pi}{2}$ (questo elemento dipende dalla modalità di inserimento del giunto nel modello tridimensionale); l'angolo finale del giunto che si ottiene sarà quindi:

$$\alpha_{\text{correct}} = \frac{\pi}{2} - \alpha$$

Joint3

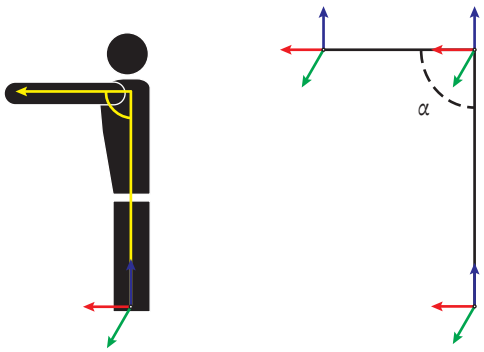
Il terzo giunto si occupa di far alzare o abbassare il braccio del robot tramite una rotazione attorno al proprio asse y. Utilizzando le informazioni provenienti dal sensore RGB-D, il controllo di questo movimento è stato fatto osservando in tempo reale la posizione della mano destra rispetto al giunto del collo e calcolando quindi l'angolo associato fra questi due giunti.

Tuttavia tale angolo non è sufficiente per garantire un corretto movimento del braccio manipolatore poiché, a differenza dei giunti precedenti, lo spostamento del *Joint3* dipende anche dall'angolo del *Joint2*. Questa circostanza è facilmente verificabile con un semplice esempio: si supponga di allungare il braccio destro in avanti in modo che il braccio formi un angolo pari a $\frac{\pi}{2}$ rispetto al corpo; a questo punto, bisogna considerare due casi:

1. il *Joint2* è fisso;
2. il *Joint2* non si trova nella posizione di riposo.

Nel primo caso, il vettore che indica la posizione della mano destra rispetto al collo è sufficiente affinché il corrispettivo movimento del braccio manipolatore sia corretto (figura 5.4).

Figura 5.4: caso 1. Joint2 a riposo



Questa condizione cambia non appena il *Joint2* viene mosso, movimento causato dall'avanzamento della persona. Come si può vedere dallo schema di figura 5.5, se non si considerasse il valore relativo all'angolo del secondo giunto, la posizione del braccio del Comau Smart-5 Six non sarebbe più coerente con quella del braccio della persona (figura centrale).

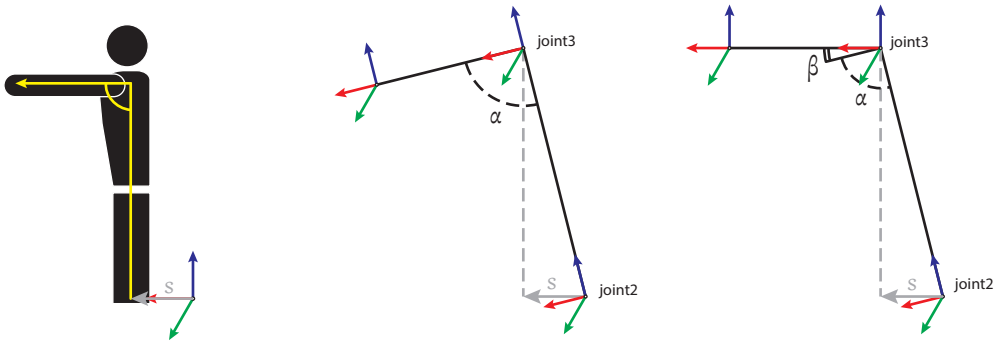


Figura 5.5: Joint2 non a riposo. A sinistra, spostamento della persona; al centro, mancata considerazione dell'angolo del secondo giunto; a destra, risoluzione del problema

Per ovviare a questo inconveniente è perciò indispensabile considerare anche l'angolo relativo al *Joint2*, come si può constatare dalla terza immagine di figura 5.5.

L'angolo utilizzato per il controllo di questo giunto corrisponde in realtà alla somma di due angoli: il primo è relativo al *Joint2* e il secondo calcolato osservando la posizione della mano destra rispetto al collo della persona. Per ottenere quest'ultimo sono stati implementati i seguenti calcoli: sia $\vec{v} = (x_v, y_v, z_v)$ il vettore che rappresenta la posizione della mano destra rispetto al collo (vettore analogo a quello utilizzato per il *Joint1*, vedere 5.2.1) si consideri la lunghezza della proiezione di tale vettore sul piano XY e quella relativa al vettore $\vec{v}$, facilmente

ottenibile con le formule

$$\|\vec{v}_{xy}\| = \sqrt{x_v^2 + y_v^2}$$
$$\|\vec{v}_{xyz}\| = \sqrt{x_v^2 + y_v^2 + z_v^2}$$

facendo attenzione al segno della coordinata z_v, il valore dell'angolo da trovare è

$$\theta = \begin{cases} \arccos\left(\frac{\|\vec{v}_{xy}\|}{\|\vec{v}_{xyz}\|}\right) & \text{se } z_v \geq 0, \\ -\arccos\left(\frac{\|\vec{v}_{xy}\|}{\|\vec{v}_{xyz}\|}\right) & \text{altrimenti.} \end{cases}$$

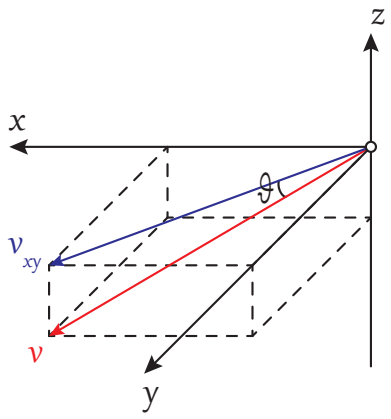


Figura 5.6: Mapping angolo θ del Joint3

Come per il *Joint2*, anche in questo caso è necessario aggiungere un fattore correttivo per far si che il movimento del braccio del Comau Smart-5 Six sia quello desiderato. Il valore dell'angolo θ_{correct} è quindi:

$$\theta_{\text{correct}} = -\left(\theta + \frac{\pi}{2}\right);$$

Facendo la somma dei due angoli, il valore inviato al giunto risulta essere:

$$\gamma = \theta_{\text{correct}} + \alpha_{\text{correct}}$$

dove il secondo angolo ha valore uguale a quello riportato alla fine della sezione 5.2.1 del *Joint2*.

5.2.2 Joint4, Joint5, Joint6

A differenza di quanto illustrato fin'ora, il metodo implementato per il controllo dei giunti *Joint4*, *Joint5* e *Joint6* è completamente differente. Innanzitutto, non viene utilizzata la telecamera RGB-D e, di conseguenza, non si agisce sugli stessi con

lo spostamento di parti del corpo della persona. Questa scelta è motivata dal fatto che, dopo aver valutato diverse possibili movimenti per ruotare i tre giunti, nessuna di esse garantiva un controllo stabile e semplice ottenendo, di fatto, movimenti scomodi e rischiando di entrare in conflitto con quelli utilizzati per gestire i *Joint1*, *Joint2* e *Joint3*. C'è da considerare anche che il tipo di rotazione di questi giunti provoca degli spostamenti del robot di natura diversa rispetto ai precedenti: infatti, mentre per i primi le tre rotazioni provocavano uno spostamento del braccio del robot nello spazio, la rotazione di questi ultimi non influenza in alcun modo la posa del Comau Smart-5 Six, tant'è che vengono utilizzati principalmente per effettuare delle operazioni meccaniche.

Per questi motivi, è stato sviluppato un metodo di controllo completamente diverso il cui obiettivo primario è stato quello di garantire un controllo di tutti e 6 i giunti del robot nel modo più comodo e naturale possibile. A tale scopo, la rotazione dei giunti avviene attraverso l'utilizzo di un semplice dispositivo esterno: il mouse, utilizzato come fonte di input data la necessità di controllare tutti e 6 i giunti con la sola mano destra.

A questo punto le operazioni da compiere risultano piuttosto semplici, selezionando come prima cosa quale dei 3 giunti controllare tramite la pressione di un pulsante del mouse:

1. *Joint4*: pressione tasto sinistro;
2. *Joint5*: pressione tasto centrale;
3. *Joint6*: pressione tasto destro.

Una volta effettuata la scelta di cui sopra, l'effettiva rotazione avviene con l'operazione di *scrolling* della rotellina del mouse:

- up: diminuzione dell'angolo di rotazione;
- down: incremento dell'angolo di rotazione.

Con questo semplice sistema le rotazioni dei giunti *Joint4*, *Joint5* e *Joint6* sono user-friendly e, quindi, facilmente gestibili.

5.3 Collision Avoidance

In questo capitolo verrà illustrato come è stato implementato il modulo del controller del Comau Smart-5 Six relativo alla gestione delle collisioni.

Prima di addentrarci nella spiegazione di come è stato sviluppato questo modulo, è doveroso delineare l'ambiente circostante in cui si trova ad operare il robot. In questo caso, si ipotizza l'inserimento all'interno un ambiente privo di ostacoli e isolato, con la certezza che non si verifichino mutamenti durante la fase operativa: non è stato perciò necessario aggiungere alcun tipo di sensore per verificare la presenza o meno di ostacoli.

Permane, tuttavia, la presenza di un ostacolo fisso che non si può eliminare: il pavimento. Il modulo risulta perciò molto semplificato dalla presenza di questo singolo ostacolo fisso, ma grazie alla modularità con cui è stato realizzato dovrebbe risultare semplice sviluppare il sistema o anche integrare moduli esterni che gestiscano situazioni più complesse, magari anche con persone che si muovono al fianco del robot stesso. Come descritto nel capitolo 3, il braccio manipolatore può essere installato sia a terra che su un piano inclinato e, a seconda di come viene fissato, il robot deve essere limitato in alcuni movimenti per evitare l'impatto con il suolo. Nell'ambiente simulato, il robot si trova installato sul pavimento mediante una piastra (vedere capitolo 4).

Per evitare la collisione con il suolo, è necessario calcolare in tempo reale la distanza dallo stesso del braccio del Comau Smart-5 Six e verificare che sia sempre superiore ad una determinata soglia. Tale calcolo è stato eseguito sfruttando la teoria della *cinematica diretta* che, come illustrato nella sezione 5.2, permette di trovare l'orientamento e la posizione dei *link* del robot in funzione degli angoli dei vari *joint*. L'altezza del braccio del robot è influenzata dal valore degli angoli di 2 dei 6 giunti del robot: i giunti che influenzano l'altezza del braccio sono: *Joint2* e *Joint3*. É quindi sufficiente applicare la teoria della *cinematica diretta* solamente a questi due giunti per ottenere istante per istante l'altezza corrente della parte più estrema del robot; oltre ai valori degli angoli, è necessario conoscere anche la lunghezza dei *link* interessati. Per semplificare i calcoli e per trovare il valore dell'effettiva altezza, è stato utilizzato il seguente schema:

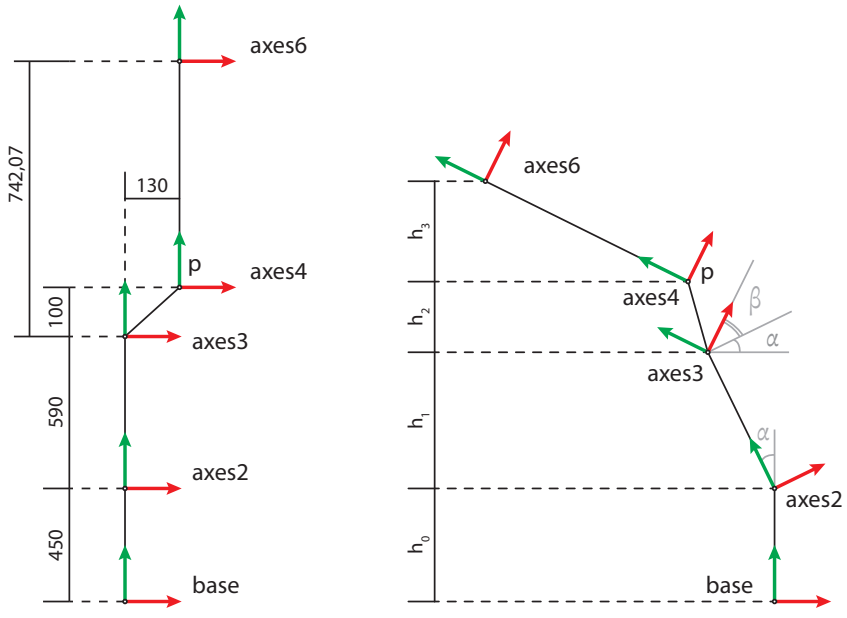


Figura 5.7: Schema Comau Smart-5 Six per la cinematica diretta. A sinistra la lunghezza dei link presi in esame; a destra è rappresentato come gli angoli influenzano l'altezza del braccio del Comau Smart-5 Six

Come illustrato nella figura 5.7, il robot è rappresentato su un piano bidimensionale: tale scelta è stata fatta per poter ignorare le coordinate superflue al fine di calcolare l'altezza del braccio, semplificando leggermente l'intero procedimento. Le lunghezze dei vari *link* sono state prelevate direttamente dalla figura 3.2: l'unica misura non presente è quella relativa al punto p, utilizzato per indicare la posizione del giunto 4 rispetto al giunto 3, le cui coordinate sono state tuttavia prelevate dal file sorgente di *SolidWorks*, scaricabile dal sito ufficiale *Comau* (capitolo 3).

Utilizzando lo schema di figura 5.7 e considerando che la distanza fra il link *base* e *axes2* rimane immutata a prescindere dai movimenti del robot, indicando con h_0 la misura, il suo valore risulta essere:

$$h_0 = 450$$

Per quanto riguarda invece gli altri link h_1 e h_3, è sufficiente effettuare la seguente operazione:

$$\begin{aligned} h_1 &= 590 \cos \alpha \\ h_3 &= (742.07 - 100) \cos (\alpha + \beta) \end{aligned}$$

Per il calcolo dell'altezza del punto p ruotato, è stata invece sfruttata la seguente matrice di rotazione M_R:

$$M_R = \begin{pmatrix} \cos (\alpha + \beta) & -\sin (\alpha + \beta) \\ \sin (\alpha + \beta) & \cos (\alpha + \beta) \end{pmatrix}$$

Quindi, per ottenere le coordinate $P_R = (x_R, y_R)$ del punto P dopo aver fatto la rotazione, è sufficiente calcolare:

$$P_R = M_R \cdot P$$

Il corrispondente valore di h_2 è quindi:

$$h_2 = y_R$$

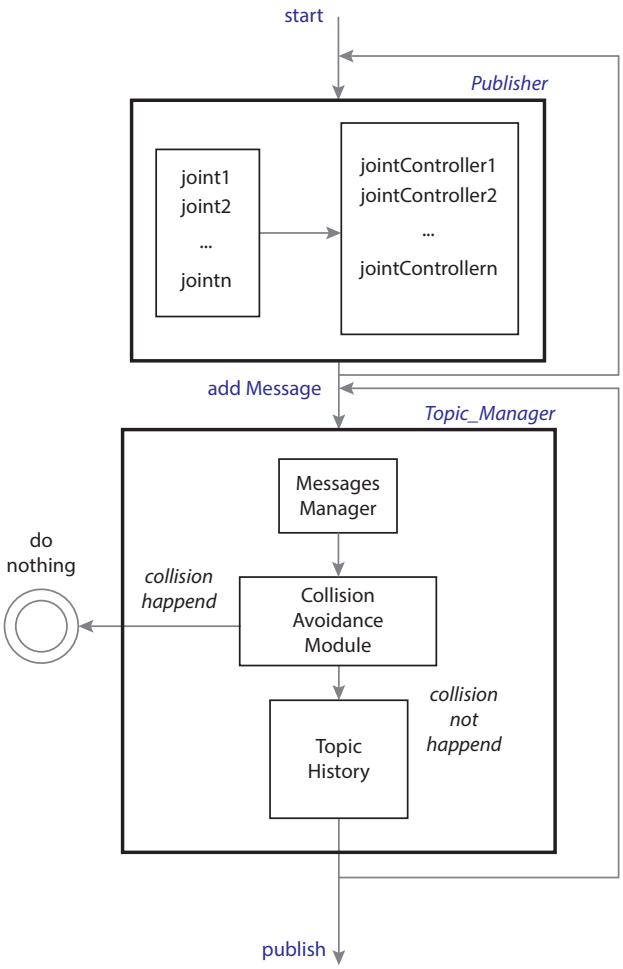
L'altezza finale h del braccio del Comau Smart-5 Six risulta essere perciò:

$$h = \sum_{j=0}^{3} h_j$$

5.4 Architettura SW

In questa sezione verranno presentati i moduli chiave del software utilizzati per il controllo del Comau Smart-5 Six; il diagramma che visualizza la struttura software è quello di figura 5.8.

Figura 5.8: Architettura Software Comau Controller



Come si può vedere dall'immagine, la struttura prevede due macroblocchi:

- *Publisher*;
- *Topic_Manager*.

Il loro funzionamento verrà ora spiegato nel dettaglio.

Publisher I compiti di questo modulo sono essenzialmente tre: 1. il primo è quello di fare da contenitore dei giunti (*joint*) del robot che si vuole controllare; 2. il secondo consiste nell'associare a ciascun giunto il relativo controller (*jointController*), la cui funzione è quella di impostare in tempo reale il valore effettivo del giunto associato; 3. infine, l'ultimo compito è quello di prelevare tutti i valori dai vari controller e inviarli al macroblocco *Topic_Manager*. Le entità coinvolte in questo modulo sono quindi: *Joint* e *JointController*. Si analizzeranno ora nel dettaglio le informazioni in esse contenute e le loro funzioni.

Ciascun *Joint* è descritto dalla seguente classe:

Joint.h

```
class Joint {
   ...
public:
   virtual std::string getName() const = 0;
   virtual float getDefaultValue() const = 0;
   virtual float getUpLimit() const = 0;
   virtual float getLowLimit() const = 0;

   virtual std::ostream & print(std::ostream &out) const = 0;
};
```

Questa classe ha lo scopo di descrivere un generico giunto al quale, come si può constatare dal codice, viene associato un nome, un valore predefinito e, qualora presente, deve essere in grado di fornire l'intervallo superiore e inferiore di valori che il giunto può assumere; Per poter stampare a terminale la descrizione stessa, viene richiesto il metodo `print()`.

L'entità *jointController* si occupa di definire il comportamento di ciascun giunto ed è stata sviluppata in due diverse tipologie: *jointController* e *jointAutoController*. La prima entità è descritta dalla classe `JointController.h`, il cui codice è il seguente:

JointController.h

```
class JointController {
   ...
public:
   virtual Joint::ConstPtr getJoint() const = 0;
   virtual void setValue(float) = 0;
};
```

La struttura di questa classe è relativamente semplice, composta solamente da due metodi: il metodo `getJoint()` restituisce il giunto che il controller si occupa di controllare; il metodo `setValue()` ha come unico scopo quello di impostare manualmente il valore di un giunto.

Il secondo controller è invece descritto dalla classe `JointAutoController.h` nella quale, a differenza della precedente, i valori di un generico giunto sono impostati in modo "automatico", ovvero attraverso l'utilizzo di un dispositivo di input esterno.

JointAutoController.h

```
class JointAutoController : public virtual JointController {
   ...
public:
   virtual float getValue() = 0;
   void getAndSetValue() {
      setValue(getValue());
   }
```

```
};
```

La classe `JointAutoController` è un'estensione della classe `JointController` e, come accennato precedentemente, varia da quest'ultima in quanto imposta i valori di un generico giunto nel caso in cui siano prelevati da una fonte esterna. Come si può notare dal codice, la classe `JointAutoController` richiede di implementare solo il metodo `getValue()`, utilizzato per prelevare il valore del giunto dal dispositivo di input: attraverso il metodo `getAndSetValue()` la classe imposta il valore effettivo del giunto chiamando il metodo `setValue()` della classe estesa.

Il *Publisher* ha il compito di unire a ciascun *Joint* il corrispettivo *JointController*; nonostante sia obbligatorio associare a ciascun giunto un solo controller, la scelta di mantenere separati questi due elementi è motivata dai seguenti fattori:

1. il primo beneficio si ha in fase di sviluppo e soprattutto durante il testing in cui è possibile valutare e confrontare diverse implementazioni di un controller per la gestione dello stesso giunto; sarà quindi sufficiente selezionare di volta in volta quale controller associare al giunto e, durante la simulazione, verificarne il comportamento;

2. il secondo vantaggio consiste nella possibilità di scegliere se controllare il giunto in modalità "manuale" (*JointController*) o "automatica" (*JointAutoController*), come accennato nel capitolo 5.1. Nel caso di riproduzione dei movimenti dei giunti, non viene associata alcuna periferica di input e i valori vengono letti da file e quindi impostati manualmente; nella modalità di salvataggio i giunti sono invece controllati da una periferica esterna ed è quindi necessario utilizzare la modalità di controllo "automatica". A seconda della modalità scelta, è sufficiente cambiare il tipo di controller del giunto;

3. il terzo elemento non porta vantaggi a livello di funzionalità, ma porta benefici nella scrittura del codice. Grazie alla separazione fra *Joint* e *JointController* vengono evitate parti di codice ridondanti: infatti, se queste due entità fossero rimaste unite, sarebbe stato necessario riscrivere tutte le caratteristiche per più controller del medesimo giunto preso in esame.

L'ultimo compito del *Publisher* è quello di inviare i valori dei giunti al *Topic_Manager* controllando che siano validi e, nel caso in cui un giunto abbia un intervallo limitato, che siano compresi all'interno dello stesso. Se il valore è in regola con le capacità del giunto viene inviato al macroblocco successivo sotto forma di messaggio formato da una coppia `(chiave, valore)` in cui la chiave corrisponde al nome del giunto (univoco) e il valore è quello validato; nel caso in cui non si verifichi questa condizione non viene inviato alcun messaggio. Ciascun messaggio inviato contiene le informazioni relative ad un singolo giunto quindi, per inviare le informazioni di n giunti, sono necessari n messaggi.

All'interno del *Publisher* non vi è alcuna indicazione riguardo la modalità di invio dei messaggi: perciò, in questa implementazione, si è scelto di inviare i

messaggi con cadenza regolare in quanto è stato ritenuto il metodo più corretto ai fini del progetto.

Topic_Manager Il secondo macroblocco è il *Topic_Manager* e le sue funzionalità sono: 1. ricevere i messaggi provenienti dal blocco *Publisher*; 2. verificare se, con i valori presenti nell'ultimo messaggio ricevuto, il robot collide o meno con l'ambiente circostante; 3. tener traccia di tutti i messaggi ricevuti attraverso la creazione di uno "storico" in modo da salvare tutti i valori dei giunti ad ogni iterazione della simulazione.

Questo blocco è gestito dalla classe `TopicManager.h` al cui interno sono presenti diversi metodi e variabili di classe per la corretta gestione di tutte le funzionalità precedentemente elencate, i più importanti dei quali sono riportati all'interno del seguente frammento di codice:

TopicManager.h

```
class TopicManager {
   ...
private:
   TopicIteration<std::string, float> it; // current iteration
   TopicHistory hst; // iteration history
   ros::Publisher sub;
   ...
private:
   void publishThread();
   bool checkValues();

public:
   ...
   void addMessage(std::string _name, float _value);
   const TopicHistory& getHistory();
   void run();
   void stop();
}
```

Prima verranno analizzate le variabili di classe presenti nel codice:

- `it`: questa variabile è utilizzata per salvare i messaggi più recenti provenienti dal *Publisher*, cosa che avviene attraverso `TopicIteration` la cui classe ha lo scopo di salvare una singola iterazione della simulazione. Ciascuna iterazione è rappresentata da un insieme di coppie (chiave, valore) che vengono aggiunte al momento della ricezione del messaggio dal *Publisher*; poiché questa variabile tiene traccia solamente dell'ultima iterazione, tutti i valori vengono quindi azzerati dopo l'effettivo invio dei dati.

- `hst`: rappresenta lo storico dei messaggi ricevuti; mentre la variabile `it` tiene traccia dell'ultima iterazione, `hst` contiene tutte le informazioni relative a tutte le repliche della simulazione. Prima che il contenuto della variabile `it` venga azzerato, tutti i valori presenti in essa vengono inseriti nello storico:

in questo modo è possibile immagazzinare i valori dei giunti effettivamente inviati.

- `sub`: variabile che si occupa di inviare i comandi a ROS. Come descritto nel capitolo 2, la comunicazione fra processi può avvenire attraverso lo scambio di messaggi: in questo frangente, i messaggi vengono inviati al topic `joint_states` il cui contenuto verrà successivamente letto dal plugin di Gazebo (descritto nella sezione 4.3.2) per simulare i movimenti del robot nell'ambiente riprodotto.

Per quanto concerne i metodi presenti nella classe `TopicManager.h`, le loro attività sono le seguenti:

- `addMessage`: rappresenta l'interfaccia fra i due macroblocchi della figura 5.8; ciascun messaggio viene ricevuto sotto forma di coppia di valori per poi essere inserito all'interno della variabile `it` da `addMessage`;

- `getHistory`: come si può intuire dal nome, la sua funzione è quella di restituire lo storico dei messaggi inviati; tale funzionalità è stata implementata per poter salvare su file l'intera simulazione e, quindi, riprodurla (procedimento descritto successivamente);

- `checkValues`: al suo interno devono essere inseriti tutti i controlli necessari a garantire un comportamento sicuro durante la fase di azione del robot. In questa implementazione il metodo `checkValues` contiene solo le funzionalità relative al blocco *Collision Avoidance Module (CAM)*, il cui scopo è quello di verificare se il braccio del robot collide con il suolo; ciò non toglie la possibilità di inserire ulteriori controlli (in questo caso non richiesti). Osservando lo schema 5.8, la scelta di porre il blocco in tale posizione si può così motivare: il modulo *CAM* deve conoscere contemporaneamente il valore di più giunti (vedere 5.3) per controllare che il robot non collida con il suolo, condizione che è possibile ottenere con sicurezza solamente a questo livello dell'architettura.

- `run, stop`: utilizzato per avviare e fermare l'invio dei messaggi al robot; più precisamente, il suo scopo è quello di avviare o terminare l'esecuzione del thread `publishThread`, la cui funzione verrà ora descritta;

- `publishThread`: contiene il codice relativo al core del *Topic_Manager*. Tale codice viene eseguito ciclicamente attraverso l'utilizzo di un thread dedicato che si occupa di svolgere secondo un preciso ordine tutte le funzioni del *Topic_Manager* rappresentate nella figura 5.8. Una volta che il thread viene avviato, vengono compiuti i seguenti passi:

  a) si controlla la presenza di nuovi messaggi ricevuti dal Publisher; finché non ce ne sono, il thread si mette nello stato di attesa;

b) alla ricezione di nuovi messaggi, tutti i valori in essi contenuti vengono inviati al modulo *Collision Avoidance Module* attraverso il metodo `checkValues`. Se tale metodo convalida i valori dei giunti garantendo la sicurezza durante i movimenti del robot, allora si può procedere con la trasmissione dei dati; altrimenti, i messaggi vengono scartati e non verrà quindi inviato alcun valore al robot;

c) si inviano i dati validati al robot attraverso la pubblicazione di un messaggio al topic `joint_states`.

Il thread permette l'invio dei messaggi al topic in due modalità:

1. *ASAP (as soon as possible)* Appena il thread rileva la presenza di nuovi messaggi all'interno del *Messages Manager* cerca di inviare le informazioni in essi contenute il più velocemente possibile;
2. *Periodic* A differenza della modalità precedente, nel caso in cui siano arrivati nuovi messaggi dal *Publisher*, prima di inviarli al topic il thread attende un intervallo di tempo che si può pre-impostare prima dell'avvio; così facendo la spedizione dei messaggi avviene periodicamente e a intervalli regolari.

La selezione della modalità con cui pubblicare i messaggi avviene specificando il valore relativo al periodo T: se $T = 0$ viene scelta la modalità *ASAP*, altrimenti viene selezionata la modalità *Periodic*.

Un ultimo appunto riguarda le operazioni presenti nella classe `TopicManager` e rese disponibili all'esterno: tutte le azioni sono thread-safe, vale a dire che non ci sono problemi di concorrenza e i dati a cui si ha accesso sono dati consistenti.

5.5 I/O su file

All'interno del software è presente un modulo il cui scopo è quello di salvare o caricare le varie traiettorie del Comau Smart-5 Six; le informazioni utili per salvare e replicare il moto del braccio manipolatore sono:

- il periodo di campionamento;
- i valori degli angoli dei giunti.

Tali dati sono stati salvati su file sfruttando il formato *xml* e creando una struttura al cui interno sono presenti due sezioni:

- intestazione, in cui sono presenti informazioni quali data/ora di registrazione dei movimenti e periodo di campionamento utilizzato per rivelare i valori dei giunti;

- simulazione, in cui sono presenti tutti i valori catturati degli angoli dei giunti; i dati sono suddivisi in *iterazioni*, ciascuna delle quali contiene una serie di coppie giunto/valore dove, come si può intuire, viene associato l'angolo al corrispettivo giunto.

La struttura *xml* segue la seguente sintassi:

Formato XML

```
<simulation>
   <description>
      <date>day-month-year</date>
      <time>hh:mm:ss</time>
      <clock_ms>clock</clock_ms>
   </description>
   <iteration number="0">
      <joint name="joint_name1">value1</joint>
      <joint name="joint_name2">value2</joint>
      <joint name="joint_name3">value3</joint>
      ...
   </iteration>
   ...
   <iteration number="..">
   ...
   </iteration>
   ...
</simulation>
```

Oltre all'utilizzo di questo formato, è possibile riprodurre i movimenti del robot anche attraverso un semplice file testuale in cui ad ogni riga è associata un'iterazione della simulazione; di seguito un esempio di tale file:

esempio di file testuale

```
0 -1.83526 -0.4114955 -2.784578
1 -1.862899 -0.4038254 -2.779821
2 -1.863929 -0.4026541 -2.781035
3 -1.85153 -0.4083631 -2.783653
4 -1.839732 -0.403978 -2.780025
...
```

Come si può constatare dall'esempio, non è specificato alcun giunto poiché si suppone che i valori presenti ad ogni riga del file siano stati inseriti osservando l'ordine di numerazione prestabilito per gli stessi (`joint1`, `joint2` e così via). Inoltre, in questo formato non è specificato alcun valore per quanto concerne il campionamento dei valori: tale mancanza è stata risolta scegliendo un valore predefinito (che non comporta alcun cambiamento rispetto ai movimenti registrati ma che determina invece una velocità dei movimenti dei giunti diversa da quella originale).

5.6 Interfaccia avvio del software

Il controller del Comau non offre alcuna finestra grafica all'avvio, motivo per il quale è stata sviluppata una semplice interfaccia testuale attraverso la quale si possono selezionare diverse opzioni, ciascuna delle quali influisce su cosa è richiesto di fare al programma. Per facilitare la spiegazione di ciascuna opzione, in Listing 5.1 è riportato un esempio della schermata di help del software che si può ottenere attraverso un apposito parametro d'avvio del programma:

Listing 5.1: Schermata Help

```
Usage: comauSim [options]
To run Comau Simulator in a terminal, after program has been launched,
execute this command into another terminal:
rosparam set launched true

Comau options:
  -h [ --help ]                     display this information
  -p [ --play ]                     play simulation from file (with -f option)
  -r [ --record ]                   record simulation (optionally save
                                    simulation with -f option)
  -x [ --xml2txt ]                  convert a xml file format into a txt file
                                    format
  -H [ --min-height (cm) ] arg (=5) set minimum height from floor of Comau Arm
  -f [ --filename ] arg             specify filename for recording OR playing;
                                    parameter behavior:
                                      PLAY mode: this parameter MUST BE ADDED;
                                      RECORD mode: this parameter is OPTIONAL.
                                                   If added, the movements are
                                                   stored in filename;
                                      XML2TXT mode: this parameter MUST BE
                                                    ADDED.
```

Il programma può essere avviato principalmente in una delle due modalità supportate: *play* o *record*.

Nella modalità *play* il controller del comau svolge l'attività di riprodurre una simulazione salvata su file che può essere sia in formato `xml` che `txt`. In questa circostanza è quindi obbligatorio fornire il nome del file da riprodurre, pena la notifica di un errore nella sintassi all'avvio del software. Un esempio corretto di avvio nella modalità *play* è:

```
rosrun comau_sim main -p -f nomefile.xml/.txt
```

Nella modalità *record* il controller si occupa di catturare i movimenti provenienti dalle periferiche di input e, se richiesto, salvare l'intera simulazione su file. Prima di avviare il software in questa modalità è necessario far partire l'eseguibile che si occupa della gestione degli eventi del mouse eseguendo la seguente riga in un terminale:

```
rosrun comau_sim mouseGesture
rosparam set launchedMouse true
```

Successivamente è possibile avviare la registrazione, fase in cui si può specificare l'altezza minima del braccio dal suolo (in centimetri): il suo valore predefinito è pari a 5cm. Qualora ci fosse l'esigenza di salvare la simulazione su file, è necessario specificare il nome di quest'ultimo ricordando di impostare come estensione `.xml`. Di seguito un esempio per l'avvio di questa attività è il seguente:

```
rosrun comau_sim main -r -H 10
rosrun comau_sim main -r -f nomefile.xml
```

Attraverso il comando riportato nella prima riga avviene la registrazione e l'altezza minima è pari a 10cm; attraverso il secondo, l'altezza minima è di 5cm e, conclusa la simulazione, avviene il salvataggio su file.

Il terzo modo attraverso il quale può essere avviato il programma non corrisponde ad una vera e propria modalità di esecuzione. Scegliendo l'opzione `-x [--xml2txt]` il programma richiede in input un file nel formato `.xml` e lo converte nel formato `.txt`; tale opzione è stata sviluppata per poter superare il limite secondo il quale il software prevede il salvataggio solamente nel formato `.xml`. Una volta specificato il file da convertire e terminata l'esecuzione del programma, il file convertito avrà estensione `.txt`.

```
rosrun comau_sim main -x -f nomefile.xml
```

Con il codice di cui sopra, si avrà in uscita un file di testo il cui nome sarà `nomefile.xml.txt`.

L'ultima modalità è quella che permette di visualizzare la schermata di *help* (vedere 5.1). Per visualizzarla, è sufficiente eseguire la seguente riga:

```
rosrun comau_sim main -h
```

6 Robot Learning

Questo capitolo contiene la descrizione del sistema di apprendimento utilizzato per addestrare il robot a compiere uno specifico task. Come già anticipato nell'introduzione, il lavoro svolto è basato sul lavoro svolto di G. et al [11] in cui si introduce un metodo relativo al learning per robot ponendo maggiore attenzione al modello matematico utilizzato a tale scopo. Questo capitolo sarà suddiviso in due parti: nella prima sarà presente un'introduzione teorica all'argomento relativo all'apprendimento e all'architettura teorica presente nel paper [11], mentre nella seconda parte verrà illustrato in dettaglio il modello utilizzato.

6.1 Robot Learning by Imitation

Il *Robot Learning* è quella parte della robotica che ha come obiettivo quello di far apprendere al robot come svolgere un determinato task; il *Robot Learning by Imitation*, detto anche *Programming by Demonstration (PbD)*, è una delle tecniche utilizzate per raggiungere l'obiettivo richiesto e, come si può intuire dal nome, si basa sul concetto di far apprendere al robot una precisa azione attraverso un insieme di dimostrazioni.

Le dimostrazioni sono rappresentate attraverso un insieme di dati corretti: a differenza di altre tecniche di apprendimento come per esempio il *Robot Learning From Failure [9]*, il *Programming by Demonstration* si basa sull'ipotesi che i dati forniti in input siano corretti, affinché il robot sia in grado di riprodurre nel modo più fedele possibile un preciso task.

Le caratteristiche relative alla tecnica *PbD* sono le seguenti:

- l'utente fornisce delle dimostrazioni corrette dell'azione che si vuole far apprendere al robot i cui dati possono essere prelevati operando direttamente il robot o tele-operando;
- il robot ha il compito di interpretare i dati di partenza per raggiungere l'obiettivo prefissato;
- l'apprendimento si basa su percezione e azione;
- il rapporto uomo-macchina è presente solo nella fase iniziale relativa alla creazione delle dimostrazioni corrette.

6.1.1 Architettura Modello

In questa sezione viene presentata l'architettura del sistema utilizzato.

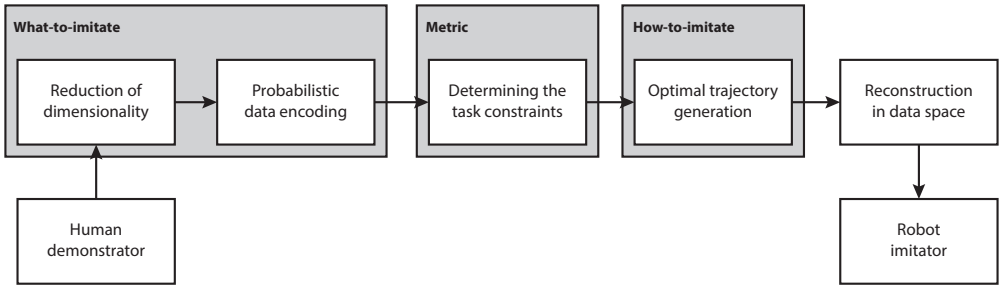


Figura 6.1: PbD - Architettura del sistema

1. *Human demonstrator*: fase relativa all'acquisizione dei movimenti del robot e alla creazione del dataset; ciascuna traiettoria prelevata rappresenta il task che si vuole far eseguire al robot. Le informazioni contenute all'interno di una dimostrazione sono relative a posizione, velocità e forza dei giunti: l'acquisizione di alcuni di questi dati avviene attraverso dei sensori presenti nel robot stesso;
2. *What-to-imitate*: viene deciso quali informazioni utilizzare fra quelle presenti nel dataset e come utilizzarle. A tal fine si svolgono due attività:
   - *Reduction of dimensionality*: riduzione della dimensionalità dello spazio campionario attraverso tecniche di analisi quali il *Principal Component Analysis (PCA)* oppure l'*Indipendent Component Analysis (ICA)*. Lo scopo è quello di semplificare i dati sui quali creare il modello, cercando le possibili correlazioni fra i dati presenti nel dataset acquisito al punto 1. Alla fine di questa procedura si ottiene un dataset di dimensionalità ridotta (*latent space*) le cui variabili sono indipendenti fra loro;
   - *Probabilistic data encoding*: codifica delle informazioni presenti nello spazio latente all'interno di un modello probabilistico i cui segnali sono per prima cosa allineati, ovvero si fa in modo che le diverse dimostrazioni abbiano il medesimo numero di dati campionati. Successivamente viene creato il modello probabilistico utilizzando modelli probabilistici quali *GMM (Gaussian Mixture Model)* o *BMM (Bernoulli Mixture Model)*;
3. *Determining the task constraints*: ricerca di vincoli quali posizione/orientamento di determinati giunti nello spazio per generalizzare il task in diversi contesti;
4. *How-to-imitate*: si determina come deve essere imitato il task verificando la compatibilità della traiettoria generata dal modello probabilistico che viene

successivamente ottimizzata con le capacità reali e i vincoli del robot preso in esame, in modo da garantire una corretta esecuzione del task;

5. *Reconstruction in data space*: procedura inversa rispetto alla fase inerente la riduzione di dimensionalità dello spazio il cui compito è quello di convertire lo spazio latente (*latent space*) nello spazio campionario originale dato in input al sistema;

6. *Robot imitator*: fase finale dell'architettura in cui si riproduce il task elaborato dal sistema nel robot reale.

6.2 Modello Realizzato

In questa sezione verrà descritto il modello utilizzato per la fase di learning del robot manipolatore che prende spunto da quello descritto precedentemente e riportato in [11].

Il modello realizzato è quello di figura 6.2.

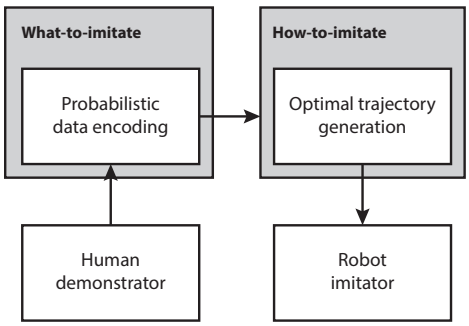


Figura 6.2: PbD - Modello Realizzato

Come si può vedere dallo schema 6.2, rispetto al modello 6.1.1 sono stati rimossi alcuni blocchi semplificandone così la struttura:

1. *Reduction of dimensionality*: lo spazio campionario utilizzato per il training del modello è composto solamente dai dati relativi alla posizione dei giunti del robot, posizioni che risultano essere indipendenti fra un giunto e l'altro. Per questo motivo non sono presenti dati quali velocità o forza poiché non ricavabili dalle specifiche tecniche disponibili: queste informazioni si potrebbero ottenere solamente dal robot reale.

2. *Determining the task constraints*: si ipotizza che l'ambiente del robot rimane invariato durante l'esecuzione del task e si suppone che gli oggetti siano sempre nella medesima posizione, condizione che elimina la necessità di valutare il task in diversi contesti;

3. *Reconstruction in data space*: data la mancanza del *latent-space*, per dualità non è richiesta nemmeno la funzione inversa, ovvero quella che si occupa di convertire lo spazio di dimensionalità ridotta allo spazio originale.

L'architettura prevede così solamente due blocchi:

1. *Probabilistic data encoding*: la funzione è analoga a quella descritta nel modello precedente; in questa realizzazione, è stato utilizzato il modello matematico *Gaussian Mixture Model (GMM)*;

2. *Optimal trajectory generation*: per la ricostruzione della traiettoria è stata utilizzata la *Gaussian Mixture Regression (GMR)*, la quale prende in input la descrizione della GMM e ricostruisce la traiettoria migliore.

Il Robot Learning si articola quindi in 4 fasi:

1. si costruisce un insieme di dimostrazioni iniziali corrette ciascuna delle quali rappresenta il task che si desidera far apprendere al robot;

2. si costruisce il modello di apprendimento attraverso la GMM;

3. si utilizzano i risultati ottenuti dal modello per creare la traiettoria corretta attraverso la GMR;

4. si riproduce l'azione risultante.

I punti chiave risultano quindi essere il secondo e il terzo: si analizzerà ora in dettaglio il funzionamento del *Gaussian Mixture Model (GMM)* e del *Gaussian Mixture Regression (GMR)*.

6.2.1 Gaussian Mixture Model (GMM)

É un modello probabilistico utilizzato per modellare distribuzioni complesse di dati: l'idea base del modello è quella di suddividere un generico dataset di dati (*popolation*) in diverse partizioni (*subpopolation*) e rappresentare queste ultime attraverso una funzione di densità di probabilità. L'obiettivo è quindi rappresentare un segnale complesso attraverso una combinazione lineare di semplici distribuzioni di probabilità (*Mixture Model*) utilizzando, come si può intuire dal nome stesso del modello, un insieme di gaussiane per la modellazione di distribuzioni complesse di dati.

Descrizione del Modello

Per la descrizione del GMM sarà utilizzata la seguente notazione:

X - dataset;

n - numero di dimostrazioni;

T_s - numero di iterazioni presenti in una generica dimostrazione s;

N - numero di datapoints presenti nel dataset X;

D - dimensione dello spazio, ovvero il numero di variabili utilizzate;

K - numero di componenti Gaussiane.

Una singola dimostrazione contiene quindi le informazioni relative alla traiettoria del task preso in esame: sia τ_s la traiettoria considerata, essa può essere rappresentata dalla seguente notazione $\tau_s = \{\xi_{t,j}, \xi_{s,j}\}_{j=1}^{T_s}$, dove

$\xi_{t,j} \in \mathbb{R}$ - istante temporale/iterazione;

$\xi_{s,j} \in \mathrm{R}^{\mathrm{D}-1}$ - (D-1)-upla contenente i valori dei giunti.

Il dataset X è l'insieme delle dimostrazioni considerate, quindi $X = \{\tau_s\}_{s=1}^{n}$ il cui numero di datapoints totale è pari a $N = \sum_{s=1}^{n} T_s$.

Il *Gaussian Mixture Model* è utilizzato per modellare il dataset X attraverso un insieme (mixture) di componenti gaussiane; il GMM di K componenti è definito dalla funzione di densità di probabilità

$$p(\xi_j) = \sum_{k=1}^{K} p(k)p(\xi_j|k) \tag{6.1}$$

dove

$$\begin{aligned} p(k) &= \pi_k \\ p(\xi_j|k) &= \mathcal{N}(\xi_j|\mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}} e^{-\frac{1}{2}\left((\xi_j-\mu_k)^T\Sigma_k^{-1}(\xi_j-\mu_k)\right)} \end{aligned}$$

In questa funzione ξ_j è il singolo datapoint considerato, $p(k)$ è la priorità associata alla gaussiana (detto anche coefficiente di mix), $p(\xi_j|k)$ è la funzione di densità di probabilità condizionata e $\mathcal{N}(\xi_j|\mu_k, \Sigma_k)$ è la variabile aleatoria gaussiana con media μ_k e matrice di covarianza Σ_k. Affinché la funzione di densità di probabilità 6.1 sia valida deve inoltre essere

$$\sum_{k=1}^{K} p(k) = \sum_{k=1}^{K} \pi_k = 1 \qquad 0 \leq \pi_k \leq 1 \quad \forall k$$

Per prima cosa è necessario trovare i parametri del modello che equivale a trovare i parametri di ogni singola gaussiana $(\pi_k, \mu_k, \Sigma_k, E_k)_{k=1}^{K}$. Il parametro E_k è detto probabilità a posteriori

$$E_k = \sum_{j=1}^{N} p(k|\xi_j)$$

e, sfruttando il teorema di Bayes, si ha:

$$p(k|\xi_j) = \frac{p(k)p(\xi_j|k)}{\sum_{i=1}^{K} p(i)p(\xi_j|i)}$$

La stima dei parametri del modello avviene utilizzando l'algoritmo di Machine Learning *EM (Expectation Maximization [6])* il quale adotta la tecnica di ricerca locale per aumentare in modo monotono la validità del modello con il dataset impiegato. La procedura si articola in due fasi:

1. si effettua una stima veloce dei parametri attraverso l'algoritmo *k-means* [10], che consente di effettuare una prima clusterizzazione dei dati presenti nel dataset, calcolando i valori iniziali del modello;

2. i valori stimati inizialmente vengono ripetutamente affinati. Questa fase si articola a sua volta in altri due step:

   E-step (expectation): viene valutata la bontà di ripartizione dei dati all'interno dei cluster descritti dalle gaussiane "attuali"; più semplicemente, per ogni punto del dataset è calcolata la probabilità a posteriori di ogni componente e si valuta in quale percentuale il punto appartiene ad un cluster;

   M-step (maximization): la massimizzazione porta a migliorare i parametri del GMM sfruttando le probabilità a posteriori calcolate precedentemente.

Il punto 2 viene ripetuto e, alla fine di ogni iterazione, viene valutata la *log-likelihood (log-verosimiglianza)* che indica il grado di adattamento del modello ai dati forniti; la *log-likelihood* del GMM è:

$$\mathcal{L} = \sum_{j=1}^{N} \log(p(\xi_j)) \tag{6.2}$$

Il processo di EM termina quando $\frac{\mathcal{L}(t+1)}{\mathcal{L}^t} < C$ in cui C ha un valore di soglia che è possibile pre-impostare ($C = 0.01$).

Scelta del modello

Una complicazione dovuta all'algoritmo EM riguarda la scelta ottimale del numero di componenti gaussiane K per cui a priori non è possibile stabilire quale sia il valore migliore. Per ovviare a questo inconveniente un metodo comune è quello di provare a generare diversi modelli GMM al variare di K e, attraverso un criterio opportuno, scegliere il più valido.

Solitamente all'aumentare di K il corrispondente modello reagisce in maniera più positiva ottenendo un miglior valore di log-likelihood 6.2; aumentare notevolmente K comporta tuttavia due svantaggi: 1. il modello può complicarsi notevolmente dato l'aumento del numero di variabili in gioco; 2. si possono creare problemi di *over-fitting*. Per questi motivi è necessario arrivare ad un compromesso fra il numero di componenti gaussiane da utilizzare e l'adattamento del modello al dataset attraverso il criterio *Bayesian Information Criterion (BIC)*: oltre ad utilizzare il valore di log-likelihood 6.2, introduce un fattore di penalizzazione all'aumentare del numero di parametri utilizzati dal modello. Il valore che rappresenta tale criterio è così calcolato:

$$S_{\text{BIC}} = -\mathcal{L} + \frac{n_p}{2}\log(N) \tag{6.3}$$

dove $\mathcal{L}$ è la verosimiglianza $n_p = (K-1) + K\left(D + \frac{1}{2}D(D+1)\right)$ che rappresenta il numero di parametri liberi richiesti da una GMM composta da K gaussiane e N è il numero di datapoints.

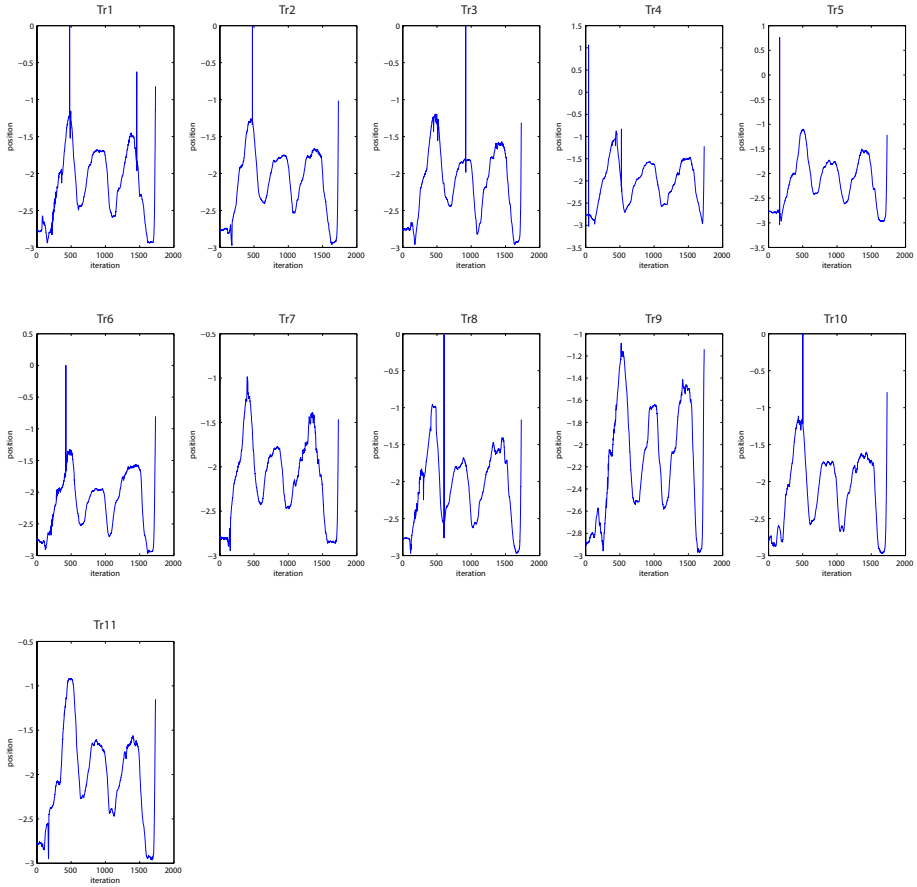
Sfruttando questo criterio e testandolo per diversi GMM al variare di K all'interno di un intervallo prestabilito (per esempio $1 \leq K \leq 20$), è possibile scegliere il modello che interagisce meglio con il dataset inserito.

Esempio Di seguito sono riportati 3 esempi del funzionamento del *Gaussian Mixture Model*. Sono stati considerati 3 dataset contenenti a loro volta i valori dei giunti 1,2,3 del robot Comau Smart-5 Six e che presentano le seguenti caratteristiche:

$n = 11$;
$T_s = 1731$ (tutte le n dimostrazioni hanno lo stesso numero di iterazioni);
$N = 1731 \cdot 11 = 19041$;
$D = 2$ - coppie (iterazione, posizione).

Le 11 traiettorie sono rappresentate graficamente nella figura 6.3.

Figura 6.3: Dataset - Joint 1

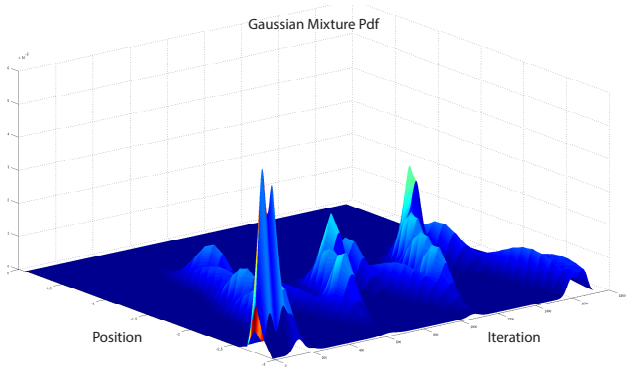


Con queste traiettorie il modello GMM generato presenta le seguenti caratteristiche:

$K = 16$
$\mathcal{L} = -132797$
$S_{\text{BIC}} = 133265$

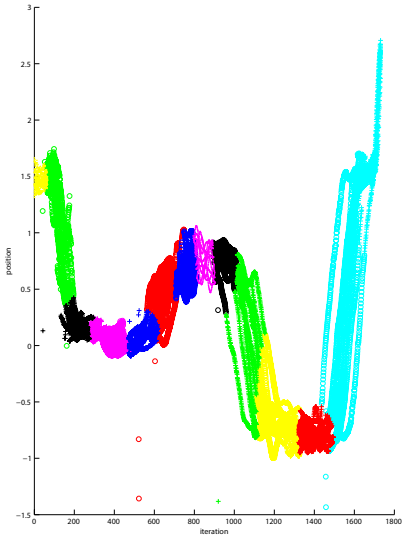
Il numero di gaussiane utilizzate per rappresentare il dataset preso in esame è 16. La funzione di densità di probabilità della mixture è rappresentata in figura 6.4.

Figura 6.4: Gaussian Mixture Model - Joint 1



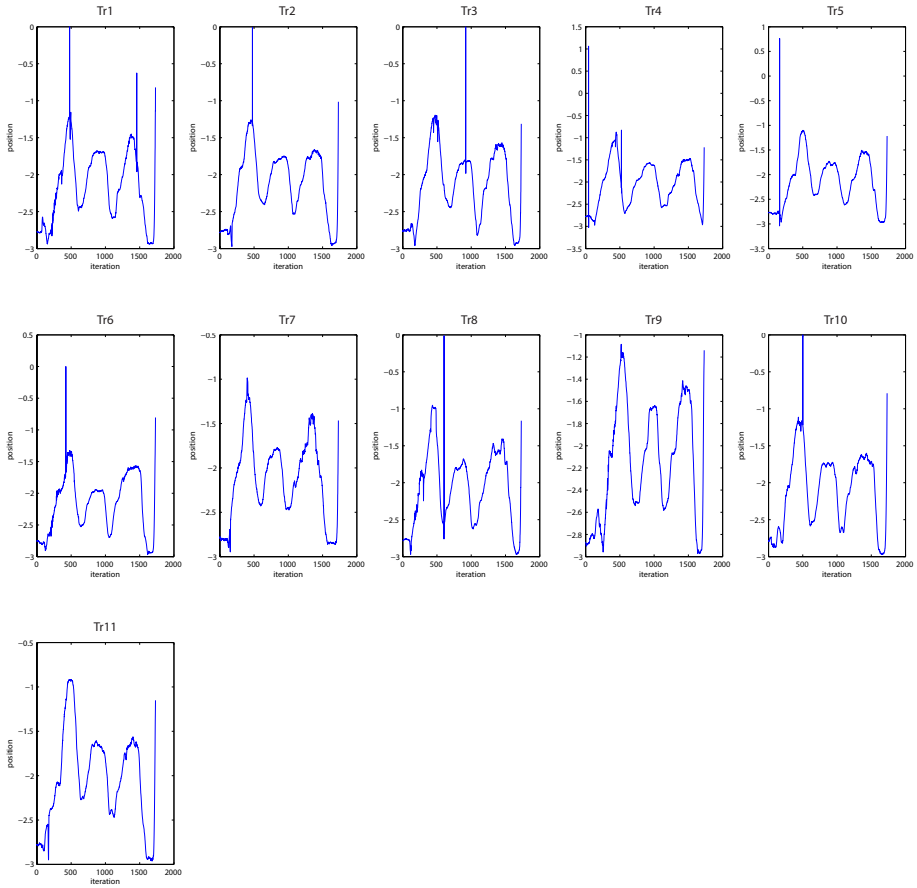
Attraverso questa GMM i dati presenti nel dataset sono clusterizzati come si può osservare in figura 6.5.

Figura 6.5: Cluster Dataset - Joint 1



Il dataset 2 contiene i movimenti relativi al giunto 2 del Comau Smart-5 Six. Le traiettorie presenti nel dataset sono:

Figura 6.6: Dataset - Joint 2



Con queste traiettorie il modello GMM generato ha le seguenti caratteristiche:

$K = 20$
$\mathcal{L} = -130595$
$S_{\text{BIC}} = 131181$

Il numero di gaussiane utilizzate per rappresentare il dataset preso in esame è 20. La funzione di densità di probabilità della mixture è rappresentata in figura 6.7 e i dati presenti nel dataset sono clusterizzati come indicato in figura 6.8.

Figura 6.7: Gaussian Mixture Model - Joint 2

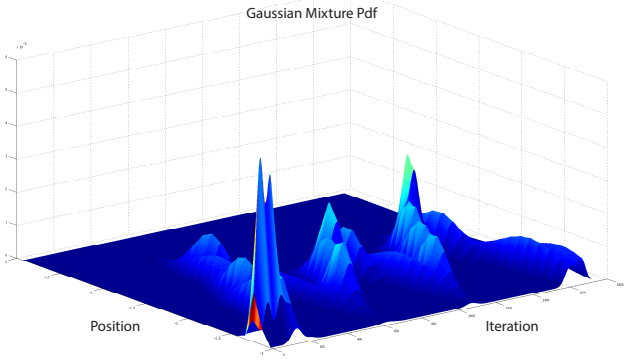
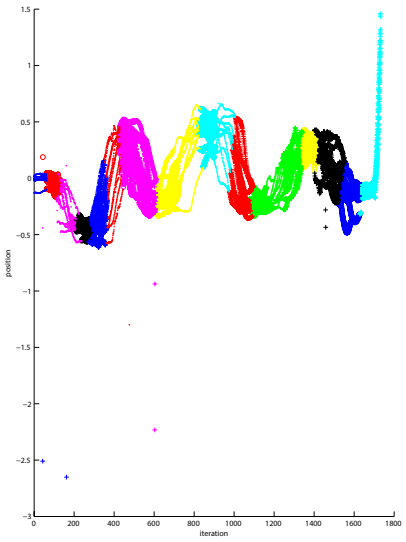
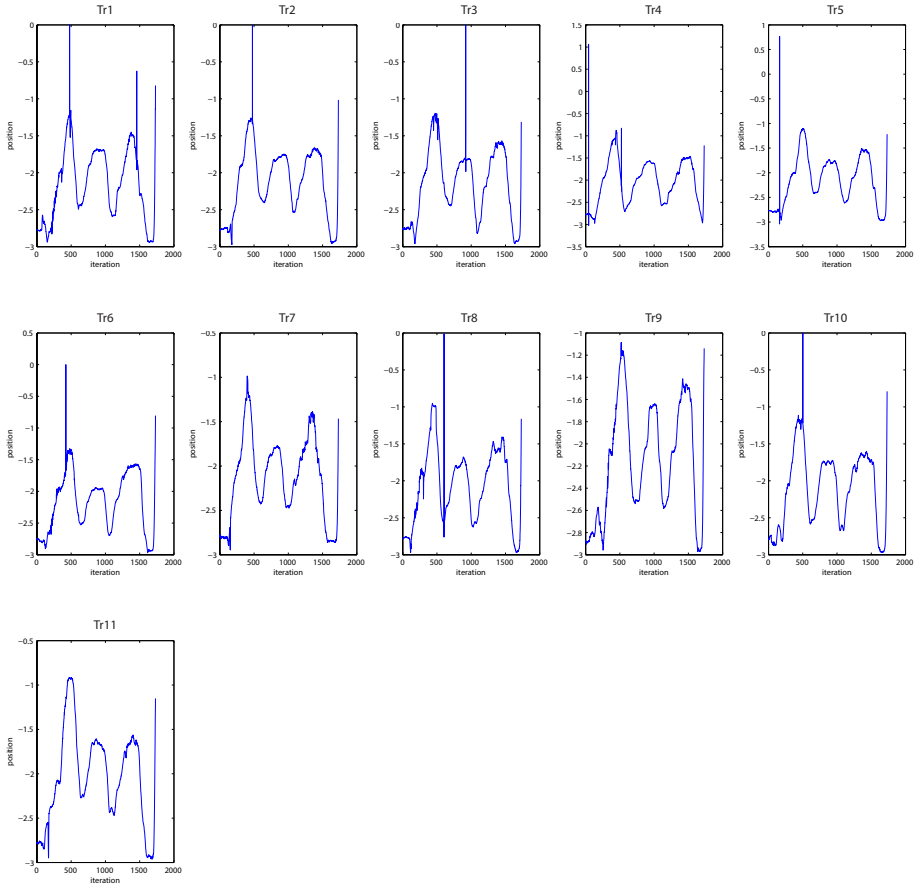


Figura 6.8: Cluster Dataset - Joint 2



Infine, il dataset relativo alle posizioni del giunto 3 del robot è:

Figura 6.9: Dataset - Joint 3

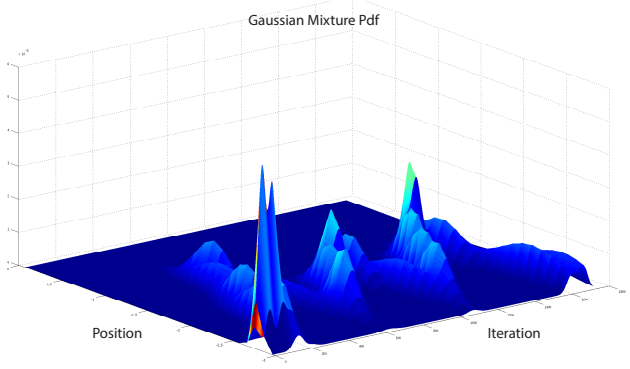


Con queste traiettorie il modello GMM generato presenta le seguenti caratteristiche:

$K = 20$
$\mathcal{L} = -136891$
$S_{\text{BIC}} = 137478$

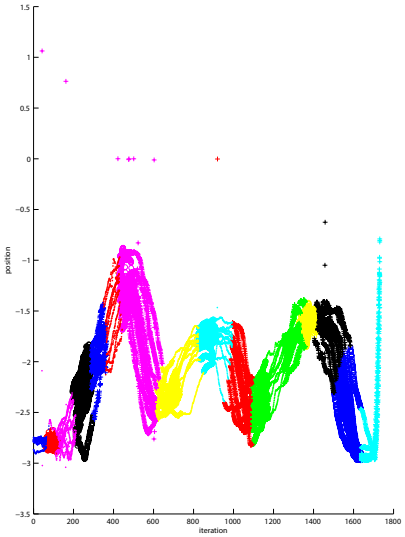
Il numero di gaussiane utilizzate per rappresentare il dataset preso in esame è 20. La funzione di densità di probabilità della mixture è rappresentata in figura 6.10.

Figura 6.10: Gaussian Mixture Model - Joint 3



Attraverso questa GMM, i dati presenti nel dataset sono clusterizzati come mostrato in figura 6.11.

Figura 6.11: Cluster Dataset - Joint 3



Il numero di gaussiane trovate dalla GMM per le traiettorie dei giunti 1,2 e 3 è rispettivamente 16, 20, 20 e, nonostante siano valori elevati, il numero di variabili in gioco risulta comunque eccessivo poiché si ha $D = 2$, ossia dataset composti da un numero ridotto di variabili. In queste circostanze il criterio S_{BIC} non ha quindi penalizzato l'aumento del numero di gaussiane per ciascuno dei 3 modelli generati.

6.2.2 Gaussian Mixture Regression (GMR)

Lo scopo della *Gaussian Mixture Regression* è la ricostruzione di una forma generale di un segnale partendo dalle informazioni probabilistiche contenute all'interno del *Gaussian Mixture Model*, che vengono utilizzate per rappresentare il dataset iniziale. Ciascuna componente gaussiana è definita da

$$\pi_k \quad , \quad \mu_k = \{\mu_{t,k}, \mu_{s,k}\} \quad , \quad \Sigma_k = \begin{pmatrix} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{pmatrix} \tag{6.4}$$

dove, indicando con D la dimensione dello spazio della GMM, si ha:

- π_k: coefficiente di mix;
- μ_k: vettore media $(D \times 1)$ della k componente gaussiana;
- Σ_k: matrice di covarianza $(D \times D)$ della k componente gaussiana.

I valori presenti nel vettore μ_k della k componente gaussiana contengono la descrizione probabilistica di due tipi di informazioni: 1. $\mu_{t,k}$, vettore contenente i dati noti a priori (input) (es. istanti temporali/componenti temporali) e i valori relativi alle componenti spaziali; 2. Σ_k, matrice contenente le correlazioni fra i dati noti a priori e quelli non.

Sia $X = \{X_t, X_s\}$ una singola iterazione del segnale che si desidera ricostruire, dove X_t indica il valore noto a priori (es. tempo) e X_s i valori da ricostruire (es. componenti spaziali); la ricostruzione del segnale avviene valutando per tutti i valori di ingresso noti a priori il seguente valore atteso condizionato di X_s dato X_t

$$E[X_s|X_t] = \sum_{k=1}^{K} \beta_k \mu_k \tag{6.5}$$

e, se necessaria, la covarianza condizionata di X_s dato X_t

$$\mathrm{Cov}\,(X_s|X_t) = \sum_{k=1}^{K} \beta_k^2 \, \hat{\Sigma}_{s,k} \tag{6.6}$$

con

$$\beta_k = \frac{\pi_k \mathcal{N}\,(X_t|\mu_{t,k}, \Sigma_{t,k})}{\sum_{j=1}^{k} \pi_j \mathcal{N}\,\left(X_t|\mu_{t,j}, \Sigma_{t,j}\right)} \tag{6.7}$$

$$\mu_k = \mu_{s,k} + \Sigma_{st,k}\,(\Sigma_{t,k})^{-1}\,(X_t - \mu_{t,k}) \tag{6.8}$$

$$\hat{\Sigma}_{s,k} = \Sigma_{s,k} - \Sigma_{st,k}\,(\Sigma_{t,k})^{-1}\,\Sigma_{ts,k} \tag{6.9}$$

Riassumendo, l'equazione chiave per la ricostruzione del segnale è la 6.5 e, valutando tale espressione per diversi valori di input noti a priori, è possibile ricostruire la traiettoria generata dal modello GMM/GMR.

7 Task

In questo capitolo si utilizzerà il modello *Gaussian Mixture Model* per far apprendere al robot Comau Smart-5 Six un semplice task e saranno considerati diversi dataset per valutare il comportamento del modello matematico al variare dei dati in ingresso. Alla fine del capitolo saranno riportati i risultati dell'apprendimento dell'azione presa in esame.

7.1 Descrizione Task

La configurazione iniziale dell'ambiente prevede il posizionamento di fronte al robot di 3 pile formate da 3 cubi ciascuna di uguale dimensione e massa, struttura che viene posizionata sopra un piano rialzato (per esempio un tavolo). Il task da far apprendere al robot è piuttosto semplice poiché l'azione del robot manipolatore consiste nel toccare ed eventualmente far cadere tutte e 3 le pile di cubi spingendo per ciascuna pila un cubo prestabilito.

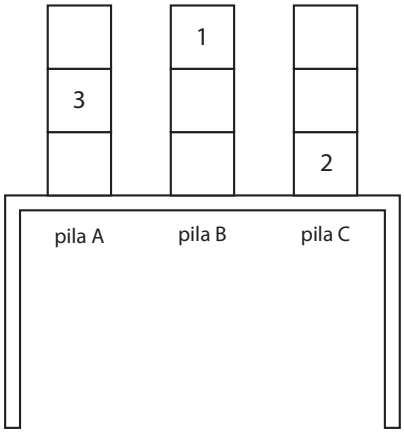


Figura 7.1: Task, configurazione ambiente

L'ordine da seguire per toccare ogni pila è riportato in figura 7.1: il primo cubo che il robot dovrà premere è quello in cima alla colonna B, il secondo cubo è quello collocato alla base della colonna C e l'ultimo è quello posizionato a metà della colonna A.

La scelta di premere ad ogni pila il cubo ad altezze diverse è stata fatta per valutare la bontà del mapping dei giunti (descritto nella sezione 5.2), per dimostrare che è possibile spostare il braccio manipolatore con una certa precisione e per verificare se il modello GMM/GMR riesce a creare una traiettoria tale da permettere al Comau Smart-5 Six di spostare i cubi secondo l'ordine prefissato.

7.2 Test

I test sono stati fatti valutando 4 diversi dataset ciascuno dei quali contiene un insieme di traiettorie descritte attraverso un determinato numero di iterazioni; per ogni iterazione sono presenti 4 campi: il primo è relativo all'istante temporale mentre gli altri 3 contengono rispettivamente i valori dei giunti 1,2 e 3. Tutti e 4 i dataset svolgono il task esaminato mentre le traiettorie presenti sono diverse fra loro e sono suddivise nel seguente modo:

1. *dataset A1*, contiene traiettorie non uniformi tra loro, perciò i movimenti del braccio manipolatore possono variare in modo visibile fra una traiettoria e l'altra;

2. *dataset A2*, questo dataset contiene le traiettorie presenti nel dataset A1 allineate in modo che abbiano tutte lo stesso numero di istanti temporali;

3. *dataset B1*, contiene traiettorie piuttosto uniformi permettendo al robot Comau Smart-5 Six di eseguire movimenti che risultano simili fra loro;

4. *dataset B2*, come per il dataset A2, le traiettorie presenti in questo insieme corrispondono a quelle del dataset B1 con lo stesso numero di istanti temporali.

Per ottenere le traiettorie presenti in A2 e B2, ovvero con lo stesso numero di istanti temporali, è stata utilizzata la tecnica di interpolazione di *Hermite (Hermite Interpolation)* che, come si potrà verificare successivamente, aiuta ad ottenere traiettorie più simili fra loro semplicemente allineandole temporalmente.

Per ogni dataset sono stati valutati diversi modelli GMM/GMR variando il numero di componenti gaussiane da 1 a 20 e scegliendo il modello più corretto, a seconda del miglior valore di S_{BIC} (6.3). Per ciascun modello saranno poi riportate le sue caratteristiche e, infine, sarà visualizzata la traiettoria generata con la *Gaussian Mixture Regression*; non sarà però possibile rappresentare graficamente la funzione di densità di probabilità di ogni GMM in quanto il parametro $D = 4$ comporta una dimensionalità troppo elevata per essere rappresentata.

7.2.1 Dataset A1

In questo dataset sono presenti 8 traiettorie.

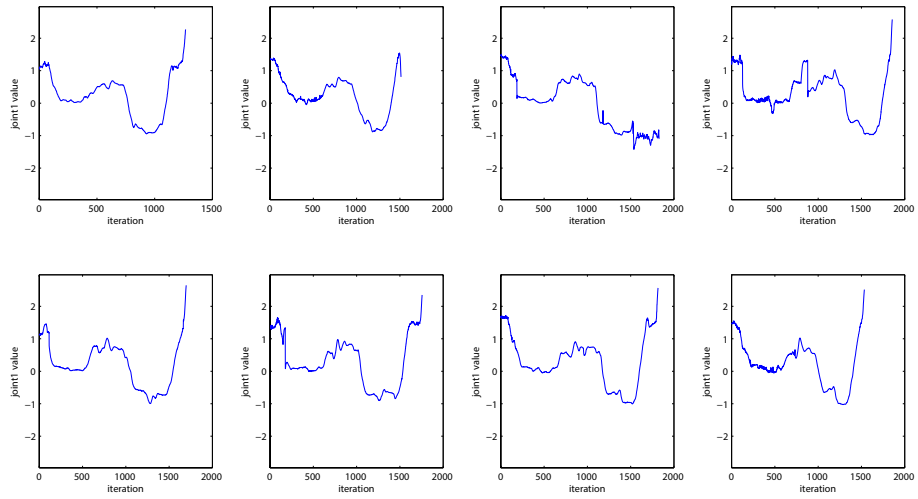


Figura 7.2: Dataset A1 - Joint 1

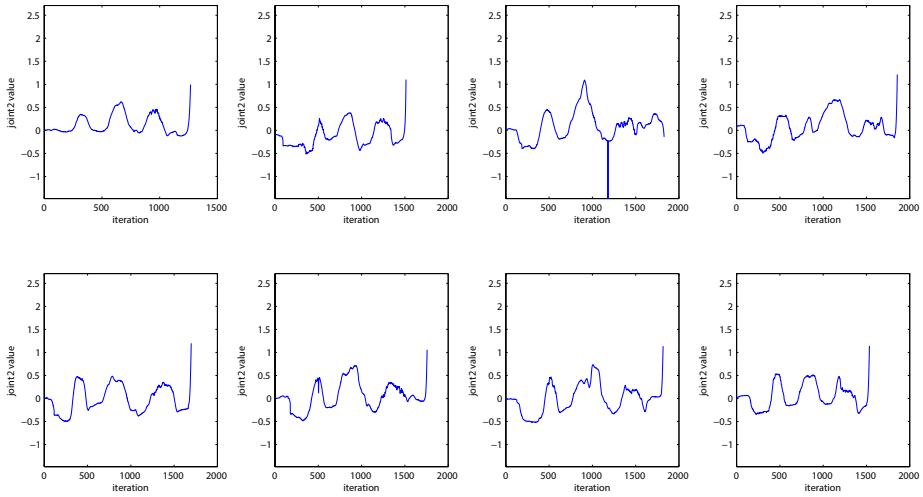


Figura 7.3: Dataset A1 - Joint 2

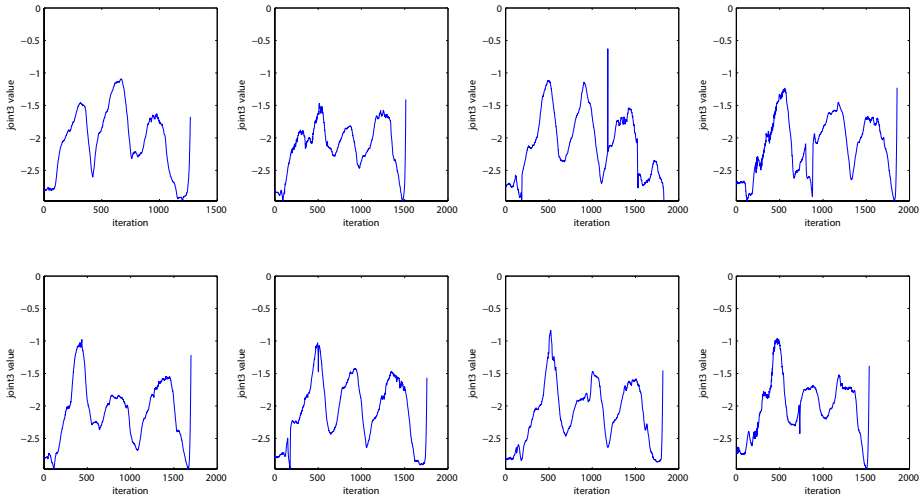


Figura 7.4: Dataset A1 - Joint 3

Tabella 7.1: Criterio S_{BIC} al variare di K

K	$\mathcal{L}$	S_{BIC}
1	-118636	118702
2	-110488	110625
3	-107054	107263
4	-102843	103123
5	-99313.3	99664.4
6	-95441.9	95864.2
7	-95398	95891.5
8	-94143.7	94708.4
9	-93005.3	93641.2
10	-93094.3	93801.4
11	-91300.2	92078.4
12	-91116.1	91965.5
13	-90652.9	91573.5
14	-90064.8	91056.5
15	-90141.2	91204.2
16	-88934.7	90068.8
17	-89104.7	90310
18	**-88508.6**	**89785.1**
19	-89065.9	90413.6
20	-88640.6	90059.4

Il modello generato ha quindi le seguenti caratteristiche:

$D = 4$

$n = 8$
$K = 18$
$\mathcal{L} = -88508.6$
$S_{\text{BIC}} = 89785.1$

Il segnale ricostruito per ognuno dei 3 giunti è quello evidenziato dalla linea rossa delle figure 7.5, 7.6 e 7.7

Figura 7.5: Dataset A1 - GMR Joint 1

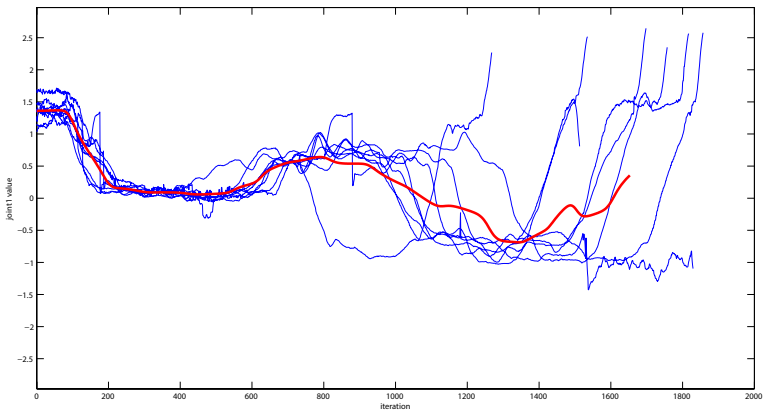


Figura 7.6: Dataset A1 - GMR Joint 2

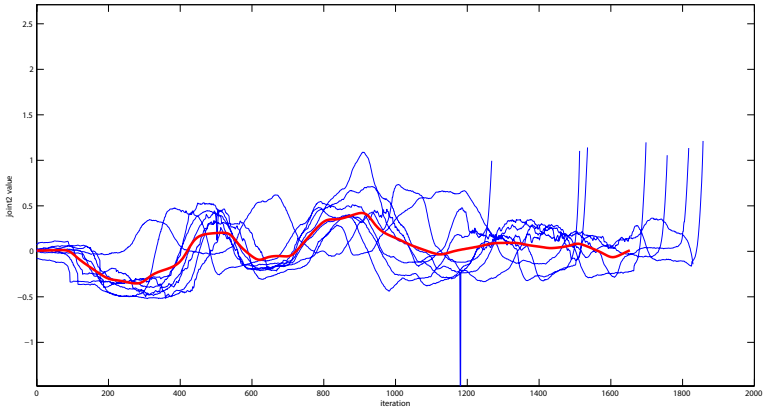
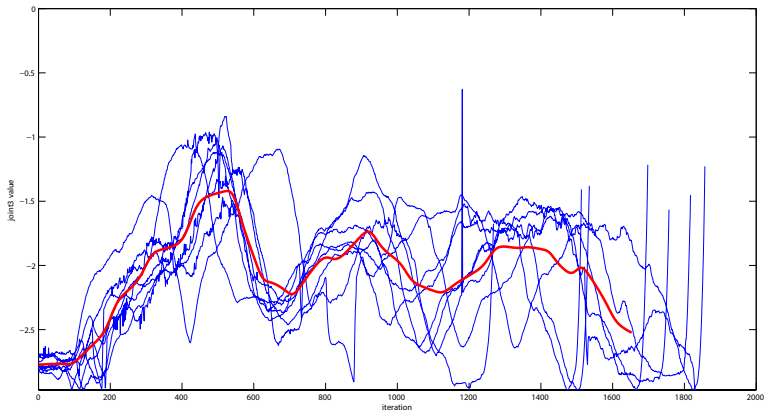


Figura 7.7: Dataset A1 - GMR Joint 3



Come si può vedere dai grafici 7.5, 7.6 e 7.7, le traiettorie presenti nel dataset sono piuttosto diverse fra loro e il modello probabilistico non riesce a crearne una valida per nessuno dei 3 giunti considerati.

7.2.2 Dataset A2

Come per il dataset A1, anche l'A2 contiene 8 traiettorie.

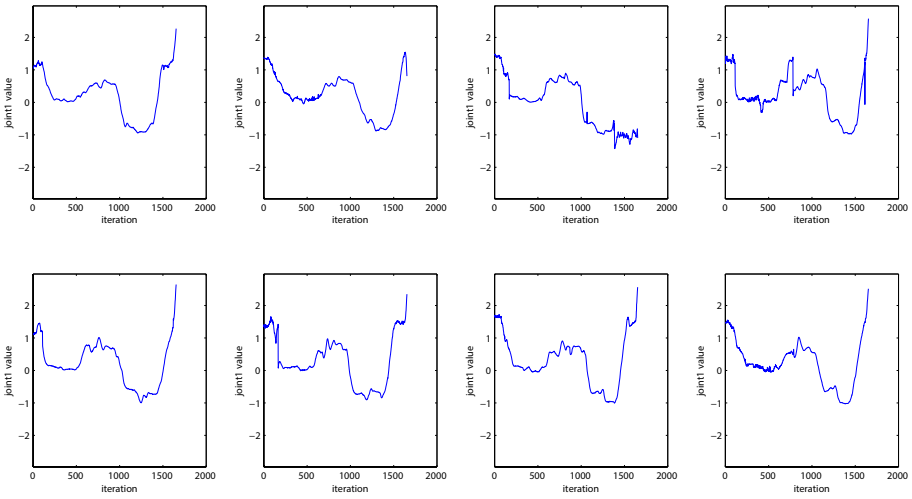


Figura 7.8: Dataset A2 - Joint 1

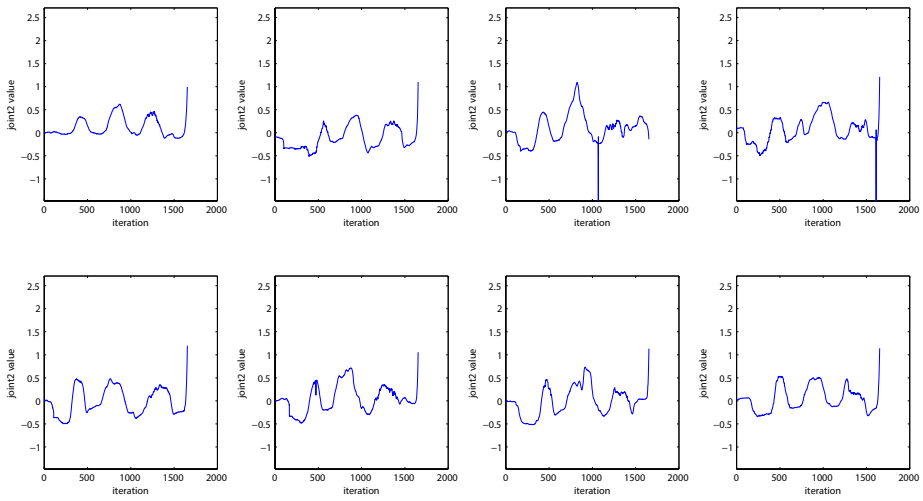


Figura 7.9: Dataset A2 - Joint 2

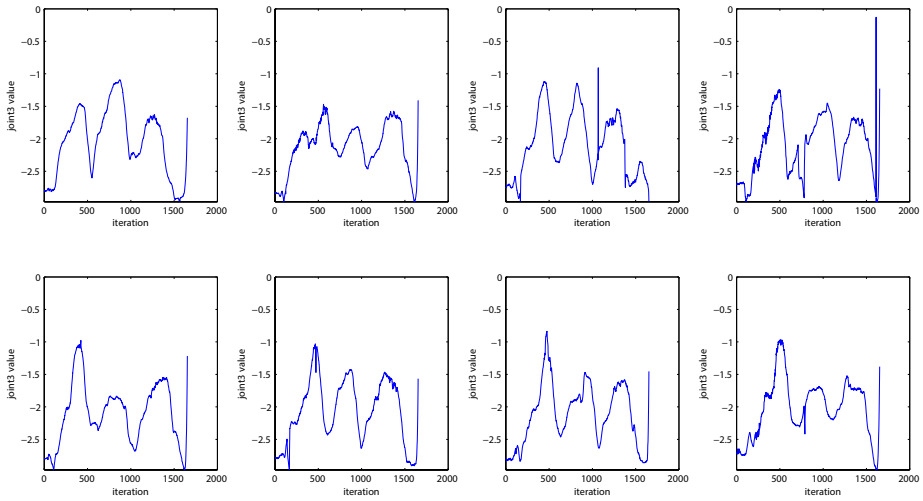


Figura 7.10: Dataset A2 - Joint 3

Tabella 7.2: Criterio S_{BIC} al variare di K

K	$\mathcal{L}$	S_{BIC}
1	-118667	118734
2	-112058	112195
3	**-105591**	**105800**
4	-106521	106801
5	-108521	108873

Tabella 7.2: continua nella prossima pagina

Tabella 7.2: continua dalla pagina precedente

K	$\mathcal{L}$	S_{BIC}
6	-105588	106010
7	-105386	105879
8	-109434	109999
9	-108303	108939
10	-107976	108683
11	-107553	108332
12	-114963	115813
13	-113599	114520
14	-115738	116730
15	-117964	119027
16	-129085	130219
17	-116150	117355
18	-128424	129701
19	-120747	122095
20	-122434	123853

Il modello generato ha quindi le seguenti caratteristiche:

$D = 4$
$n = 8$
$K = 3$
$\mathcal{L} = -105591.6$
$S_{\text{BIC}} = 105800$

Il segnale ricostruito per ognuno dei 3 giunti è quello evidenziato dalla linea rossa delle figure 7.11, 7.12 e 7.13

Figura 7.11: Dataset A2 - GMR Joint 1

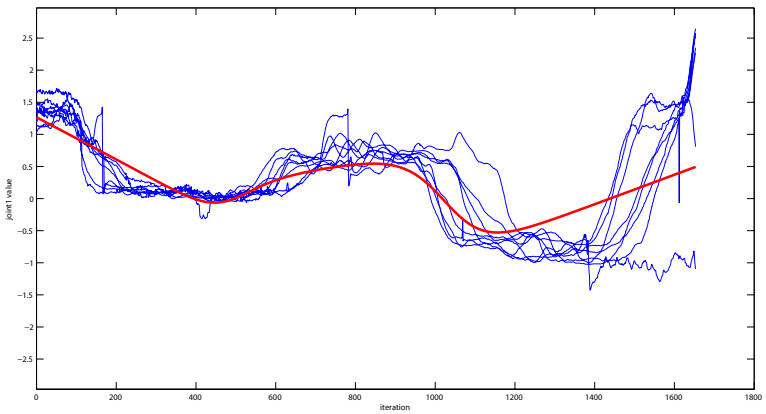


Figura 7.12: Dataset A2 - GMR Joint 2

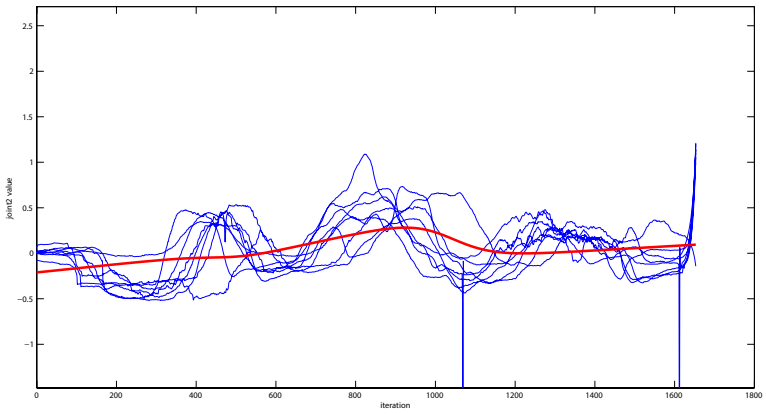
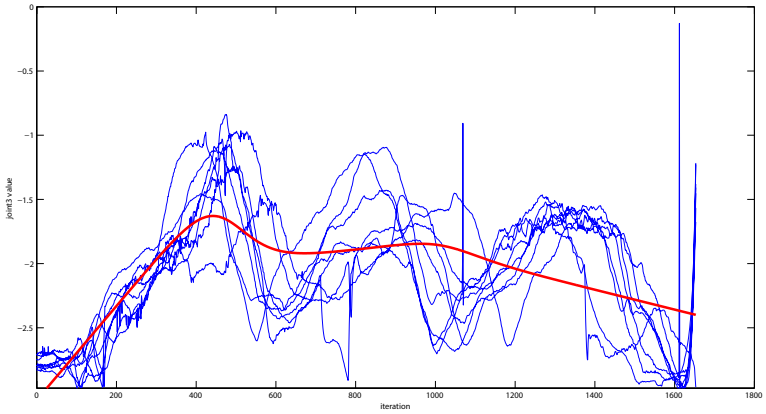


Figura 7.13: Dataset A2 - GMR Joint 3



A differenza del dataset precedente, le traiettorie presenti in questo insieme sono allineate e, come si può vedere dai grafici 7.11, 7.12 e 7.13, presentano tutte il medesimo numero di iterazioni. Le traiettorie generate dalla GMR per i giunti 1 e 2 seguono meglio l'andamento delle altre traiettorie rispetto al dataset A1, cosa non valida invece per il moto generato per il giunto 3: a causa della diversità delle traiettorie presenti nel dataset, la GMR approssima eccessivamente l'andamento rispetto agli altri percorsi generando un segnale che non rispetta quelli presenti nel dataset.

7.2.3 Dataset B1

Il dataset B1 contiene 11 traiettorie simili fra loro.

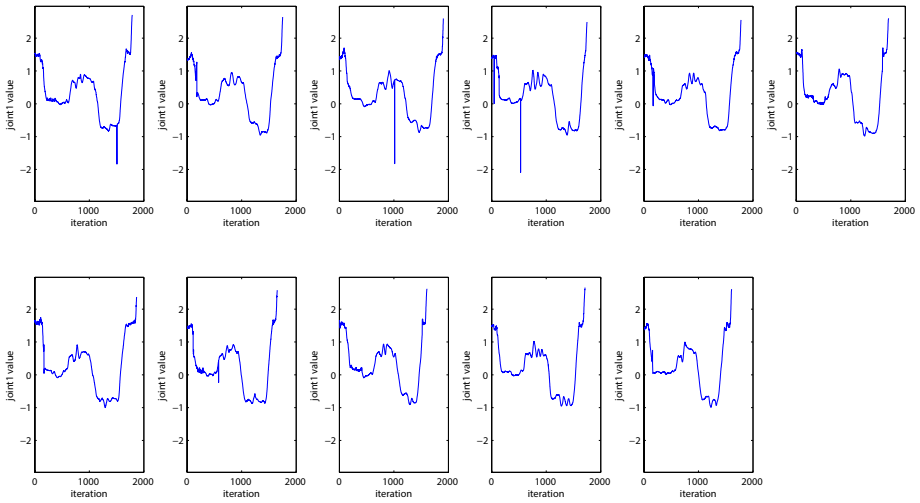


Figura 7.14: Dataset B1 - Joint 1

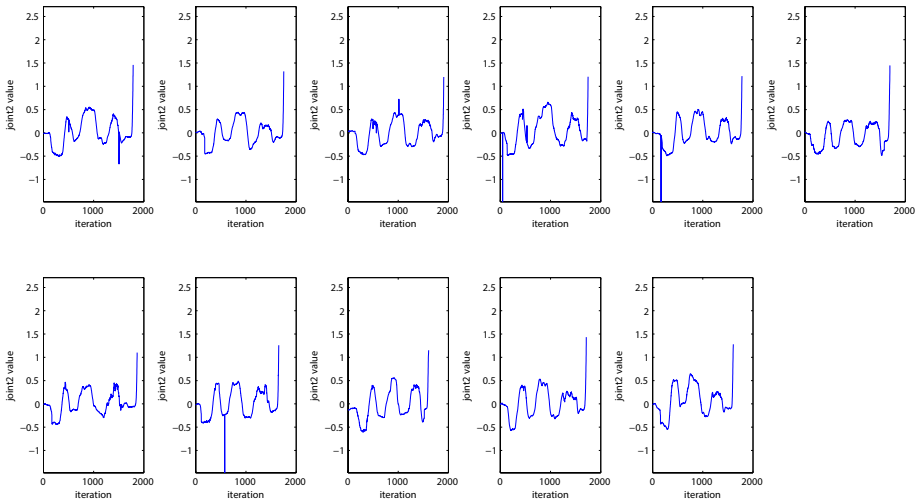


Figura 7.15: Dataset B1 - Joint 2

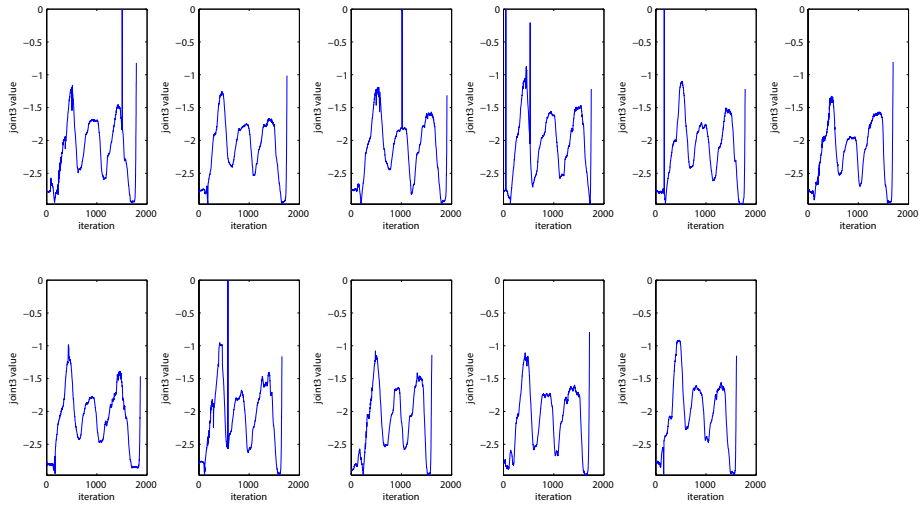


Figura 7.16: Dataset B1 - Joint 3

Tabella 7.3: Criterio S_{BIC} al variare di K

K	$\mathcal{L}$	S_{BIC}
1	-169650	169719
2	-155045	155188
3	-145950	146167
4	-135852	136142
5	-125412	125776
6	-117553	117991
7	-114536	115049
8	-113815	114401
9	-113119	113779
10	-112013	112747
11	-105721	106529
12	-104101	104983
13	-103483	104439
14	-106180	107210
15	**-102886**	**103990**
16	-104591	105768
17	-104006	105258
18	-105663	106988
19	-104110	105510
20	-104018	105492

Il modello generato ha quindi le seguenti caratteristiche:

$D = 4$

$n = 11$
$K = 15$
$\mathcal{L} = -102886$
$S_{\text{BIC}} = 103990$

Il segnale ricostruito per ognuno dei 3 giunti è quello evidenziato dalla linea rossa delle figure 7.17, 7.18 e 7.19

Figura 7.17: Dataset B1 - GMR Joint 1

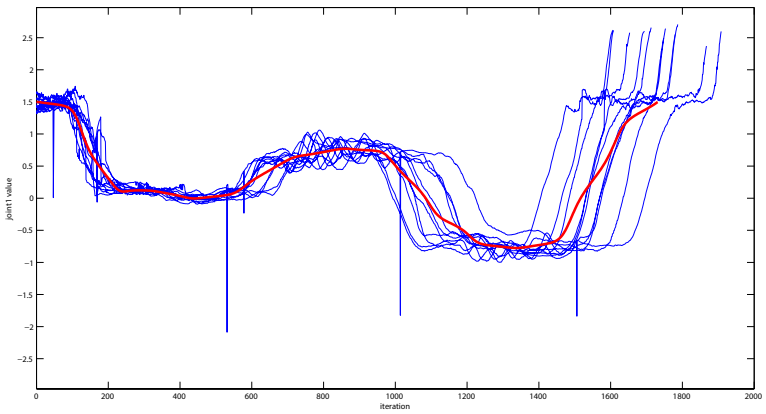


Figura 7.18: Dataset B1 - GMR Joint 2

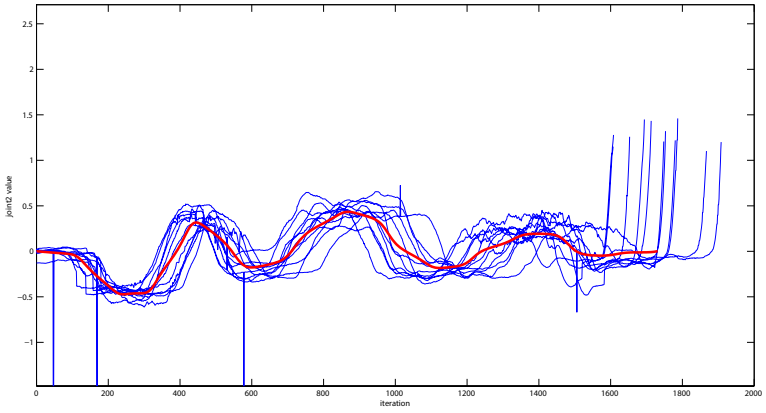
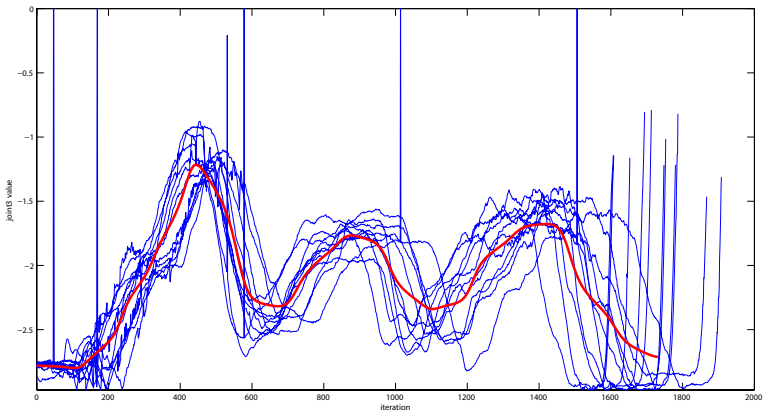


Figura 7.19: Dataset B1 - GMR Joint 3



I segnali generati dalla *Gaussian Mixture Regression* per questo dataset sono decisamente migliori rispetto a quelli di A1 e A2, fattore dovuto principalmente all'omogeneità delle traiettorie presenti in B1. I segnali generati per i giunti 1,2 e 3 seguono più fedelmente il percorso delle traiettorie presenti nel dataset rendendo i movimenti del braccio manipolatore più fluidi.

7.2.4 Dataset B2

Il dataset B2 contiene 11 traiettorie simili e allineato fra loro.

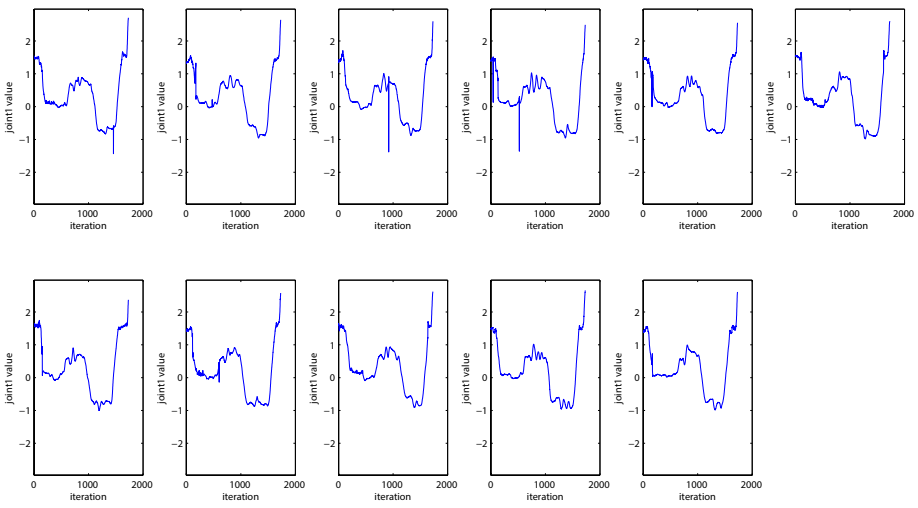


Figura 7.20: Dataset B2 - Joint 1

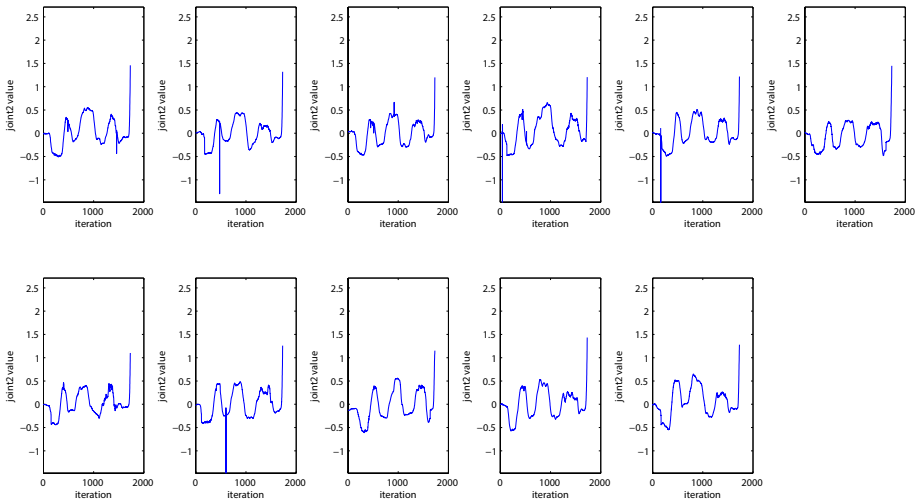


Figura 7.21: Dataset B2 - Joint 2

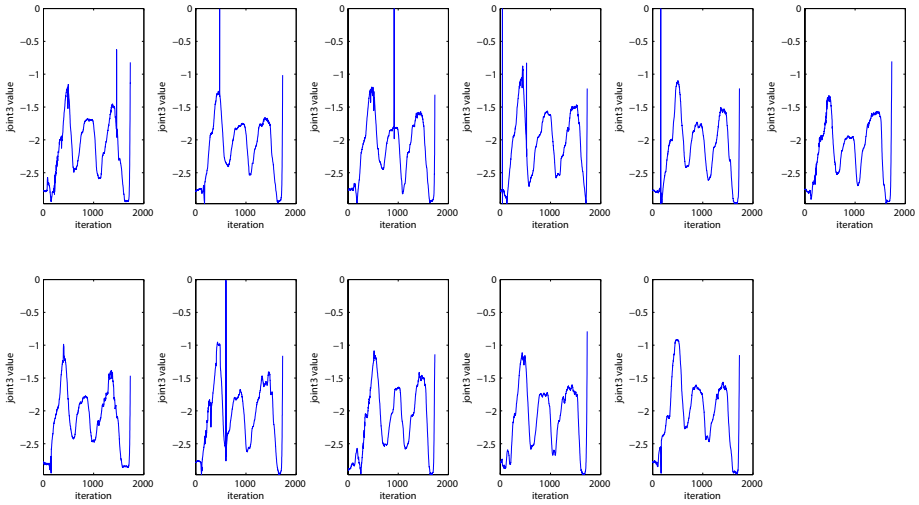


Figura 7.22: Dataset B2 - Joint 3

Tabella 7.4: Criterio S_{BIC} al variare di K

K	$\mathcal{L}$	S_{BIC}
1	-169659	169728
2	-156342	156485
3	-145406	145622
4	-138666	138957
5	-128003	128368
6	-120875	121314

Tabella 7.4: continua nella prossima pagina

Tabella 7.4: continua dalla pagina precedente

K	$\mathcal{L}$	S_{BIC}
7	-117007	117519
8	-118805	119391
9	-114401	115061
10	-118304	119038
11	-113713	114521
12	**-109103**	**109985**
13	-112292	113247
14	-118126	119156
15	-109722	110825
16	-114032	115210
17	-115882	117133
18	-111068	112394
19	-114400	115800
20	-115836	117309

Il modello generato ha quindi le seguenti caratteristiche:

$D = 4$
$n = 11$
$K = 12$
$\mathcal{L} = -109103$
$S_{\text{BIC}} = 109985$

Il segnale ricostruito per ognuno dei 3 giunti è quello evidenziato dalla linea rossa delle figure 7.23, 7.24 e 7.25

Figura 7.23: Dataset B2 - GMR Joint 1

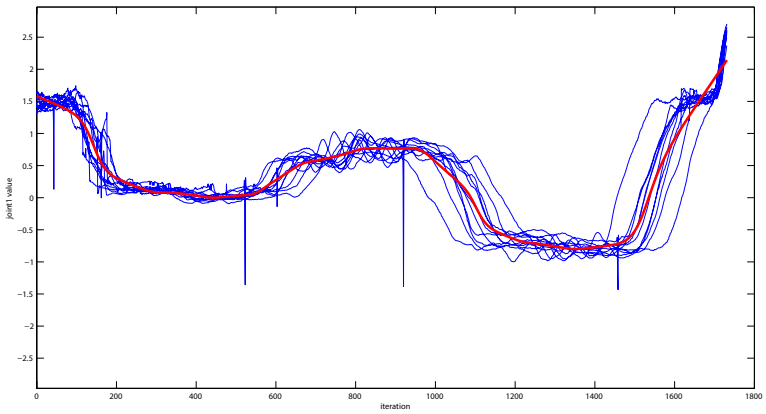


Figura 7.24: Dataset B2 - GMR Joint 2

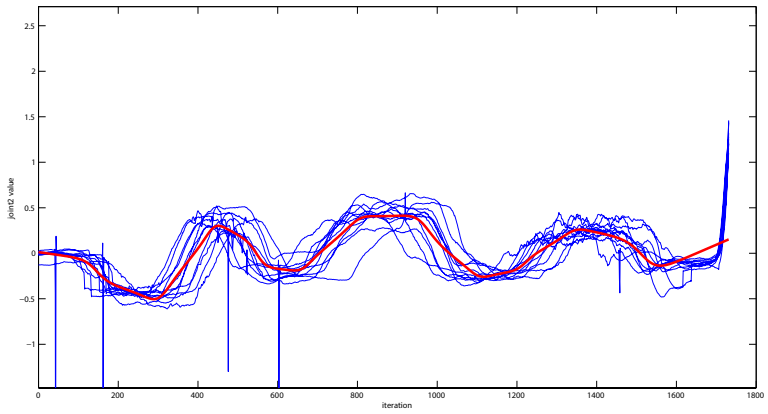
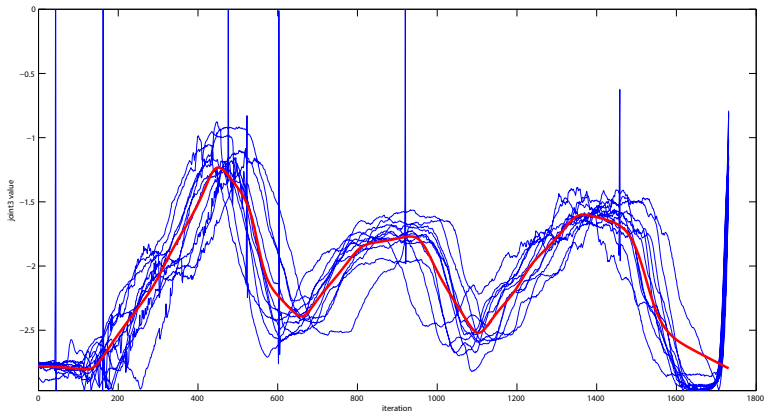


Figura 7.25: Dataset B2 - GMR Joint 3



Rispetto ai dataset A1,A2 e B1, i segnali generati dal modello GMM/GMR per il B2 si possono considerare i migliori corrispondendo del tutto le aspettative prefissate: questo risultato si ottiene poiché si hanno traiettorie simili fra loro e con lo stesso numero di iterazioni. Come si può vedere dalle figure 7.23, 7.24 e 7.25 i segnali generati seguono infatti fedelmente le traiettorie presenti nel dataset rendendole, come nel caso precedente, più *smooth*.

7.3 Risultati

L'obiettivo del task è far in modo che il robot Comau Smart-5 Six sposti le 3 pile di cubi facendole eventualmente cadere. Osservando le traiettorie generate dal modello GMM/GMR per tutti i dataset presi in esame si possono fare le seguenti valutazioni:

1. utilizzare le traiettorie con il medesimo numero di iterazioni comporta la

generazione di un segnale più fedele alle posizioni presenti nel dataset e, confrontando i dataset stessi a coppie, i segnali generati per A2 e B2 sono migliori rispetto a quelli creati per A1 e B1;

2. avere traiettorie omogenee fra loro all'interno del dataset aiuta drasticamente il modello GMM/GMR nella generazione di segnali che seguono un andamento più fedele ai dati in input. Utilizzando sia B1 che B2 il modello probabilistico è in grado di creare dei segnali più corretti rispetto a quelli prodotti per i dataset A1 e A2.

Il secondo punto è quello di maggiore importanza nel modello probabilistico: come descritto nel capitolo 6, una delle ipotesi che viene fatta è che l'utente fornisca dimostrazioni corrette dell'azione che si vuole far apprendere al robot. La correttezza delle dimostrazioni non considera quindi solo il fatto di portare a termine il task prestabilito ma tiene in considerazione che ci siano anche dimostrazioni simili che compiono l'azione in modo adeguato.

Riproducendo i segnali ottenuti dal modello GMM/GMR per tutti e 4 i dataset considerati, si ottengono i seguenti risultati:

Tabella 7.5: Risultati Learning

Dataset	T1	T2	T3	FA	FB	FC
A1	no	no	no	no	si	no
A2	no	no	no	no	si	no
B1	si	si	si	si	si	no
B2	si	si	si	si	si	si

T1 = robot manipolatore tocca il cubo 1
T1 = robot manipolatore tocca il cubo 2
T1 = robot manipolatore tocca il cubo 3
FA = robot manipolatore fa cadere la pila A
FB = robot manipolatore fa cadere la pila B
FC = robot manipolatore fa cadere la pila C

I risultati in tabella 7.5 seguono e confermano le aspettative: il task è stato appreso correttamente utilizzando il dataset B2, quasi completamente utilizzando il dataset B1, mentre l'apprendimento dello stesso risulta piuttosto scadente utilizzando i dati presenti in A1 e A2.

8 Conclusioni

La tecnica di programmazione *Programming by Demonstration (PbD)* è un processo che si occupa di far apprendere ad un robot l'esecuzione di una determinata azione. L'utilizzo di questa metodologia apporta il vantaggio di non dover scrivere alcuna linea di codice per far compiere i movimenti al robot poiché viene tutto appreso attraverso delle dimostrazioni. Questa tesi ha come obiettivo primario quello di valutare la bontà di questa tecnica di programmazione laddove si applichi ad un braccio manipolatore attraverso l'utilizzo del modello probabilistico *Gaussian Mixture Model - Gaussian Mixture Regression*. I risultati ottenuti rispecchiano quanto previsto: il modello si comporta molto bene nel caso in cui le dimostrazioni siano corrette e uniformi fra loro generando traiettorie tali da permettere l'esecuzione esatta del task; inoltre, i segnali creati risultano essere regolari, caratteristica importante per far compiere al robot dei movimenti dolci e privi di scatti.

Le dimostrazioni hanno quindi un ruolo chiave nel Pbd: rispetto a quanto sviluppato in altri lavori relativamente al Programming by Demonstration in cui le dimostrazioni avvengono spostando manualmente il robot e rilevando i dati dai sensori in esso presenti, in questa tesi il robot è comandato attraverso i dati RGB-D catturando i movimenti di una persona e mappando in modo opportuno questi spostamenti in quelli del robot manipolatore. Questa operazione di mapping ha richiesto un'accurata analisi per soddisfare gli obiettivi prefissati: permettere alla persona di avere subito familiarità con il braccio manipolatore ed evitare al robot stesso di effettuare movimenti bruschi e improvvisi.

I movimenti del robot sono stati valutati in un ambiente simulato per il quale è stato necessario sviluppare un modello tridimensionale del Comau Smart-5 Six per il framework utilizzato (ROS); il modello è stato realizzato inserendo i dati reali del robot, qualora disponibili, mentre per quelli mancanti sono stati utilizzati dei dati provenienti da un software di simulazione. In seguito, oltre al modello grafico, è stato sviluppato il software per permettere il controllo del robot all'interno del simulatore utilizzato implementando funzionalità secondarie quali il salvataggio e caricamento dei movimenti effettuati durante una simulazione.

I possibili sviluppi del lavoro svolto sono molteplici: per quanto riguarda il modello tridimensionale del braccio manipolatore, sarebbe opportuno sostituire i valori simulati con quelli reali controllando, successivamente, i movimenti di robot reale e simulato, verificando l'effettiva fedeltà del modello virtuale, scopo per il quale è necessario sviluppare dei driver dedicati per permettere al framework ROS di interfacciarsi con il robot reale.

Per quanto concerne il modello probabilistico, si potrebbero effettuare ulteriori

osservazioni riguardanti la valutazione di un dataset al cui interno siano presenti informazioni aggiuntive, quali la velocità dei giunti o la forza impartita (informazioni prelevate dai sensori interni del robot reale), oppure, si potrebbero prendere in considerazione diversi criteri di valutazione oltre al *Bayesian Information Criterion* per la scelta del modello e, dopo un'opportuna analisi, scegliere il criterio che soddisfa nel modo migliore le proprie esigenze.

Bibliografia

[1] Collada - digital asset and fx exchange schema. https://collada.org/.

[2] Gazebo. http://www.gazebosim.org.

[3] Openni - the standard framework for 3d sensing. http://www.openni.org.

[4] Ros (robot operating system). http://http://www.ros.org.

[5] Ruediger Dillmann Aude Billard, Sylvain Calinon and Stefan Schaal. *Handbook of Robotics, Chapter 59: Robot Programming by Demonstration*. 2007.

[6] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. 2007.

[7] Comau Robotics. *SMART SiX - Specifiche Tecniche*, 2005.

[8] Comau Robotics. *SMART 5 SiX*, 2011.

[9] Daniel H Grollman and Aude Billard. Donut as i do: Learning from failed demonstrations. 2011.

[10] Andrew W. Moore. K-means and hierarchical clustering - tutorial slides, 2001. http://www.autonlab.org/tutorials/kmeans11.pdf.

[11] Florent Guenter Sylvain Calinon and Aude Billard. On learning, representing and generalizing a task in a humanoid robot. 2007.

Grazie di cuore...

...al professore Emanuele Menegatti e al dottor Stefano Michieletto, per avermi assistito in questi mesi.
...ai miei genitori, Angelo e Mirella, per avermi sempre sostenuto e motivato durante tutti questi anni di studio.
...alla mia compagnia, per le innumerevoli esperienze trascorse assieme e per l'affetto che mi avete regalato.
...a Laura, per avermi aiutato nella stesura e correzione della tesi.
...a Enrico, per il tempo dedicatomi nell'effettuare i test del progetto.
...ai compagni di corso, per aver reso speciale questi anni accademici e per aver sopportato le mie "simpatiche" battutine (ammettetelo, sono stupende!).