Generazione e Generalizzazione di Traiettorie per i Robot

di Giorgio Grioli

Versione Provvisoria del 27 Ottobre 2023

Indice

1 Introduzione **5**

2 Generazione di Traiettorie per Interpolazione **9**
2.1 Interpolazione Polinomiale 9
2.2 Spline 11
2.3 Curve di Bezier 13
2.3.1 Esempio: generazione di traiettorie minimum-jerk con le curve di Bezier. 16
2.3.2 Note finali sulle curve di Bezier 17
2.4 Interpolazione di rotazioni 17
2.5 B-Spline e NURBS 20
2.5.1 B-Spline (completare) 20
2.5.2 NURBS 21
2.6 Note finali sulla generazione di traiettorie per interpolazione . 21

3 Interpolazione dinamica **23**

4 Generalizzazione di traiettorie **27**
4.1 Dynamic Movement Primitives 27
4.2 Modelli probabilistici impliciti 29
4.3 Filtrazione di traiettorie e Gaussian Mixture Models 29
4.4 Gaussian Mixture Models e Gaussian Mixture Regression . . . 31
4.4.1 I Gaussian Mixture Model 31

Capitolo 1

Introduzione

Testi Consigliati: [1] (versione del 2020 in italiano), [2] (versione del 2008, in inglese).

Spesso, il problema di descrivere una traiettoria per un sistema robotico viene risolto in un insieme finito di punti, a partire dai quali è richiesto costruire una traccia continua e/o una traiettoria definita in un intervallo continuo di tempo. Questo può capitare ad esempio quando una cinematica non è invertibile esplicitamente ed una traiettoria definita nello spazio cartesiano viene riportata nello spazio dei giunti tramite algoritmi numerici, oppure quando la traiettoria viene specificata soltanto da una serie di way-points.

In forma simbolica, questo problema noto con il nome di interpolazione, è formalizzabile come segue. Data una serie di $N+1$ punti $\{x_i\} \in \mathbb{X}$, con $i \in [0, N]$, dove $\mathbb{X}$ è un opportuno spazio metrico, si vuole determinare una funzione continua $f(t) : \mathbb{R} \to \mathbb{X}$ che assuma in sequenza tutti i valori x_i, in corrispondenza di altrettanti istanti $t_i \in \mathbb{R}$ dal primo all'ultimo.

Può essere richiesto che i valori x_i siano assunti in corrispondenza di una serie di t_i dati tali che $t_i < t_{i+1}$, nel qual caso si parla di interpolazione di traiettoria, o che la scelta degli istanti t_i sia libera (e che in generale la variabile t non rappresenti un tempo ma semplicemente una coordinata parametrica), nel qual caso si parla di interpolazione di traccia o di percorso.

Come abbiamo visto nei capitoli precedenti, una traccia può essere espressa da una funzione implicita nella forma $C(x) = 0$ equazione scalare con argomento vettoriale x, o in forma parametrica come $x = f(s)$, funzione vettoriale di un parametro scalare s. In questo capitolo tratteremo strumenti per la definizione di traiettorie, cioè della forma $x = f(t)$ e non di tracce.

Diverse varianti del problema, poi, possono richiedere che la funzione f appartenga a particolari classi (un requisito comune ad esempio è che $f \in C^2$, cioè che abbia velocità ed accelerazioni continue e finite, che la derivata della

traiettoria nei punti x_i possa essere assegnata, che una o più derivate della traiettoria siano limitate, etc..

Un importante aspetto del problema della generazione di traiettorie è legato alla possibile presenza di vincoli (ad esempio di posizione, velocità, accelerazione o jerk) che possono essere definiti nello spazio delle coordinate dell'end-effector, nello spazio delle coordinate di giunto, o in entrambi. Il problema può risultare particolarmente ostico quando la traiettoria è specificata nello spazio operativo, ma i vincoli sono specificati nello spazio dei giunti. A tal fine ricordiamo la fondamentale importanza delle relazioni di cinematica diretta e le loro versioni differenziali che dipendono dal jacobiano e dalle sue derivate, cioè

$$x = Q(q) \tag{1.1}$$

$$\dot{x} = \frac{\partial Q}{\partial q}\frac{dq}{dt} = J(q)\dot{q}$$

$$\ddot{x} = \dot{J}(q,\dot{q})\dot{q} + J(q)\ddot{q} \tag{1.2}$$

$$\dddot{x} = \ddot{J}(q,\dot{q},\ddot{q})\dot{q} + 2\dot{J}(q,\dot{q})\ddot{q} + J(q)\dddot{q}$$
$$\text{etc...} \tag{1.3}$$

I metodi per una corretta gestione dei limiti nello spazio dei giunti durante la definizione di traiettorie o tracce nello spazio operativo sono sostanzialmente 2. Il primo deriva da un opportuno utilizzo delle eventuali ridondanze del manipolatore, e consiste nel considerare il problema di inversione della cinematica come un task in un framework di inversione cinematica a priorità di task[3, 4, 5, 6], dove gli altri task (a priorità maggiore) sono definiti in modo da non violare i limiti suddetti.

Il secondo, consiste nella scalatura dinamica della traiettoria tramite la scelta di una opportuna legge oraria. Data infatti una traiettoria nella forma $x = f(t)$, è possibile definire una nuova traiettoria $x = f'(t) = f(s(t))$, che definisce lo stesso percorso utensile (la stessa traccia) ma con una differente legge oraria $s(t)$. La funzione $s(t)$ può essere soggetta a diversi vincoli, a seconda della applicazione, ma in generale dovrà essere sempre continua, monotona non-decrescente, e derivabile un numero opportuno di volte.

L'introduzione di $s(t)$ permette di trasformare le equazioni in (1.3) in

$$x' \ldots = \frac{\partial K}{\partial q}\frac{dq}{dt} = J(q)\dot{q} \tag{1.4}$$

$$x'' \ldots^2 + x' \ddots = \dot{J}(q,\dot{q})\dot{q} + J(q)\ddot{q} \tag{1.5}$$

etc...

Dove la comparsa dei vari coefficienti $\dot{s}, \ddot{s}$, può essere usata per scalare opportunamente le varie $q, \dot{q}, \ddot{q}$ etc, a parità di percorso eseguito, compatibilmente con l'applicazione. Se ad esempio $s(t) = \alpha t$ con α constante, e considerando per semplicità solo le velocità, si ottiene

$$x'\alpha = J(q)\dot{q}\,. \tag{1.6}$$

Capitolo 2

Generazione di Traiettorie per Interpolazione

2.1 Interpolazione Polinomiale

Concentrandosi per un attimo sul problema dell'inseguimento di traiettoria, e limitando per semplicità la dimensione di $x \in \mathbb{R}$, un importante risultato è dato dal fatto che dati $N+1$ punti (t_i, x_i), nello spazio dei polinomi esiste ed è unico il polinomio di grado N

$$p(t) = a_0 + a_1 t^1 + a_2 t^2 + ... + a_N t^N \tag{2.1}$$

tale che $p(t_i) = x_i \forall i$. Questo risultato si può dimostrare semplicemente costruendo il sistema di equazioni

$$\begin{cases} a_0 + a_1 t_0^1 + a_2 t_0^2 + ... + a_N t_0^N &= x_1 \\ a_0 + a_1 t_1^1 + a_2 t_1^2 + ... + a_N t_1^N &= x_1 \\ a_0 + a_1 t_2^1 + a_2 t_2^2 + ... + a_N t_2^N &= x_2 \\ ... &= ... \\ a_0 + a_1 t_N^1 + a_2 t_N^2 + ... + a_N t_N^N &= x_N \end{cases} \tag{2.2}$$

che è un sistema lineare nelle variabili a_j, con matrice caratteristica uguale alla matrice di Vandermonde, che come noto ha determinante sempre diverso da zero per t_i distinti.

Il sistema (2.2) fornisce anche un metodo semplice per calcolare la soluzione del problema, ma sfortunatamente, come noto dai corsi di calcolo numerico, la matrice di Vandermonde ha la tendenza ad essere mal-condizionata, rendendo perciò la ricerca della soluzione tramite l'inversione del sistema (2.2) numericamente instabile.

Un metodo numericamente stabile che conduce alla medesima soluzione è quello che si basa sulla costruzione dei polinomi di Lagrange. Dato l'insieme degli $N+1$ punti t_i, detti anche nodi, si costruiscono i corrispondenti polinomi di Lagrange di grado N come

$$l_i = \prod_{j=0 j\neq i}^{N} \frac{t - t_j}{t_i - t_j} \,. \tag{2.3}$$

Si può notare come gli $N+1$ polinomi, tutti di grado N, siano tutti linearmente indipendenti tra loro, ed in particolare come per costruzione $l_i(t_i) = 1 \forall i$ e $l_i(t_j) = 0 \forall i \neq j$. Questo ci permetti di costruire la soluzione $p(t)$ semplicemente come

$$p(t) = \sum_{i=0}^{N} x_i l_i(t) \,. \tag{2.4}$$

Sfortunatamente, l'interpolazione polinomiale non è esente da problemi. Uno tra tutti il fatto che, nonostante lo spazio dei polinomi sia denso nello spazio delle funzioni continue utilizzando la norma infinito, ipotizzando che i nodi della traiettoria siano estratti da una funzione ideale $\hat{f}(t)$, a cui vorremmo tendere all'aumentare del numero dei nodi, non si può garantire che ci sia convergenza uniforme di $p(t)$ a $\hat{f}(t)$ all'aumentare di N. Questo implica che al di fuori dei nodi, l'errore $p(t) - \hat{f}(t)$ possa non essere limitato in norma, ed in genere si osserva (e si può dimostrare) come questo si tenda a verificare negli estremi della funzione, cioè tra il primo ed il secondo nodo e tra il penultimo e l'ultimo.

Un esempio eclatante di questo fenomeno si ha cercando di interpolare la funzione di Runge

$$f(t) = \frac{1}{1 + t^2} \tag{2.5}$$

nell'intervallo $[-1, 1]$. In particolare si può dimostrare che scegliendo punti equidistanti, all'aumentare del numero dei punti (e quindi del grado del polinomio interpolante), il massimo errore $\max |f(t) - p(t)|$ nell'intervallo cresce arbitrariamente.

Una riduzione del massimo errore $p(t) - \hat{f}(t)$, si può avere se la scelta dei nodi è libera ed è possibile utilizzare i cosiddetti polinomi di Chebyshev (la cui formulazione è facilmente reperibile in letteratura). Tuttavia neanche questi danno garanzie sulla convergenza di $p(t)$ a $\hat{f}(t)$.

Un'altra forma in cui può essere posto il problema dell'interpolazione, è quello di interpolare la traiettoria non soltanto nei punti x_i, agli istanti t_i, ma di volerlo fare con una derivata assegnata $\dot{x}_i$. Questo problema, noto anche

come problema di interpolazione osculatoria, si risolve in maniera numericamente robusta con una costruzione simile a quella di (2.4), ma utilizzando i polinomi di Hermite, costruiti a partire da quelli li Lagrange. Questi sono divisi in due famiglie, una $h_i(t)$ costruita per interpolare le ordinate, simile ai polinomi di Lagrange ma con derivata nulla in tutti i nodi, e una $\hat{h}_i(t)$ costruita per interpolare le derivate e caratterizzata da ordinata nulla in tutti i nodi e derivata nulla in tutti i nodi tranne quello i-esimo, dove presentano derivata unitaria. Si lascia per esercizio verificare le suddette proprietà, e si riportano le formule di costruzione di entrambe le famiglie, che sono:

$$\begin{cases} h_i(t) = \left[1 - 2l_i'(t_i)(t - t_i)\right] l_i^2(t) \\ \hat{h}_i(t) = (t - t_i) l_i^2(t)\,, \end{cases} \tag{2.6}$$

e la formula di interpolazione osculatoria che risulta essere semplicemente

$$p(t) = \sum_{i=0}^{N} x_i h_i(t) + \dot{x}_i \hat{h}_i(t)\,. \tag{2.7}$$

2.2 Spline

Oltre all'assenza di garanzie di convergenza uniforme del polinomio interpolante alla funzione ideale $\hat{f}(t)$ e alla tendenza del polinomio interpolante ad avere un comportamento molto oscillante vicino ai nodi più esterni, la costruzione di una traiettoria che interpoli molti punti presenta anche dei problemi pratici, in particolare (i) la complessità via via crescente nel costruire la base di polinomi da utilizzare per l'interpolazione (che ricordiamo essere di grado N per $N+1$ punti, o di grado $2N+1$ nel caso dell'interpolazione osculatoria) e (ii) il fatto che ogni nodo influisca in modo additivo a tutto il polinomio interpolante risultante, anche arbitrariamente lontano da esso.

Una idea semplice per aggirare molti di questi ostacoli è quella di scomporre il problema in molte istanze più piccole, generare cioè la traiettoria raccordando tante traiettorie più piccole, definite negli intervalli (t_i, t_{i+1}) come polinomi di grado generalmente basso. La traiettoria sarà perciò definita come

$$P(t) = p_i(t) \;\text{ per }\; t \in (t_{i-1}, t_i] \;\text{ e }\; i \in [1, N]\,, \tag{2.8}$$

cioè da N polinomi $p_i(t)$, definiti ognuno nel suo intervallo $t \in (t_{i-1}, t_i]$, e tali da interpolare ognuno soltanto due nodi, cioè

$$p_i(t_{i-1}) = x_{i-1} \;\text{ e }\; p_i(t_i) = x_i\,. \tag{2.9}$$

Questa tecnica porta alla definizione di quella che viene chiamata in genere *spline*, cioè curva ottenuta raccordando polinomi. Notiamo fin da subito che la soluzione più semplice che rispetta i vincoli sinora imposti è quella in cui i segmenti $p_i(t)$ si riducono tutti a segmenti rettilinei (cioè polinomi di grado 1). Dato che questa soluzione è spesso non soddisfacente, si aggiunge in genere a (2.8) il requisito che la soluzione $P(t)$ sia continua ed ammetta derivata di ordine m, che sia cioè di classe C^m. Così facendo otteniamo quella che si definisce come una C^m-spline. Spesso si parla semplicemente di C-spline, cioè di curve appartenenti alla classe C^1, per le quali si richiede la continuità dei polinomi e delle loro derivate prime nei punti di raccordo, cioè che

$$p_i(t_i) = p_i + 1(t_i) = \quad \text{e} \quad p_i'(t_i) = p_i' + 1(t_i)\,. \tag{2.10}$$

Si noti che, a differenza del caso di interpolazione osculatoria, non si sta imponendo il valore della derivata della funzione $P(t)$ nei nodi, ma se ne sta richiedendo soltanto la continuità.

In generale, una C^m-spline, cioè di grado m, che interpola $N+1$ valori nodali x_i nei nodi temporali t_i, sarà formata dal m polinomi $p_i(t)$, definiti ognuno nell'intervallo $(t_{i-1}, t_i]$. Pertanto gli m polinomi saranno definiti da $(m+1)N$ parametri. L'interpolazione continua degli x_i impone $2N+2$ vincoli, mentre l'esistenza della derivata k-esima impone ulteriori $(N-1)k$ vincoli (cominciando a contare da $k=1$). Pertanto i gradi di libertà di progetto sono

$$\begin{aligned} \#GDL &= (m+1)N - [2N+2+(N-1)k] \\ &= (m+1)N - [2(N+1)+(N-1)k] \\ &= (m+1)N - [2(N-1)+2+(N-1)k] \\ &= (m+1)N - (N-1)(k+2) - 2\,. \end{aligned} \tag{2.11}$$

La scelta di $m = k+1$ (quindi $m+1 = k+2$) permette la soluzione per qualunque N (per valori piccoli potrebbero essere possibili più soluzioni) e implica che i gradi di libertà residui siano

$$\begin{aligned} \#GDL &= (k+2)N - (N-1)(k+2) - 2 \\ &= (k+2)(N-1) + (k+2) - (k+2)(N-1) - 2 \\ &= k\,. \end{aligned} \tag{2.12}$$

Una scelta molto comune di interpolazione spline è quella con le spline cubiche, anche dette spline naturali, dove si chiede che la funzione $p(t) \in C^2$, imponendo quindi la continuità di posizioni velocità ed accelerazioni nei nodi. Si può dimostrare che la soluzione al problema di determinare o coefficienti

dei polinomi per le spline cubiche è in genere ben condizionato perché può essere descritto come un sistema lineare con matrice tri-diagonale. Restano due gradi di libertà nella costruzione della spline, che possono essere risolti, ad esempio, (i) assegnando la velocità iniziale e finale della curva, oppure (ii) assegnando le accelerazioni iniziale e finale della curva (condizioni chiamate naturali), o (iii) nel caso di traiettorie cicliche, dove cioè $x_0 = x_N$, imponendo la continuità di velocità ed accelerazione anche tra il primo e l'ultimo nodo.

2.3 Curve di Bezier

Le curve di Bezier sono uno strumento per la costruzione di segmenti polinomiali adatti a definire segmenti di traiettoria. Per definire una traiettoria che passi da più punti è possibile raccordare più curve di Bezier in maniera opportuna. Il vantaggio delle curve di Bezier risiede in una serie di proprietà che le rende di semplice utilizzo e costruzione, e dal comportamento intuitivo.

Una curva di Bezier nasce come generalizzazione di una interpolazione lineare. Dati due punti (t_i, x_i) e (t_f, x_f), il polinomio che interpola linearmente i due si può scrivere facilmente come

$$\begin{aligned} p_l(x) &= x_i + \tfrac{(t-t_i)}{(t_f-t_i)}(x_f - x_i) = x_i \left(1 - \tfrac{(t-t_i)}{(t_f-t_i)}\right) + x_f \left(\tfrac{(t-t_i)}{(t_f-t_i)}\right) \\ = p_l(s) &= x_i(1-s) + x_f(s)\,, \end{aligned} \tag{2.13}$$

dove nell'ultimo passaggio si è definita per comodità la variabile $s = (t - t_i)/(t_f - t_i)$, riguardo la quale è facile notare come, per costruzione, sia sempre compresa nell'intervallo $[0, 1]$. L'ultimo passaggio del sistema (2.13), evidenzia la scrittura di $p_l(t) = p_l(s)$ come *combinazione convessa* dei punti x_i e x_f. Prendiamo ora 3 punti distinti, x_1, x_2 e x_3, e applichiamo in maniera ricorsiva la costruzione precedente. Costruiamo dapprima x_{12} e x_{23} come

$$\begin{aligned} x_{12}(s) &= x_1(1-s) + x_2(s) \\ x_{23}(s) &= x_2(1-s) + x_3(s)\,, \end{aligned} \tag{2.14}$$

e combiniamo poi gli stessi due punti con lo stesso algoritmo, ottenendo

$$\begin{aligned} x_{123}(s) &= x_{12}(1-s) + x_{23}(s) = \\ &= \big(x_1(1-s) + x_2(s)\big)(1-s) + \big(x_2(1-s) + x_3(s)\big)(s) = \\ &= x_1(1-s)^2 + x_2(s)(1-s) + x_2(1-s)s + x_3(s)^2 \\ &= x_1(1-s)^2 + 2x_2(s)(1-s) + x_3(s)^2\,. \end{aligned} \tag{2.15}$$

Questo modo di costruire la curva di Bezier è noto come Algoritmo di de Casteljau.

Possiamo notare come il punto $x_{123}(s)$ risulti appartenere, per costruzione, all'involucro convesso dei punti x_1, x_2 e x_3. Inoltre, un'altra importante proprietà della traiettoria tracciata da $x_{123}(s)$ al variare di $s \in [0,1]$, è che questa parta da x_1 e termini in x_3, come nel caso dell'interpolazione lineare, ma lungi dall'essere inutile, x_2 ha l'effetto di deformare la traiettoria attraendola verso di se. In particolare, se guardiamo alla velocità della traiettoria

$$\begin{aligned}
\tfrac{dx_{123}(s)}{dt} &= \tfrac{\partial x_{123}(s)}{\partial s}\tfrac{ds}{dt} = \\
&= \tfrac{\partial x_1(1-s)^2+2x_2(s)(1-s)+x_3(s)^2}{\partial s}\tfrac{ds}{dt} = \\
&= \left(-2x_1(1-s)+2x_2(s-1)-2x_2(s)+2x_3(s)\right)\tfrac{ds}{dt} = \\
&= \left(2(x_2-x_1)(s-1)+2(x_3-x_2)(s)\right)\tfrac{ds}{dt}
\end{aligned} \tag{2.16}$$

che in uscita dal punto x_1, cioè per $s \to 0$, diventa $2(x_2 - x_1)(1-s)ds/dt \to 2(x_2 - x_1)ds/dt$, allineata al vettore $x_2 - x_1$. Analogamente la velocità in ingresso al punto x_3 è $2(x_3 - x_2)ds/dt$ allineata a $x_3 - x_2$. L'accelerazione è invece costante e pari a $(x_3 - 2x_2 + x_1)ds/dt$.

Infine, una ultima proprietà della di (2.15) è quella di ricordare molto da vicino lo sviluppo di una potenza di un binomio nelle variabili s e $(1-s)$, in questo caso elevato alla potenza di 2.

La semplicità a la bellezza delle curve di Bezier risiede nel fatto che il precedente procedimento si può generalizzare ad $n+1$ punti da x_0 a x_n, detti punti di controllo, tramite una costruzione analoga, mantenendo proprietà sostanzialmente simili alle precedenti. In particolare, dati gli $n+1$ punti, la formula risultante sarà semplicemente

$$p_b(s) = \sum_{i=0}^{n} \binom{n}{i} x_i (s)^i (1-s)^{(n-i)}\,, \tag{2.17}$$

e la traiettoria risultante manterrà sempre le seguenti proprietà

1. la traiettoria per $s \in [0,1]$ sarà sempre contenuta nell'involucro convesso dell'insieme dei punti $\{x_i\}$,.

2. la traiettoria inizierà sempre dal punto x_0 e terminerà al punto x_n,

3. la curva sarà un polinomio di grado n,

4. la derivata della curva sarà anche essa una curva di Bezier di grado $n-1$, pari a

$$\frac{\partial p_b(s)}{\partial s} = \sum_{i=0}^{n-1} \binom{n-1}{i} x'_i (1-s)^i (s)^{(n-1-i)}\,, \tag{2.18}$$

definita dai coefficienti $x'_i = n(a_{i+1} - a_j)$ (con $i \in [0, n-1]$). Da ciò consegue anche che:

5. la generica derivata k-esima della traiettoria sarà sempre una curva di Bezier,

6. la derivata n-esima della traiettoria sarà costante,

7. ed in particolare, la derivata prima uscente dal primo punto sarà sempre allineata con $x_1 - x_0$ e quella entrante nell'ultimo punto sarà allineata con $x_n - x_{n-1}$.

Ricordiamo infine che (2.18) e le sue derivate possono essere riportate nelle coordinate temporali generiche t, tramite le sostituzioni $s = (t - t_i)/(t_f - t_i)$ e $(1-s) = (t - t_f)/(t_i - t_f)$, e le loro derivate.

Grazie alle precedenti proprietà risulta facile e graficamente intuitivo costruire un tratto di curva con le proprietà geometriche desiderate. Per questo motivo le curve di Bezier sono usate molto spesso anche nei programmi di grafica per descrivere linee curve, solitamente mediante la specificazione di 4 punti di controllo che definiscono la partenza e dell'arrivo della curva (x_0 e x_3) e le velocità della curva nei due punti precedenti (allineate ai vettori $x_1 - x_0$ e $x_3 - x_2$). Trasportando queste conoscenze nell'ambito della robotica e della generazione di traiettorie, le curve di Bezier possono costituire uno strumento facile ed efficace per definire il comportamento dei vari segmenti che formano una spline, e per imporre e/o conoscere il comportamento della curva in termini di velocità, accelerazione etc. in corrispondenza dei vari nodi della curva. A tal fine si noti che è importante non confondere i valori nodali della spline $x_0, ..., x_N$, con i punti di controllo della curva di Bezier: se vogliamo costruire una spline raccordando curve di Bezier, i valori nodali andranno infatti a costituire soltanto i punti di controllo iniziale e finale di ogni curva di Bezier, mentre rimangono da specificare il numero di punti di controllo interni e la loro posizione, che possono essere scelti ad esempio per garantire l'appartenenza della spline ad una determinata classe C^m. Per i dettagli su come effettuare tali scelte, si rimanda il lettore a testi specializzati.

2.3.1 Esempio: generazione di traiettorie minimum-jerk con le curve di Bezier.

Una quantità che spesso è utile mantenere sotto controllo (entro certi limiti o, in genere bassa) quando si progetta una traiettoria per un manipolatore è il *jerk* (chiamato anche *scuotimento* in italiano), cioè la derivata terza della posizione $\dddot{x}$. Esistono diversi motivi per desiderare questo, uno dei motivi, richiamato esplicitamente dal nome italiano *scuotimento*, risiede nel fatto che quando si richiede ad un macchinario particolarmente grande di seguire traiettorie con valori di jerk molto elevati, queste tendono ad eccitare la flessibilità (spesso parassita) presente nelle trasmissioni dei motori, e a far appunto scuotere il sistema per colpa delle vibrazioni che si instaurano. Un altro motivo, altrettanto importante, è che esistono studi che dimostrano come anche gli essere umani si muovano in genere secondo traiettorie che tendono a minimizzare il jerk, e pertanto quando un robot si muove lungo traiettorie minimum jerk, i suoi movimenti risultano più dolci e naturali all'aspetto, elementi che ne possono incrementare la predicibilità e l'accettabilità da parte di eventuali persone che possano trovarsi nelle vicinanze dei robot stessi. In particolare, assegnato un intervallo di tempo $[t_0, t_f]$, un punto di partenza x_0 e uno di arrivo x_f, si possono ottenere traiettorie che ricordano i movimenti umani e con valori contenuti di jerk, andando a minimizzare la norma L_2 del jerk durante la traiettoria stessa, cioè

$$J = |\dddot{x}|_{L_2} = \int_{t_0}^{t_f} (\dddot{x})^2 dt \ . \tag{2.19}$$

Notiamo innanzitutto che se lasciassimo libera la scelta della durata della traiettoria t_f, la definizione del problema in termini della norma L_2 (ed in particolare la presenza del quadrato al suo interno), farebbe si che la soluzione degenererebbe in una traiettoria che impiega un jerk infinitesimo per un tempo tendente all'infinito, con costo J tendente a 0. Pertanto le soluzioni di interesse del problema sono quelle in cui t_f è assegnato.

Per motivi simili, anche la posizione iniziale x_0 e quella finale x_f della traiettoria dovranno essere date (e distinte), altrimenti una traiettoria ottima banale sarebbe qualunque traiettoria con posizione costante $x(t) = x_c$, la quale avrebbe jerk sempre identicamente nullo, e costo J totale nullo.

Notiamo infine che, volendo minimizzare $J = J(\dddot{x})$, ed ipotizzando perciò di poter scegliere liberamente l'andamento di $\dddot{x}(t)$, la soluzione ottima dovrà rispettare opportune condizioni al contorno naturali, definite anche in termini delle velocità iniziale $\dot{x}_0$ e finale $\dot{x}_f$ e delle accelerazioni iniziale $\ddot{x}_0$ e finale $\ddot{x}_f$.

Il set completo delle condizioni al contorno $t_f, x_0, \dot{x}_0, \ddot{x}_0, x_f, \dot{x}_f, \ddot{x}_f$, insieme alla (2.19), definiscono completamente il nostro problema variazionale di ricerca di costo minimo nello spazio delle funzioni. I valori delle condizioni al contorno saranno definiti caso per caso dal task che dobbiamo eseguire.

Per trovare una soluzione a (2.19) nello spazio delle soluzioni ammissibili al problema (e cioè che rispettino le condizioni al contorno), vogliamo procedere in maniera analoga a come si minimizza una funzione nello spazio del suo dominio, andando a calcolare la derivata della funzione rispetto al suo argomento e ponendola uguale a 0.

Ma nel caso dell'ottimizzazione di un funzionale, ci muoviamo nello spazio delle funzioni e dobbiamo scrivere la nostra derivata rispetto ad una variazione funzionale. Questo tipo di analisi è sviluppata nel campo del *calcolo variazionale*.

Esempio: una traiettoria minimum jerk come quella definita in [7] si ottiene con una Bezier a 6 punti di controllo. (Completare)

2.3.2 Note finali sulle curve di Bezier

Una ultima proprietà importante delle curve di Bezier è quella di disporre di un metodo numericamente stabile per la loro costruzione. Questo non è basato sulla forma pseudo-binomiale che abbiamo visto e che ci è servita a caratterizzarne le proprietà, ma è proprio l'algoritmo di de Casteljau che abbiamo visto sopra. Inoltre, data la sua generalità, possiamo notare come l'algoritmo di de Casteljau sia applicabile in modo più generale della forma pseudo-binomiale, basandosi sulla applicazione reiterata di interpolazioni lineari tra due punti. Una importante applicazione di ciò è che si possono generare curve di Bezier anche in domini non lineari, ad esempio nello spazio delle rotazioni, ad esempio tramite la applicazione reiterata di interpolazioni SLERP.

2.4 Interpolazione di rotazioni

Ricordiamo che lo spazio delle rotazioni in 3d è $\mathbb{SO}(3)$, il quale non ammette una parametrizzazione in $\mathbb{R}^3$ priva di singolarità. Pertanto, seppure sia possibile utilizzare i precedenti metodi per interpolare rotazioni espresse in termini di angoli di Eulero, le traiettorie interpolanti risultanti non hanno il comportamento che ci si potrebbe aspettare.

Per comprendere questo aspetto, si consideri il semplice caso di una interpolazione lineare tra due orientazioni generiche $\{0\}$ e $\{1\}$. Si può facilmente risalire alla rotazione relativa tra i due frame, ad esempio utilizzando il forma-

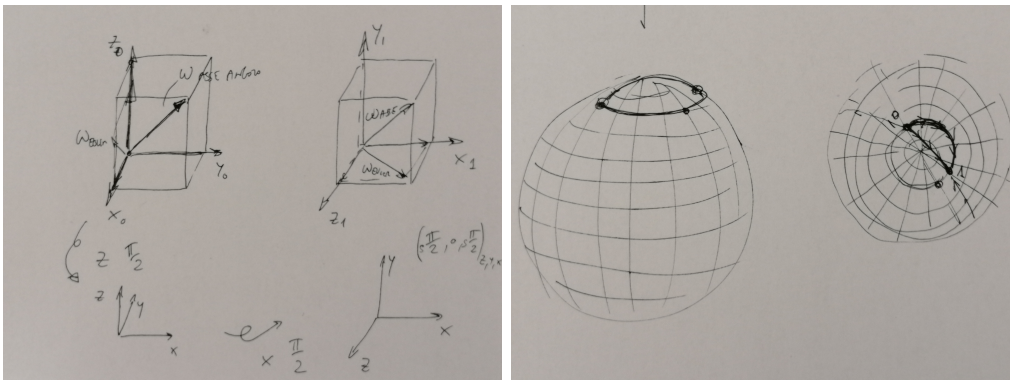

Figura 2.1: Perché l'interpolazione nello spazio degli angoli di Eulero non funziona come ci si aspetterebbe.

lismo delle matrici o quello dei quaternioni. Supponiamo di voler interpolare linearmente (forzando un po' il concetto, data la non linearità dello spazio $\mathbb{SO}(3)$) la rotazione tra queste due orientazioni in funzione di un parametro $s \in [0, 1]$. Ricordando che una data rotazione è sempre esprimibile con una parametrizzazione asse-angolo, cioè come una rotazione definita da un vettore unitario v e da una rotazione attorno ad esso di un angolo $\bar{\alpha}$ di modulo non superiore a π radianti, è lecito (e solitamente è ciò che ci si aspetta) domandare che l'equivalente dell'interpolazione lineare corrisponda a "percorrere" questa rotazione mantenendo fisso l'asse di rotazione e variando linearmente il parametro $\alpha \in [0, \bar{\alpha}]$.

Questo comportamento (la cui descrizione fornisce anche un importante strumento implementativo), non è mantenuto se si interpola nello spazio degli angoli di Eulero, dove non è vero, in genere, che l'asse di rotazione sia mantenuto costante (salvo che per casi molto banali e specifici).

Scegliamo ad esempio come angoli di Eulero le rotazioni attorno agli assi (z, y', x''). Ipotizziamo che il frame $\{0\}$ sia allineato al nostro frame base, mentre il frame $\{1\}$ sia generato da due rotazioni successive, una attorno all'asse z di $\pi/2$ radianti, e poi una seconda di altri $\pi/2$ attorno all'asse x''. La matrice di rotazione relativa è,

$$R_1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{2.20}$$

mentre la parametrizzazione (z, y', x'') è $(\pi/2, 0, \pi/2)$.

Come è evidente da figura 2.1, la rotazione relativa che porta il primo frame nel secondo, è una rotazione attorno all'asse $v_0 = (1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})^T$ di

$2\pi/3$ radianti. La traiettoria più corta nello spazio delle rotazioni, perciò, corrisponde ad una rotazione attorno a v_0 di un angolo $\alpha(s) = s2\pi/3$. La velocità angolare di tale traiettoria sarà allineata con v_0 e uguale a $\omega_{AA} = 2\pi/3ds/dt$. Invece, interpolando linearmente gli angoli di Eulero, otteniamo una traiettoria costituita da rotazioni istantanee attorno ad un asse non fisso, ad esempio per $s \to 0$ la velocità sarà allineata con $v_{Euler}(0) = (1/\sqrt{2}, 0, 1/\sqrt{2})^T$, mentre per $s \to 1$ la velocità sarà allineata con $v_{Euler}(1) = (1/\sqrt{2}, 1/\sqrt{2}, 0)^T$.

Questo problema è analogo a quello che si ha quando si parametrizza una sfera (ad esempio il globo terrestre, vedi sempre figura 2.1) con paralleli e meridiani. La strada più corta tra due punti p_0 e p_1 (l'interpolazione lineare) non si ottiene interpolando linearmente paralleli e meridiani, ma muovendosi lungo la geodesica, cioè la circonferenza massima passante per i due punti.

Questo problema si risolve con un algoritmo noto come SLERP, Sferical Linear intERPolation, la cui espressione è

$$p(p_0, p_1, s) = \frac{sin((1-s)\Omega)}{sin(\Omega)} p_0 \frac{sin(s\Omega)}{sin(\Omega)} p_1 \tag{2.21}$$

dove l'angolo Ω è l'angolo sotteso dai due punti nel centro della sfera, che si può calcolare a partire dalla nota proprietà del prodotto scalare per cui

$$< p_0, p_1 >= |p_0| \cdot |p_1| \cos\Omega \,. \tag{2.22}$$

Fortunatamente, lo stesso algortimo può essere applicato alle rotazioni in $\mathbb{SO}(3)$, ricordando che è sempre possibile rappresentare una rotazioni tramite quaternioni di modulo unitario, cioè quaternioni vincolati a giacere sulla superficie di una sfera, come lo erano i punti p_0 e p_1 del precedente esempio.

Dati perciò due quaternioni di modulo unitario q_0 e q_1, che descrivono due orientazioni in $\mathbb{SO}(3)$ espresse rispetto ad un frame base, la (2.21), può essere applicata per generare un quaternione interpolante $q(s)$ che descrive il percorso più diretto nello spazio delle rotazioni, andando da q_0 a q_1. Come unica accortezza, dobbiamo ricordare che la corrispondenza tra $\mathbb{SO}(3)$ e i quaternioni unitari è duplice (nel senso che un quaternione ed il suo negato descrivono entrambi la stessa rotazione), ciò comporta che al momento di generare lo SLERP quaternionico, l'outuput di (2.21) potrebbe corrispondere tanto alla rotazione più breve, quanto a quella nella direzione opposta. Questo aspetto è però facilmente risolvibile andando a guardare il segno di $cos\Omega$ (e quindi del prodotto scalare tra q_0 e q_1): nel caso in cui questo risulti negativo otterremmo la rotazione nella direzione sbagliata, pertanto sarà sufficiente invertire il segno di q_1 per risolvere il problema.

2.5 B-Spline e NURBS

2.5.1 B-Spline (completare)

Un ulteriore modo per definire una spline, che riunisce alcuni delle caratteristiche e dei vantaggi delle C-spline e delle curve di Bezier, è attraverso la costruzione delle cosiddette B-spline, cioè "'Basis Splines"' ovvero spline di base.

L'idea di fondo, mutuata dalle curve di Bezier ma anche dalla costruzione del polinomio interpolante con il metodo di Lagrange, è quella di costruire la traiettoria $s(t)$ come somma pesata di $N+1$ spline di base $B_i(t)$.

Una seconda idea, che in qualche modo si ispira alle C-spline, è quella di assegnare un grado massimo p ai polinomi che costituiscono le curve di base, che saranno quindi indicate come $B_i^p(t)$

Pertanto, la traiettoria sarà definita come

$$s(t) = \sum_{i=0}^{N} x_i B_i^p(t) \tag{2.23}$$

Inoltre, sempre mutuando dalle curve di Bezier, si vuole costruire la curva come combinazione convessa degli $N+1$ punti x_i. Ciò sarà garantito imponendo che le $B_i^p(t)$ costituiscano una *partizione dell'unità*, e cioè che

$$\sum_{i=0}^{N} B_i^p(t) = 1 \ \forall t \text{ dove è definita la curva } s(t)\,. \tag{2.24}$$

C'è però un aspetto per cui le B-spline differiscono dalle C-spline e dalle curve di Bezier. Questo è l'interpretazione degli $N+1$ punti x_i. Questi infatti non saranno obbligatoriamente ne dei way-point come per le C-spline, ne dei punti di controllo come per le curve di Bezier, ma potranno assumere ciascuna delle due nature in maniera intercambiabile e modulabile, a seconda delle necessità di progetto.

Per ottenere questo risultato, viene svincolata la corrispondenza uno ad uno tra il numero di nodi e il numero di punti x_i. Pertanto le B-spline $B_i^p(t)$ saranno definite a partire da un vettore di N_k+1 tempi nodali (o semplicemente nodi) $\mathbf{t} = (t_0, t_1, t_{N_k})^T$ con in genere $N_k > N$. Inoltre viene ammorbidito uno dei vincoli fondamentali che avevamo incontrato finora, difatti avremo che

$$t_i \leq t_{i+1} \tag{2.25}$$

(nota bene nella (2.25) la presenza del segno di minore uguale a posto di quello di minore).

Per ottenere mettere insieme tutte queste caratteristiche, so costruisce in maniera iterativa un set di B-spline $B_i^p(t)$ di grado p (o ordine $p+1$) basandosi sui due assiomi seguenti. Al grado 0:

$$B_i^0(t) = \begin{cases} 1 & \text{se } t_i \leq t < t_{i+1} \\ 0 & \text{altrove} \end{cases} \tag{2.26}$$

mentre ai gradi successivi si costruiscono le B-spline di grado p tramite combinazione lineare di due B-spline di grado inferiore $(p-1)$ come

$$B_i^p(t) = \frac{t - t_i}{t_{i+p} - t_i} B_i^{p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1}^{p-1}(t)\,. \tag{2.27}$$

Sfruttando la costruzione iterativa delle $B_i^p(t)$ è facile dimostrare come l'insieme delle $B_i^p(t)$ di un dato grado p, definito da un inseme di tempi nodali $\mathbf{t}$ costituisca una partizione dell'unità per tutti i tempi $t \in [t_0, t_{N_k}]$.

2.5.2 NURBS

(da completare) Vedasi appendice B [2].

2.6 Note finali sulla generazione di traiettorie per interpolazione

Due importantissime caratteristiche, condivise da tutte le spline (quindi dalle Curve di Bezier costruite in spazi cartesiani, dalle C-spline, dalle B-spline ma anche dai polinomi di Lagrange che possono essere visti come una forma degenere di spline con un unico polinomio definito in tutto l'intervallo da interpolare) sono le seguenti

1. Trasformazione Affine: Dato che tutte le precedenti famiglie di traiettorie sono esprimibili come somme pesate di way-points $\{x_i\}$ moltiplicati per funzioni del tempo, cioè come

$$f(t) = \sum_{i=0}^{N} x_i f_i(t)\,, \tag{2.28}$$

data una qualunque trasformazione affine $T()$ tale che

$$x' = T(x) = Ax + b \tag{2.29}$$

con A e b matrice e vettore di dimensione opportuna, è possibile ottenere la trasformazione della traiettoria $f(t)$ semplicemente trasformando i punti $\{x_i\}$ e costruendo una nuova traiettoria con le stesse funzioni $f_i(t)$, cioè:

$$f'(t) = T(f(t)) = A\left(\sum_{i=0}^{N} x_i f_i(t)\right) + b = \sum_{i=0}^{N} (Ax_i + b) f_i(t) = \sum_{i=0}^{N} T(x_i) f_i(t)\,. \tag{2.30}$$

Questa caratteristica rende facile la traslazione, rotazione e scalatura di traiettorie già costruite.

2. Scalatura Temporale: Se si scala linearmente il vettore dei tempi nodali $\mathbf{t}' = \alpha\mathbf{t}$, la i-esima derivata della traiettoria viene scalata di un fattore $\frac{1}{\alpha^i}$. Ad esempio, se si raddoppiano i tempi nodali ($\alpha = 2$), le velocità della traiettoria dimezzano e le accelerazioni diventano un quarto delle precedenti.

Capitolo 3

Interpolazione dinamica

Un approccio alternativo per trasformare una sequenza di punti in una traiettoria ci è suggerito dalla teoria dei sistemi dinamici. Guardando ai punti da interpolare come una sequenza di riferimenti a gradino da inseguire, è possibile costruire una funzione che interpoli quasi esattamente tali punti e garantisca valori finiti di velocità, accelerazione, fino alla derivata desiderata, tramite un filtro dinamico di ordine opportuno. Ad esempio con un modello del primo ordine

$$\dot{x} = u = K_P(x_i - x) \text{ per } t \in [t_{i-1}, t_i) \,, \tag{3.1}$$

o con un modello del secondo ordine (con feed forward di velocità, ipotizzando che la velocità al punto finale sia nota)

$$\ddot{x} = u = K_P(x_i - x) + K_D(\dot{x}_i - \dot{x}) \text{ per } t \in [t_{i-1}, t_i) \,. \tag{3.2}$$

Questo approccio ha il particolare vantaggio di avere la naturale propensione ad essere applicato online, consentendo ad esempio una eventuale orrezione della traiettoria in anello chiuso. Notiamo tuttavia che al contrario dei metodi proposti in precedenza, le tecniche descritte da (**??**) e (**??**) introducono sistematicamente un piccolo errore nell'inseguimento dei punti, dovuto alla dinamica dell'errore dei filtri. Questo errore può essere ridotto con opportuna taratura dei guadagni, ma nel caso in cui un tale errore, seppur piccolo, non fosse accettabile, è possibile estendere la tecnica precedente in modo da ridurre questo errore a zero esattamente mediante l'applicazione di un sistema di controllo non lineare (di tipo bang-bang) al sistema dinamico del filtro. Seguendo questa strada, in particolare, si possono anche introdurre abbastanza facilmente vincoli di massima velocità ed accelerazione nel filtro, ottenendo un sistema dinamico capace di generare traiettorie a tempo minimo, limitate in accelerazione e velocità.

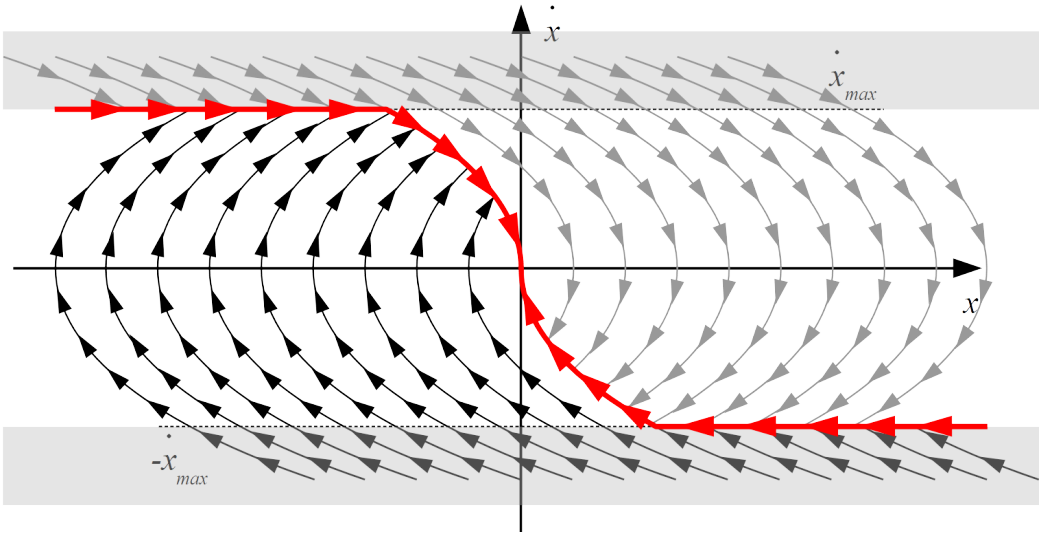

Figura 3.1: diagramma delle fasi per sistema dinamico switching che controlla in tempo ottimo con vincoli di velocità e accelerazione massime.

Trattandosi di una traiettoria bang bang, sappiamo che $\ddot{x} = u$ sarà uguale a $\pm a_{max}$ o nullo. Integrando la traiettoria per $\ddot{x} = a_{max}$, ed esplorando lo spazi delle fasi, possiamo vedere come otteniamo un fascio di traiettorie paraboliche con equazione

$$x = \frac{1}{2a_{max}}\dot{x}^2 + x_0 \tag{3.3}$$

ed analogamente per $\ddot{x} = -a_{max}$ otteniamo $x = -\frac{1}{2a_{max}}\dot{x}^2 + x_0$.

Andando a disegnare lo spazio delle fasi, è facile definire per ispezione il controllo, che risulterà

$$\ddot{x} = \begin{cases} \text{if } x < -\frac{1}{2a_{max}}\dot{x}|\dot{x}| \begin{cases} \text{if } \dot{x} < \dot{x}_{max} & \text{then } a_{max} \\ \text{if } \dot{x} = \dot{x}_{max} & \text{then } 0 \\ \text{if } \dot{x} > \dot{x}_{max} & \text{then } -a_{max} \end{cases} \\ \text{if } x = -\frac{1}{2a_{max}}\dot{x}|\dot{x}| \begin{cases} \text{if } \dot{x} > 0 & \text{then } -a_{max} \\ \text{if } \dot{x} = 0 & \text{then } 0 \text{ (obiettivo raggiunto)} \\ \text{if } \dot{x} < 0 & \text{then } a_{max} \end{cases} \\ \text{if } x > -\frac{1}{2a_{max}}\dot{x}|\dot{x}| \begin{cases} \text{if } \dot{x} < -\dot{x}_{max} & \text{then } a_{max} \\ \text{if } \dot{x} = -\dot{x}_{max} & \text{then } 0 \\ \text{if } \dot{x} > -\dot{x}_{max} & \text{then } -a_{max} \end{cases} \end{cases} \tag{3.4}$$

Il precedente risultato, notevole nella sua semplicità, può non essere sempre sufficiente perché non garantisce limiti ne continuità sul jerk della traiettoria. Esistono in letteratura estensioni [8] basate su sistemi di ordine 3 che colmano tale mancanza.

Capitolo 4

Generalizzazione di traiettorie

4.1 Dynamic Movement Primitives

Una diversa evoluzione di approcci basati sul concetto di filtro dinamico, che per altri versi si ispira anche ad alcune proprietà delle curve di Bezier è l'approccio basato su Dynamic Movement Primitives. Questa strategia, sviluppata recentemente [9], nasce con il duplice obiettivo di trasformare un insieme di punti in una funzione traiettoria e saper trasformare una traiettoria in un set di informazioni che possa facilmente essere utilizzato per *generalizzare* la traiettoria dimostrata al variare dei punti iniziali e finali. Per fare ciò, la tecnica delle DMP descrive la traiettoria mediante un sistema dinamico del secondo ordine, che si muove in maniera elastica smorzata dal punto iniziale al punto finale, perturbato da una forza f che ne determina l'andamento.

$$\begin{cases} \tau\dot{v} &= K(g-x) - D\dot{x} + f(s), \\ \tau\dot{x} &= v \end{cases} \tag{4.1}$$

dove τ è un fattore di scala temporale, K rappresenta la costante elastica, D è il guadagno di smorzamento e g il goal. La funzione $f(s)$ è definita come

$$f(s) = \frac{\sum_{i=0}^{n} \omega_i \psi_i(s)}{\sum_{i=0}^{n} \psi_i(s)} s(g - x_0), \tag{4.2}$$

- x_0 rappresenta la condizione iniziale del sistema
- g il goal
- s è una variabile di fase che evolve in base ad un cosiddetto *sistema canonico*

$$\tau\dot{s} = -\alpha s, \tag{4.3}$$

la cui condizione iniziale è $s = 1$ e converge asintoticamente a 0 (garantendo la stabilità del sistema (4.1)), mentre α è una costante positiva.

- La sommatoria $\sum_{i=0}^{n} \omega_i \psi_i(s)$ rappresenta una combinazione lineare di funzioni Gaussiane

$$\psi_i(s) = e^{(-h_i(s-c_i)^2)} \tag{4.4}$$

centrate in

$$c_i = e^{((-\alpha d_i)/\tau)} \text{ con } d_i = i\frac{t_f - t_0}{n}\,, \tag{4.5}$$

con varianza

$$h_i = n/c_i^2\,. \tag{4.6}$$

pesata dai pesi ω_i.

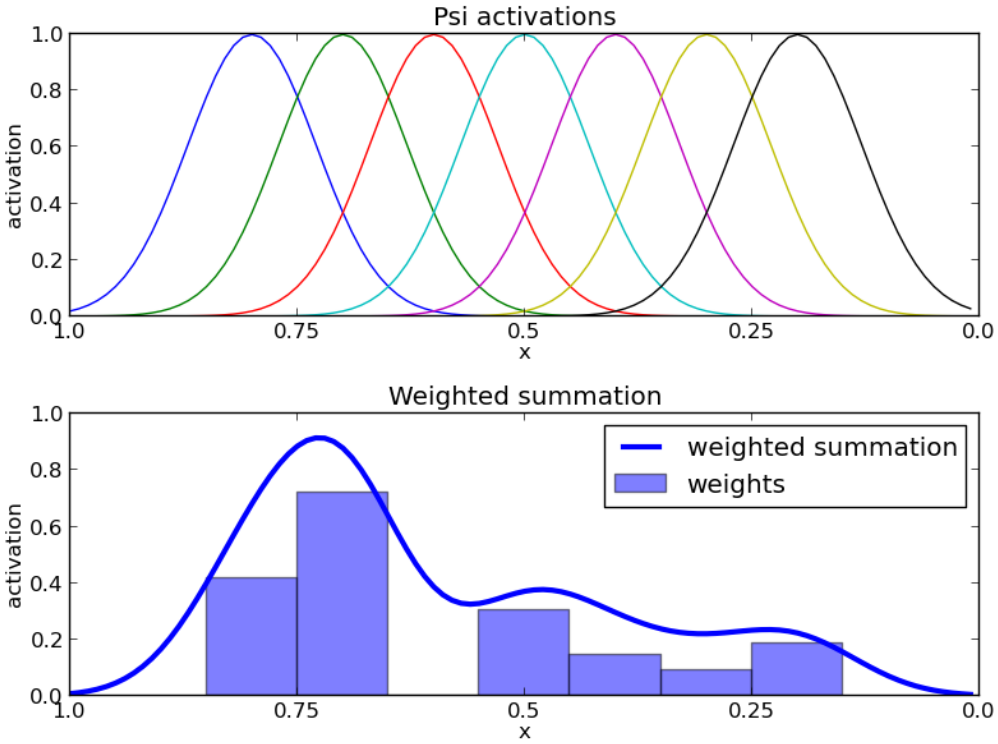

Figura 4.1: Combinazione lineare pesata di funzioni Gaussiane

Il sistema complessivo formato dalle (4.1), (4.3) e (4.2), gode di 3 importanti proprietà:

- il sistema converge sempre al goal g nella sola ipotesi che i pesi ω_i siano limitati, dato che $f(s)$ svanisce esponenzialmente.

- data una qualunque traiettoria sufficientemente smooth, esiste un set di pesi ω_i che la riproduce
- le equazioni sono spazio- e tempo-invarianti, nel senso che i movimenti sono autosimilari per cambiamenti di posizione iniziale, finale e per scalatura dei tempi, senza cambiare i pesi ω_i.

Dopo aver descritto l'idea di come le DMP possano generare una traiettoria partendo da un sistema del secondo ordine perturbato da un termine esterno f, adesso affrontiamo il seguente problema: come posso calcolare la funzione f affinché il sistema (4.1) segua una traiettoria data?

Definiamo con x_{des} la traiettoria desiderata, da cui possiamo calcolare $\dot{x}_{des}$ e $\ddot{x}_{des}$. Sostituendo queste in (4.1) otteniamo:

$$f_{des} = \ddot{x}_{des} - K(g - x_{des}) + D\dot{x}_{des} \tag{4.7}$$

dove g è l'ultimo valore della traiettoria desiderata. Infine è possibile calcolare i pesi ω_i che minimizzano l'errore quadratico medio

$$J = \int_s (f_{des} - f(s))^2, \tag{4.8}$$

la cui soluzione è data da

$$\omega_i = \frac{\int_s s\psi_i(s) f_{des}(s)}{\int_s \psi_i(s) s^2} , \tag{4.9}$$

o approssimando l'integrale con una sommatoria in corrispondenza di un vettore di campioni $\mathbf{s}$ da

$$\omega_i = \frac{s^T \Psi_i \mathbf{f}_{des}}{s^T \Psi_i s} , \tag{4.10}$$

dove $\Psi_i = \text{diag}(\psi_i(\mathbf{s}))$, e $\mathbf{f}_{des}$ è il vettore dei campioni di $f_{des}(s)$.

Dopo aver calcolato i pesi ω_i è possibile generare la traiettoria desiderata $\hat{x}_{des}$ con il sistema (4.1) inserendo valori opportuni di x_0, g e τ. Al variare di x_0 e g è possibile generare traiettoria simili a x_{des} ma con diverse condizioni iniziali e finali, e al variare di τ se ne può scalare la temporizzazione.

4.2 Modelli probabilistici impliciti

4.3 Filtrazione di traiettorie e Gaussian Mixture Models

Un altro strumento introdotto recentemente [10] per la definizione di traiettorie, il cui utilizzo è per certi versi duale al precedente, e che mantiene

una certa ispirazione alla generazione di una traiettoria attraverso un filtro dinamico, è quello dei modelli probabilistici per la rappresentazione delle traiettorie. Il loro scopo è quello di risalire ad una traiettoria ideale a partire da una serie di sue realizzazioni imperfette (perché affette, ad esempio, da errori). Come esempio di come questo tipo di strumenti possa essere utilizzato in pratica, si consideri di registrare i movimenti di robot mentre questo viene mosso da un operatore. Questo movimento potrà essere realizzato tramite teach-pendant, o nei robot più moderni e collaborativi trascinando l'end-effector mentre il robot si trova in una modalità di controllo a bassa impedenza, o addirittura tramite interfacce quali sistemi di realtà virtuale o aumentata. Indipendentemente dal sistema di controllo scelto, è lecito assumer che la traiettoria dimostrata possa risentire di imprecisioni dovute a cause quali. la natura imperfetta dei movimenti dell'operatore, gli effetti della dinamica del robot se non compensata esattamente, le interazioni con l'ambiente e probabilmente altri ancora dipendenti dal task.

Una possibilità per ridurre gli effetti di queste imprecisioni sulla traiettoria memorizzata è quello di registrare più volte la traiettoria e risalire, tramite opportuni strumenti, ad una qualche *traiettoria media*, in cui gli effetti delle imprecisioni sono stati *filtrati via*. Questo approccio si basa sull'assunzione che gli effetti delle imprecisioni siano modellabili come qualche tipo di processo stocastico ${}_jn(t)$ a media nulla che si somma ad una traiettoria ideale $x^*(t)$ per generare le varie realizzazioni

$$ {}_jx(t) = x^*(t) + {}_jn(t) \tag{4.11} $$

e puntano a separare la componente costante da quella stocastica. Se ad esempio fosse nota la distribuzione di probabilità del processo stocastico $X(t)$, generatore delle ${}_jx^*(t)$, nell'ipotesi che $X(t) = x^*(t) + N(t)$ con $N(t)$ a media nulla, la traiettoria ideale $x^*(t)$ si avrebbe semplicemente utilizzando l'operatore aspettazione, cioè

$$ x^*(t) = E[X(t)] \tag{4.12} $$

Osservazione Un importante aspetto di questo tipo di modellazione è che la ricostruzione di $X(t)$ porta con se più informazioni rispetto alla sola traiettoria ideale $x^*(t)$. Nell'ipotesi, che tutte le traiettorie dimostrate siano ad esempio ammissibili soluzioni al task robotico che stiamo studiando, considerando la distribuzione di $X(t)$ (o alcune sue caratteristiche come ad esempio la matrice di covarianza $\Sigma(t)$) si possono estrapolare informazioni su quanto la traiettoria stessa possa discostarsi da quella ideale nei vari punti. Questa informazione può essere usata, ad esempio, per ricostruire la presenza di

eventuali vincoli (si pensi ad una strettoia, o ad un muro) e di conseguenza regolare i parametri del controllo del robot (ad esempio la sua impedenza cartesiana).

Più in generale comunque, disponendo di una ricostruzione $\hat{X}(t)$ della vera $X(t)$, la sua covarianza, ed in genere la sua distribuzione di probabilità, ci danno una misura della confidenza con cui stiamo ricostruendo $x^*(t)$.

4.4 Gaussian Mixture Models e Gaussian Mixture Regression

Un modo ancora migliore per fare questa operazione di filtraggio probabilistico, su cui si basa la regressione con *Gaussian Mixture Model* che andremo a vedere tra poco, è quello di considerare i campioni registrati delle traiettorie generate $x_j = [{}_l x(t_i) = {}_l x(iT)^T]^T$ come degli insiemi di punti spazio-temporali, detti *datapoint*. Si considerano cioè i punti $\xi_j = [{}_l t_i, {}_l x_i^T]^T$, e si assume che essi vengano generati da un una variabile casuale $P(t, x) = P(\xi)$, che definisce una distribuzione di probabilità nello spazio $\mathbb{R} \times \mathbb{X}$.

Supponendo di conoscere $P(\xi)$, a questo punto è possibile risalire al processo stocastico generatore della traiettoria ideale passando alla probabilità condizionata $P(x|t)$, che definisce una distribuzione di probabilità nello spazio $\mathbb{X}$, in funzione di un parametro $t \in \mathbb{R}$, e da questo alla traiettoria ideale tramite l'operatore aspettazione $x^*(t) = E[P(x|t)]$.

La scelta di rappresentare il problema "mescolando" dimensioni spaziali e temporali consente di modellare tra le varie imprecisioni possibili anche sfasamenti temporali nell'esecuzione della traiettoria (anticipi e ritardi), un tipo di imprecisione di fatto abbastanza frequente nel caso in cui le traiettoria siano eseguite da un operatore umano. Inoltre, questo estende le considerazioni riportate nella precedente osservazione, includendo la possibilità di stimare anche vincoli di temporizzazione della traiettoria e non soltanto spaziali.

4.4.1 I Gaussian Mixture Model

Un *Gaussian Mixture Model* è un particolare tipo di modello probabilistico. Esso è stato proposto per ricostruire una traiettoria ideale soggiacente ad un set di osservazioni in [10]. Un GMM punta a ricostruire $P(\xi)$ approssimandola come una combinazione finita di K distribuzioni gaussiane, cioè assumendo che

$$P(\xi) \approx \hat{P}(\xi) = \sum_{k=1}^{K} p(k) P_k(\xi|k)\,. \tag{4.13}$$

dove $p(k)$ è una funzione di probabilità categorica discreta definita in $k \in [1, K]$, che ci dice la probabilità che ha un qualunque datapoint ξ di essere generato secondo una di K distribuzioni gaussiane $P_k(\xi|k)$. Ogni $P_k(\xi|k)$ è una distribuzione gaussiana di probabilità nello spazio $\mathbb{R} \times \mathbb{X}$, definita univocamente da una media μ_k e da una matrice di covarianza Σ_k, cioè

$$P_k(\xi|k) = \mathcal{N}(\xi, \mu_k, \Sigma_k) = \frac{e^{-\frac{1}{2}(\xi-\mu_k)^T \Sigma_k^{-1}(\xi-\mu_k)}}{\sqrt{(2\pi)^D \det \Sigma_k}} \tag{4.14}$$

dove D è la dimensione di ξ.

Notando che (4.13) è scritta nella forma di una distribuzione di probabilità a posteriori (in una delle tante forme del Teorema di Bayes), il termine $p(k)$ è anche chiamato *priori* mentre le $P_k(\xi|k)$ sono chiamate probabilità condizionali. Di fatto, $p(k)$ è associata ad una variabile casuale categorica latente (cioè nascosta) e descrive la probabilità che un campione qualunque sia generato da una delle varie gaussiane che costituiscono il GMM.

Chiamando $p(k) = \pi_k$ (con il vincolo che $0 \leq \pi_k \leq 1$ e che $\sum_k \pi_k = 1$), un modello GMM è definito dal meta-parametro K e dalle K tuple di parametri $\big\{(\pi_k, \mu_k, \Sigma_k)\big\}$ con $k \in [1, K]$.

Fissato il valore di K, esistono diversi approcci per stimare i parametri del modello GMM. Uno dei più diffusi è l'algoritmo di massimizzazione dell'aspettazione (Expectation Maximization - EM) che si basa su di una inizializzazione dei parametri del modello utilizzando un algoritmo di clustering (k-means clustering) e poi ottimizza iterativamente la massima verosimiglianza (Maximum Likelyhood del modello in maniera iterativa. Di seguito vedremo come eseguire questi due step. Infine riporteremo i dettagli di come ricostruire $\hat{x^*}(t)$ a partire da $\hat{P}(\xi)$, secondo un processo chiamato *Gaussian Mixture Regression* (GMR).

L'algoritmo Expectation Maximization

L'algoritmo EM si pone l'obiettivo di massimizzare la media del logaritmo della verosimiglianza a posteriori del modello $\mathcal{L}\left(\big\{(\pi_k, \mu_k, \Sigma_k)\big\}\right)$, date le osservazioni $\{\xi_j\}$, agendo sul valore dei parametri di un modello probabilistico (nel nostro caso un modello GMM). Il logaritmo della verosimiglianza a posteriori del modello $\mathcal{L}\left(\big\{(\pi_k, \mu_k, \Sigma_k)\big\}\right)$ è definito come

$$\mathcal{L}\left(\big\{(\pi_k, \mu_k, \Sigma_k)\big\}\right) = \frac{1}{N} \sum_{j=1}^{N} log(\hat{P}(\xi_j)) \ . \tag{4.15}$$

Purtroppo, la massimizzazione diretta di (4.15) non ammette una soluzione in forma chiusa, data la dipendenza complessa di $p(\xi_j)$ dai parametri del modello GMM. Ma la soluzione sarebbe banale se fosse nota la a probabilità $p(k|\xi_j)$ che ξ_j appartenga al cluster k data la sua osservazione, e sarebbe pari (per un modello GMM) a

$$\mu_k = \frac{\sum_{j=1}^{N} p(k|\xi_j)\xi_j}{\sum_{j=1}^{N} p(k|\xi_j)} \tag{4.16}$$

$$\Sigma_k = \frac{\sum_{j=1}^{N} p(k|\xi_j)(\xi_j - \mu_k)(\xi_j - \mu_k)^T}{\sum_{j=1}^{N} p(k|\xi_j)} \tag{4.17}$$

$$\pi_k = \frac{\sum_{j=1}^{N} p(k|\xi_j)}{N}\,, \tag{4.18}$$

Dualmente a ciò, se fossero noti tutti i parametri del GMM, sarebbe possibile ricostruire $p(k|\xi_j)$ con l'uso della formula di Bayes, come

$$p(k|\xi_j) = \frac{p(k)P_k(\xi_j|k)}{\sum_{i=1}^{K} p(i)P_i(\xi_j|i)} = \frac{\pi_k \mathcal{N}(\xi_j, \mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\xi_j, \mu_i, \Sigma_i)}\,. \tag{4.19}$$

L'algoritmo EM si basa su queste osservazioni e propone di applicare i due set di formule (4.18) e (4.19) in maniera iterativa per giungere ad una soluzione. Una proprietà molto importante dell'algoritmo EM consiste nel fatto che si può dimostrare la sua convergenza ad un equilibrio quando viene utilizzato su un modello di tipo GMM.

Formalizzando l'algoritmo, abbiamo perciò che partendo da una stima qualunque[1] dei parametri del modello $\left\{\left(\pi_k^0, \mu_k^0, \Sigma_k^0\right)\right\}$, si applicano i seguenti due passi in maniera iterativa, che sono:

E-step: (a) si aggiorna la stima di $p_{k,j}^{(t+1)} = \frac{\pi_k^t \mathcal{N}(\xi_j^t, mu_k^t, \Sigma_k^t)}{\sum_{i=1}^{K} \pi_i^t \mathcal{N}(\xi_j^t, mu_i^t, \Sigma_i^t)}$

(b) si calcola la quantità $E_k^{(t+1)} = \sum_{j=1}^{N} p_{k,j}^{(t+1)}$

M-step: si aggiornano le stime del modello come:

(a) $\pi_k^{(t+1)} = \frac{E_k^{(t+1)}}{N}$

(b) $\mu_k^{(t+1)} = \frac{\sum_{j=1}^{N} p_{k,j}^{(t+1)} \xi_j}{E_k^{(t+1)}}$

[1] Una stima di partenza migliore, che porta ad una convergenza più veloce si può effettuare con l'algoritmo k-means clustering, presentato nel seguito.

(c) $\Sigma_k^{(t+1)} = \frac{\sum_{j=1}^{N} p_{k,j}^{(t+1)} (\xi_j - \mu_k^{(t+1)})(\xi_j - \mu_k^{(t+1)})^T}{E_k^{(t+1)}}$

Le iterazioni si fermano quando l'evoluzione di $\mathcal{L}$ rallenta, in pratica misurando quando $\frac{\mathcal{L}^{(t+1)}}{\mathcal{L}^{(t)}} < C$ dove C è una costante arbitraria (spesso si sceglie $C = 1\%$).

La quantità introdotta

$$E_k = \sum_{j=1}^{N} p(k/\xi_j) \text{ dove } p(k/\xi_j) = \frac{p(k) P_k(\xi_j|k)}{\sum_{i=1}^{K} p(i) P_i(\xi_j|i)} , \tag{4.20}$$

è una cumulata a posteriori delle probabilità che un datapoint ξ_j appartenga alla k-esima gaussiana, data la sua osservazione. La sua introduzione serve non soltanto a semplificare la notazione, ma principalmente ad ottimizzare i calcoli, dato che compare diverse volte nelle formule soprastanti. Data la sua ubiquità, questa viene spesso inserita come quarto elemento delle tuple che costituiscono i parametre del GMM, che in molti testi vengono presentati come $\big\{(\pi_k, \mu_k, \Sigma_k, E_k)\big\}$, anche se di fatto per definire il modello GMM bastano i primi 3.

Gaussian Mixture Regression

Una volta costruito il nostro modello GMM $\hat{P}(\xi)$ a partire dalle osservazioni $\{\xi_j\}$, che sarà definito dal meta-parametro K e dalle K tuple di parametri $\big\{(\pi_k, \mu_k, \Sigma_k)\big\}$ (includendo opzionalmente anche E_k), che ricordiamo definire una distribuzione di probabilità nello spazio delle $\boldsymbol{\Xi} = \mathbb{X} \times \mathbb{R}$, vogliamo utilizzarlo per ricostruire il nostro segnale ideale $x^*(t)$, cioè una funzione $\mathbb{R} \to \mathbb{X}$.

Per fare ciò continuiamo ad utilizzare gli strumenti della teoria della probabilità e dei modelli statistici. In particolare andremo a cercare la probabilità condizionata $\hat{P}(x|t)$, partendo dalla conoscenza di $\hat{P}(\xi)$ e ricordando che $\xi = [t, x^T]^T$, e ricostruiremo la traiettoria come

$$\hat{x}^*(t) = E[\hat{P}(x|t)] . \tag{4.21}$$

Per fare ciò utilizziamo la tecnica delle *Gaussian Mixture Regression* (GMR), che si basa sulla proprietà delle combinazioni lineari di Gaussiane.

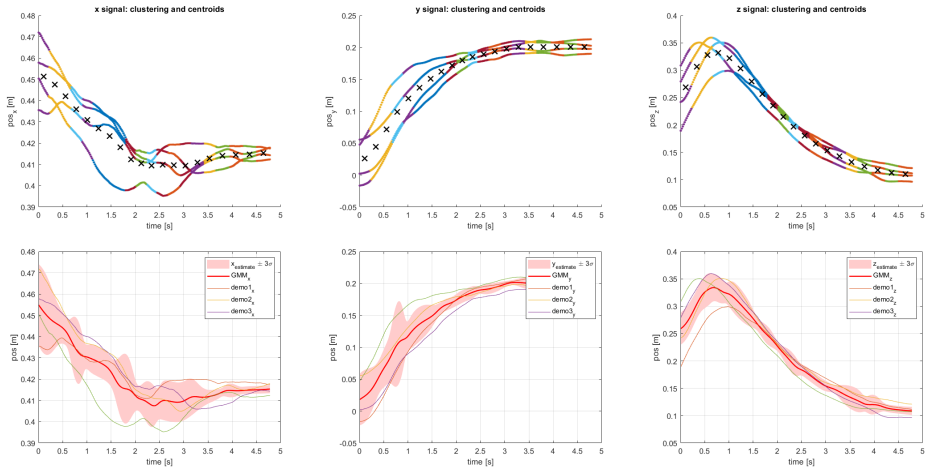

Figura 4.2: Traiettorie originali (x, y e z in funzione di t), clusterizzate con k-means (sopra) e con la ricostruzione tramite GMM e GMR (sotto).

Notiamo innanzitutto che è possibile separare le componenti temporali e spaziali che definiscono le varie gaussiane semplicemente prendendo

$$\mu_k = \begin{bmatrix} \mu_{t,k} \\ \mu_{x,k} \end{bmatrix} \tag{4.22}$$

$$\Sigma_k = \begin{bmatrix} \Sigma_{tt,k} & \Sigma_{tx,k} \\ \Sigma_{xt,k} & \Sigma_{xx,k} \end{bmatrix} \tag{4.23}$$

Per ognuna delle K distribuzioni gaussiane, la distribuzione condizionata dato il tempo t è

$$P_k(x|t) = \mathcal{N}(x, \widehat{x}_k(t), \widehat{\Sigma}_{xx,k}(t)) \,, \tag{4.24}$$

dove

$$\widehat{x}_k(t) = \mu_{x,k} + \Sigma x_{t,k}(\Sigma_{tt,k})^{-1}(t - \mu_{t,k}) \tag{4.25}$$

$$\tag{4.26}$$

$$\widehat{\Sigma}_{xx,k}(t) = \Sigma_{xx,k} - \Sigma_{xt,k}(\Sigma_{tt,k})^{-1}\Sigma_{tx,k} \,. \tag{4.27}$$

Le nostre gaussiane si combinano linearmente nel tempo secondo dei pesi β_k che dipendono dalla distribuzione di probabilità categorica $p(k)$, in particolare in base alla probabilità che la gaussiana k-esima abbia contribuito alla generazione di t. Perciò, ad un dato istante di tempo t avremo

$$\beta_k = \frac{p(k)P_k(t|k)}{\sum_{i=1}^{K} p(i)P_k(t|i)} \,, \tag{4.28}$$

dove

$$P_k(t|k) = \mathcal{N}(t, \mu_{t,k}, \Sigma_{tt,k}) \tag{4.29}$$

Ora abbiamo tutti gli ingredienti per scrivere finalmente

$$\hat{x}^*(t) = E[\hat{P}(x|t)] = \sum_{k=1}^{K} \beta_k \widehat{x}_k(t) = \sum_{k=1}^{K} \beta_k \left(\mu_{x,k} + \Sigma_{xt,k}(\Sigma_{tt,k})^{-1}(t - \mu_{t,k})\right) . \tag{4.30}$$

Ricordiamo, infine, che la ricostruzione di $\hat{x}^*(t)$ non è l'unica informazione che possiamo estrarre da $\hat{P}(\xi)$. Infatti, in base alla osservazione precedente, ricordiamo che possiamo ad esempio estrarre informazioni riguardanti eventuali vincoli spazio-temporali del movimento studiando la forma di $\hat{P}(\xi)$.

Ad esempio studiando $E[\text{cov}(\hat{P}(x|t))]$ e vedendo quando questa si riduce e lungo quali direzioni, possiamo dedurre eventuali vincoli spaziali del movimento in funzione del tempo. A tal fine riportiamo anche la formula per la sua ricostruzione che è semplicemente

$$E[\text{cov}(\hat{P}(x|t))] = \widehat{\Sigma}_{xx}(t) = \sum_{k=1}^{K} \beta_k^2 \widehat{\Sigma}_{x,k}(t) . \tag{4.31}$$

Una trattazione più completa dell'argomento è presente nel libro [**?**].

K-Means Clustering

L'algoritmo di *K-Means Clustering* è un algoritmo che dati n punti ξ_i appartenenti ad uno spazio vettoriale $\boldsymbol{\Xi}$, ed un numero K, suddivide i punti ξ_i in K insiemi S_k, in modo da minimizzare la somma della distanza quadratica totale di tutti i punti dal centroide c_k del cluster cui appartengono, cioè

$$S_k = \arg\min_{S} \sum_{i=1}^{K} \sum \xi \in S_k ||\xi - c_k||^2 . \tag{4.32}$$

Il centroide c_k del cluster S_k è definito come la media dei suoi punti, cioè

$$c_k = \frac{1}{\#S_i} \sum_{\xi \in S_k} \xi , \tag{4.33}$$

dove $\#S_k$ indica il numero di elementi nel cluster S_k.

Inizializzazione dell'algoritmo L'algoritmo del K-Means Clustering procede iterativamente a partire da una partizione iniziale stimata, di solito generata casualmente. Un modo per generare tale stima iniziale è quello di scegliere K punti $\hat{\xi}_k$ a caso nell'insieme $\{\xi\}$, di generare la partizione iniziale assegnando al set S_k i punti più vicini a $\hat{\xi}_k$ e calcolando le stime iniziali dei centroidi c_k come in (4.33).

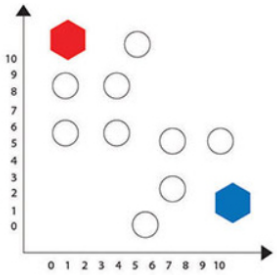

Figura 4.3: Inizializzazione casuale dell'algoritmo k-means.

L'algoritmo k-means Una volta inizializzati i cluster, l'algoritmo k-means procede iterativamente secondo i seguenti passi:

1. Si calcolino tutte le distanze di tutti gli ξ da tutti i c_k,
2. Si assegni ogni osservazione al cluster con il centroide più vicino,
3. Si ricalcolino i centroidi con (4.33),
4. Si ripetano i passi da 2 a 4 fino a che al passo 2 nessuno dei punti cambia più cluster (e quindi l'algoritmo è arrivato a convergenza).

A questo punto, possiamo inizializzare i parametri dell'algoritmo EM per la ricerca dei parametri del GMM, associando una gaussiana ad ogni cluster S_k, definita da

- media $\mu_k^0 = c_k$ centroide del cluster k,
- covarianza $\Sigma_k^0 = E[(\xi_j - c_k)(\xi_j - c_k)^T]$, pari cioè alla covarianza dei punti in S_k,
- e probabilità a priori del cluster $\pi_k^0 = \#S_k/N$, proporzionale alla numerosità di punti nel cluster.

k-means++ Un modo leggermente più efficiente di inizializzare l'algoritmo k-means cluster, chiamato *k-means++* è espresso dal seguente algoritmo, che sceglie i punti $\hat{\xi}_k$ cercando di prenderli distanti tra loro:

a. Si scelga un primo punto $\hat{\xi}_1$ a caso nell'insieme $\{\xi\}$ con probabilità uniforme, e lo si inserisca nell'insieme degli $\hat{\xi}$
b. Si calcoli la distanza di ogni punto ξ dall'insieme $\{\hat{\xi}\}$ definita come

$$d(\xi_j, \hat{\xi}) = \min_k d(\xi_j, \xi_k) \tag{4.34}$$

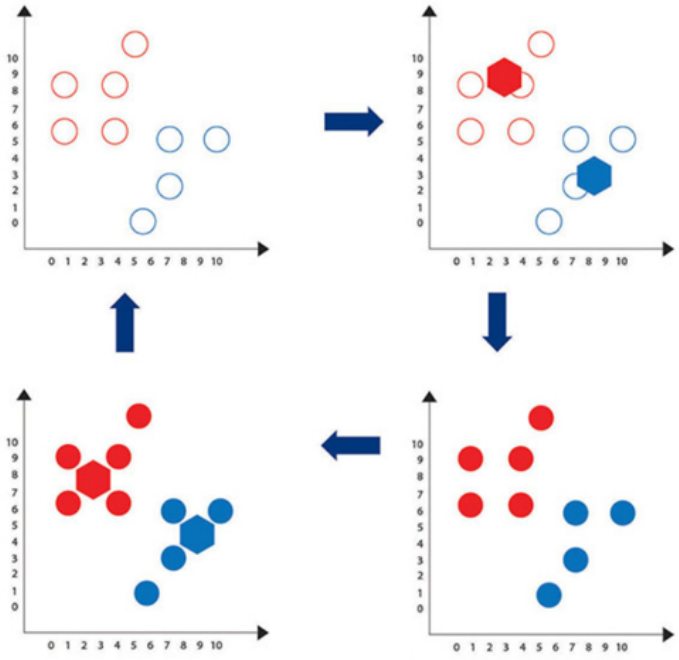

Figura 4.4: Processo iterativo dell'algoritmo k-means.

cioè quella dal punto più vicino (notare che il range di k dipende da quanti punti sono già stati scelti).

c. Si selezioni un nuovo punto $\hat{\xi}_{k+1}$ a caso dall'insieme $\{\xi\}$ a caso estraendolo con probabilità proporzionale al quadrato delle distanze calcolate, cioè

$$p(\xi = \xi_{k+1}) = \frac{d^2(\xi, \{\hat{\xi}\})}{\sum_{j=1}^{n} d^2(\xi_j, \hat{\xi})} \,. \tag{4.35}$$

d. Si ripetano i punti [b.] e [c.] fino alla scelta di K punti iniziali $\{\hat{\xi}\}$.

L'inizializzazione termina assegnando ogni punto in $\{\xi\}$ al cluster S_k corrispondente al $\hat{\xi}_k$ più vicino, e calcolando le stime iniziali dei centroidi c_k come in (4.33).

Bibliografia

[1] C. Melchiorri, *Traiettorie per azionamenti elettrici*. Società Editrice Esculapio, 2020.

[2] L. Biagiotti and C. Melchiorri, *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008. ,

[3] S. B. Slotine and B. Siciliano, "A general framework for managing multiple tasks in highly redundant robotic systems," in *proceeding of 5th International Conference on Advanced Robotics*, vol. 2, pp. 1211–1216, 1991.

[4] A. De Luca, G. Oriolo, B. Siciliano, *et al.*, "Robot redundancy resolution at acceleration level," *Laboratory Robotics and Automation*, vol. 4, pp. 97–106, 1992.

[5] F. Flacco and A. De Luca, "A reverse priority approach to multitask control of redundant robots," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2421–2427, IEEE, 2014.

[6] F. Flacco and A. De Luca, "Unilateral constraints in the reverse priority redundancy resolution method," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2564–2571, IEEE, 2015.

[7] R. Shadmehr and S. Wise, "Supplementary documents for "computational neurobiology of reaching and pointing"," 2005.

[8] C. G. L. Bianco and F. Ghilardelli, "A discrete-time filter for the generation of signals with asymmetric and variable bounds on velocity, acceleration, and jerk," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4115–4125, 2013.

[9] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[10] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pp. 255–262, 2007. ,

[11] C. Sylvain, *Robot programming by demonstration: A probabilistic approach*. EPFL Press, 2009.