UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI

SEDE DI VICENZA

CORSO DI LAUREA IN INGEGNERIA MECCANICA

Tesi di Laurea in

Ingegneria Meccanica

(Laurea triennale L9 - Ingegneria industriale)

SVILUPPO DI UN SOFTWARE PER LA GENERAZIONE DI TRAIETTORIE SPLINE IN AMBIENTE MATLAB

Relatore: Prof. Richiedei Dario

Laureando: Michele Littea

ANNO ACCADEMICO 2015/2016

INTRODUZIONE **1**

SPLINE: INTRODUZIONE ..... ERRORE. IL SEGNALIBRO NON È DEFINITO.

IMPLEMENTAZIONE SPLINE **5**

2.1 Creazione function propedeutiche **5**
2.1.1 Coefficienti della spline 5
2.1.2 Polinomiale di 3° grado 8
2.1.3 Polinomiale di 4° grado 8
2.1.4 Spline 4-3-4 9
2.1.5 Spline con tratti a velocità costante o nulla 10

2.2 Main **11**
2.2.1 Inserimento dati 11
2.2.2 Definizione automatica intervalli temporali 13
2.2.3 Concatenazione funzioni 14
2.2.4 Analisi e warning 17
2.2.5 Grafici 19

2.3 Esempio **21**

2.4 Confronto Spline 4-3-4 e Spline Cubica **28**
2.4.1 Posizione 28
2.4.2 Velocità 29
2.4.3 Accelerazione 29

OTTIMIZZAZIONE TEMPO TOTALE SPLINE **31**

3.1 Spline Cubica **33**

3.2 Spline 4-3-4 ..... **35**

3.3 Confronto **38**

B-SPLINE **43**

4.1 Inserimento dati, vettore dei nodi e dei termini noti **44**

4.2 Funzioni base **45**
4.2.1 Posizione 45
4.2.2 Velocità 46
4.2.3 Accelerazione 47

4.3 Costruzione legge di moto 48

4.4 Confronto con spline 4-3-4 50

CONCLUSIONI 55

BIBLIOGRAFIA 57

Introduzione

La trattazione contenuta in questo documento di tesi di laurea triennale in Ingegneria Meccanica ha lo scopo di implementare tramite l'utilizzo del software MATLAB dei metodi che permettano la rappresentazione e lo studio di traiettorie per il movimento. In particolare si intende: l'interpolazione di un dato numero di punti, rappresentanti il passaggio della traiettoria negli intervalli di tempo definiti, con profili che dovranno essere quanto più morbidi e privi di discontinuità. Si dovranno altresì evitare brusche variazioni di velocità ed accelerazione per lavorare al meglio e per evitare danni alle apparecchiature o vibrazioni. Il tutto dovrà essere eseguito dal software MATLAB in modo quanto più rapido e soprattutto con dati generici, e per questo si studieranno diverse tipologie per l'implementazione e le casistiche presenti in ognuno, confrontando i risultati.

Capitolo 1

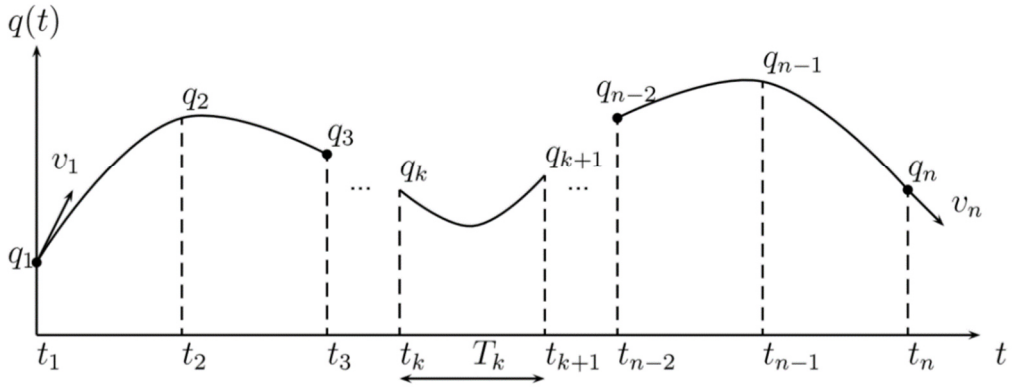
SPLINE: INTRODUZIONE

Per programmare un movimento i dati fondamentali da definire sono: il cammino geometrico, ossia le varie posizioni assunte dal sistema in un determinato momento temporale; per descrivere questi cammini solitamente si utilizza la legge oraria (o legge di moto), basata sulle funzioni polinomiali di un determinato ordine, in funzione del tempo. Essenzialmente essa è una legge matematica che descrive il passaggio da un punto desiderato all'altro tramite delle linee dolci; il termine inglese *smooth* rende meglio l'idea di come devono essere questi tratti. Una funzione polinomiale di ordine n è descritta dalla seguente formula in funzione del tempo t:

$$s(t)=a_0+a_1*t+a_2*t^2+\ldots+a_n*t^n$$

Ricavabile analiticamente date delle condizioni iniziali. Nella più classica soluzione per interpolare n punti si potrebbe pensare di utilizzare una legge di moto di ordine n-1, infatti dati n punti esiste un polinomio di grado n-1 che passa per tali punti. Questa soluzione può essere accettabile se si interpolano pochi punti e il grado del polinomio resta contenuto; se come nella maggior parte delle applicazioni n è molto elevato, all'aumentare del grado del polinomio aumenta il suo carattere oscillatorio rendendo così incontrollate le velocità e le accelerazioni che vanno ad assumere valori incompatibili con quelli previsti inizialmente. Si hanno quindi linee a curvatura e oscillazione ridotte quanto più è basso l'ordine del polinomio interpolante. Una soluzione plausibile è quella di interpolare gli n punti con n-1 polinomi di grado inferiore, nel nostro caso si useranno n-1 polinomi di terzo grado. Per risolvere tale problema servirebbero quindi 4 condizioni per ogni polinomio di grado 3 considerato. Si impongono quindi due vincoli di passaggio e due di velocità agli estremi; facendo ciò però andrebbe persa la continuità dell'accelerazione (la funzione sarebbe continua con derivata prima continua ma derivata seconda non continua), ingestibile da una polinomiale di terzo grado. Per ovviare a questo problema che renderebbe incompatibile la nostra traiettoria con la corrispettiva nella realtà si utilizza la *spline*, acronimo di *Smooth Path Line* (linea con un percorso regolare/liscio). La spline è una funzione che interpola un numero n di punti ed è formata da n-1 polinomi di terzo grado raccordati da condizioni di

continuità di posizione, velocità e accelerazione. Per poter imporre la continuità nell'accelerazione si vanno a sacrificare, rispetto alla soluzione precedente, l'imposizione delle velocità, che liberà così i gradi di libertà necessari. Da un'analisi di questa soluzione si vanno a definire quali e quante condizioni a contorno e intermedie si necessita per poter trovare i coefficienti della spline, ma di questo si parlerà nella parte successiva. Infine si precisa che volendo anche imporre le accelerazioni iniziali e finali della spline (unici punti in cui non abbiamo continuità), oltre alle velocità nei medesimi punti, si utilizzerà una spline 4-3-4 che presenta tutti polinomi di terzo grado tranne che nel primo e nell'ultimo tratto appunto dove si hanno polinomi di quarto grado. La trattazione legata al software MATLAB verrà fatta pari passo alla parte analitica, con descrizione dell'implementazione e dei fondamenti teorici.



Capitolo 2

IMPLEMENTAZIONE SPLINE

2.1 Creazione function propedeutiche

2.1.1 Coefficienti della spline

Per prima cosa si andrà ad analizzare la soluzione descritta sopra a riguardo dell'interpolazione, ossia quella della spline 4-3-4. Tale spline interpola n punti (P_n) ed è formata da n-1 polinomi, 2 di grado quattro e n-3 di grado tre. Ricordiamo le equazioni di questi polinomi:

$$q_j(t)=a_{0j}+a_{1j}(t-t_j)+a_{2j}(t-t_j)^2+a_{3j}(t-t_j)^3 \qquad q_j(t)=a_{0j}+a_{1j}(t-t_j)+a_{2j}(t-t_j)^2+a_{3j}(t-t_j)^3+a_{4j}(t-t_j)^4$$

Si avranno quindi 4*(n-1) condizioni più due condizioni extra date dall'uso delle polinomiali di quarto grado. Vediamo quali sono queste condizioni (la velocità sarà espressa con P_n'; l'accelerazione sarà espressa con P_n''):

- 6 condizioni saranno a contorno, ossia ad inizio e fine della spline:
  $q_1(t_1)=P_1$ e $q_{n-1}(t_n)=P_n$ -> condizioni iniziali e finali di posizione;
  $q_1'(t_1)=P_1'$ e $q_{n-1}'(t_n)=P_n'$ -> condizioni iniziali e finali di velocità;
  $q_1''(t_1)=P_1''$ e $q_{n-1}''(t_n)=P_n''$ -> condizioni iniziali e finali di accelerazione;
- 4*(n-2) condizioni intermedie legate alla continuità:
  $q_j(t_{j+1})=P_{j+1}$ e $q_{j+1}(t_{j+1})=P_{j+1}$ -> 2*(n-2) condizioni di continuità di posizione;
  $q_j'(t_{j+1})=q_{j+1}'(t_{j+1})$ -> (n-2) condizioni di continuità di velocità;
  $q_j''(t_{j+1})=q_{j+1}''(t_{j+1})$ -> (n-2) condizioni di continuità di accelerazione;

Da queste condizioni si ottiene un sistema lineare che ha: come incognite il vettore dei coefficienti a_{ij} delle polinomiali che saranno 4*n-2; come vettore dei termini noti i punti di passaggio e le velocità e accelerazioni iniziali e finali, oltre a tutte le condizioni di continuità; come matrice che moltiplica le incognite si hanno i termini dipendenti dagli istanti temporali. Il sistema sarà quindi risolto invertendo la matrice dei termini dipendenti dal tempo e moltiplicandola per i termini noti. Per costruire la matrice si andranno a sviluppare le condizioni sopra esposte nei tratti iniziale e finale e nel generico tratto k-

esimo centrale. Dati: P_n i punti di passaggio e V_1, V_n e A_1, A_n rispettivamente velocità e accelerazione iniziali e finali.

$q_1(t_1)=a_{10}=P_1$

$q_1'(t_1)=a_{11}=V_1$

$q_1''(t_1)=2*a_{12}=A_1$

$q_1(t_2)=a_{10}+a_{11}(t_2-t_1)+a_{12}(t_2-t_1)^2+a_{13}(t_2-t_1)^3+a_{14}(t_2-t_1)^4-a_{20}=0$

$q_1'(t_2)=a_{11}+2*a_{12}(t_2-t_1)+3*a_{13}(t_2-t_1)^2+4*a_{14}(t_2-t_1)^3-a_{21}=0$

$q_1''(t_2)=2*a_{12}+6*a_{13}(t_2-t_1)+12*a_{14}(t_2-t_1)^2-2*a_{22}=0$

$q_k(t_k)=a_{k0}=P_k$

$q_k(t_{k+1})=a_{k0}+a_{k1}(t_{k+1}-t_k)+a_{k2}(t_{k+1}-t_k)^2+a_{k3}(t_{k+1}-t_k)^3+a_{k4}(t_{k+1}-t_k)^4-a_{k+1,0}=0$

$q_k'(t_{k+1})=a_{k1}+2*a_{k2}(t_{k+1}-t_k)+3*a_{k3}(t_{k+1}-t_k)^2+4*a_{k4}(t_{k+1}-t_k)^3-a_{k+1,1}=0$

$q_k''(t_{k+1})=2*a_{k2}+6*a_{k3}(t_{k+1}-t_k)+12*a_{k4}(t_{k+1}-t_k)^2-2*a_{k+1,2}=0$

$q_{n-1}(t_{n-1})=a_{n-1,0}=P_{n-1}$

$q_{n-1}(t_n)=a_{n-1,0}+a_{n-1,1}(t_n-t_{n-1})+a_{n-1,2}(t_n-t_{n-1})^2+a_{n-1,3}(t_n-t_{n-1})^3+a_{n-1,4}(t_n-t_{n-1})^4=P_n$

$q_{n-1}'(t_n)=a_{n-1,1}+2*a_{n-1,2}(t_n-t_{n-1})+3*a_{n-1,3}(t_n-t_{n-1})^2+4*a_{n-1,4}(t_n-t_{n-1})^3=V_n$

$q_{n-1}''(t_n)=2*a_{n-1,2}+6*a_{n-1,3}(t_n-t_{n-1})+12*a_{n-14}(t_n-t_{n-1})^2=A_n$

Dalle relazioni scritte sopra se ne ricava il sistema lineare descritto. Implementando in MATLAB quanto appena spiegato si ottiene la seguente funzione che dati in ingresso i punti, la lunghezza di ogni intervallo temporale e le velocità e accelerazioni iniziali e finali dà in uscita il vettore contenente tutti i coefficienti $a_{ij}=[a_{10},a_{11},a_{12},a_{13},a_{14},\ldots a_{k0}, a_{k1},a_{k2},a_{k3},\ldots, a_{n-1,0},a_{n-1,1},a_{n-1,2},a_{n-1,3},a_{n-1,4}]$

```
%% Coefficenti
function[CF]=coeff(P,T,vi,vf,ai,af)
n=length(P);%numero punti
D=zeros(4*n-2,4*n-2);%istanzio a una matrice di tutti zeri la matrice
dei
%termini dipendenti da T
TN=zeros(4*n-2,1);%istanzio a zero la matrice dei termini noti
%costruzione matrice termini noti
TN(1)=P(1);
TN(2)=vi;
TN(3)=ai;
TN(end)=af;
TN(end-1)=vf;
```

```
TN(end-2)=P(end);
TN(end-3)=P(end-1);
TNr=zeros(4*n-12,1);
m=1;
for k=2:n-2
    TNr(m)=P(k);
    m=m+4;
end
TN(7:4*n-6,1)=TNr;
%costruzione matrice termini dipendenti da T
D1=diag([1,1,2]);
D(1:3,1:3)=D1;
D2=[1,T(1),T(1)^2,T(1)^3,T(1)^4
    0,1,2*T(1),3*T(1)^2,4*T(1)^3
    0,0,2,6*T(1),12*T(1)^2];
D3=-D1;
D(4:6,6:8)=D3;
D(4:6,1:5)=D2;
D4=[1,0,0,0,0
    1,T(end),T(end)^2,T(end)^3,T(end)^4
    0,1,2*T(end),3*T(end)^2,4*T(end)^3
    0,0,2,6*T(end),12*T(end)^2];
D(4*n-5:4*n-2,4*n-6:4*n-2)=D4;
for k=2:n-2
    D5(:,:,k)=[1,0,0,0,0,0,0
        1,T(k),T(k)^2,T(k)^3,-1,0,0
        0,1,2*T(k),3*T(k)^2,0,-1,0
        0,0,2,6*T(k),0,0,-2];
end

D6=zeros(4*n-12,4*n-7);
    r=1;
    c=1;
for k=2:n-2
    D6(r:r+3,c:c+6)=D5(1:4,1:7,k);
    r=r+4;
    c=c+4;
end
D(7:4*n-6,6:4*n-2)=D6;
CF=pinv(D)*TN;
return
```

I coefficienti trovati saranno usati rispettivamente: i primi cinque e gli ultimi cinque per la polinomiale di quarto grado; tutti gli altri a quattro a quattro per le polinomiali di terzo grado della spline.

Adesso che si sono trovati i coefficienti si andranno a costruire le funzioni che daranno in uscita la posizione, la velocità e l'accelerazione secondo la legge polinomiale rispettivamente di quarto o terzo grado. Come ingresso entrambe richiedono gli intervalli temporali e i rispettivi coefficienti calcolati al punto precedente.

2.1.2 Polinomiale di 3° grado

```
%% Poly3

function[pos,vel,acc]=poly3(t,a)
t0=t(1);
tr=t-t0;
for k=1:length(t)
    pos(:,1)=a(1)+a(2)*tr+a(3)*tr.^2+a(4)*tr.^3;
    vel(:,1)=a(2)+2*a(3)*tr+3*a(4)*tr.^2;
    acc(:,1)=2*a(3)+6*a(4)*tr;
end
return
```

2.1.3 Polinomiale di 4° grado

```
%% Poly4

function[pos,vel,acc]=poly4(t,a)
t0=t(1);
tr=t-t0;
for k=1:length(t)
    pos(:,1)=a(1)+a(2)*tr+a(3)*tr.^2+a(4)*tr.^3+a(5)*tr.^4;
    vel(:,1)=a(2)+2*a(3)*tr+3*a(4)*tr.^2+4*a(5)*tr.^3;
    acc(:,1)=2*a(3)+6*a(4)*tr+12*a(5)*tr.^2;
end
return
```

2.1.4 Spline 4-3-4

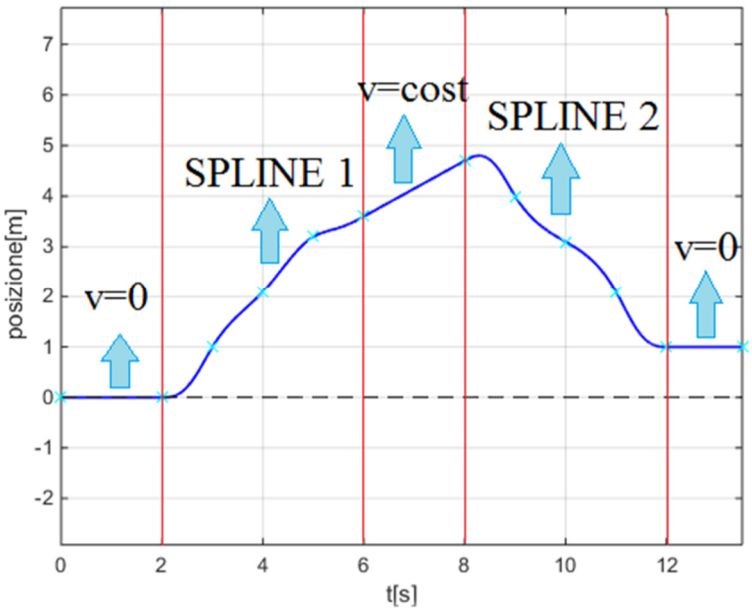
Adesso che si son create delle funzioni per le polinomiali si possono utilizzare per implementare la legge di moto descritta sopra, ossia la spline 4-3-4, in cui si utilizzeranno per il primo e l'ultimo intervallo la Poly4 e per gli intervalli intermedi la Poly3. Di seguito il codice MATLAB della funzione spline 4-3-4; si osservi che devono essere richiamate tutte e tre le funzioni create sopra.

```
%% SPLINE 4-3-4
function[pos,vel,acc]=spline434(P,T,vi,vf,ai,af)
n=length(P);
dt=0.01;
t(1)=0;
for k=2:n
    t(k)=t(k-1)+T(k-1);
end
%% Calcolo coefficenti
A=coeff(P,T,vi,vf,ai,af);

A4i=A(1:5,1);
A4f=A(4*n-6:4*n-2,1);
A3=A(6:4*n-7,1);
m=1;
for k=2:n-2
A3s(:,:,k)=[A3(m);A3(m+1);A3(m+2);A3(m+3)];
m=m+4;
end
%% Calcolo della legge di moto
Pos2=cell(1,n-3);
Vel2=cell(1,n-3);
Acc2=cell(1,n-3);
for k=2:n-2
[Pos2{k},Vel2{k},Acc2{k}]=poly3(t(k):dt:t(k+1)-dt,A3s(:,:,k));
end
pos2=cat(1,Pos2{:});
vel2=cat(1,Vel2{:});
acc2=cat(1,Acc2{:});

[pos1,vel1,acc1]=poly4(t(1):dt:t(2)-dt,A4i);
[pos3,vel3,acc3]=poly4(t(n-1):dt:t(n)-dt,A4f);
```

```
pos=[pos1;pos2;pos3];
vel=[vel1;vel2;vel3];
acc=[acc1;acc2;acc3];

return
```

2.1.5 Spline con tratti a velocità costante o nulla

A questo punto si hanno tutti i punti che compongo la spline, e quindi si ha la legge di moto completa che interpola un dato numero di punti. Per le applicazioni reali però, il movimento può non essere descritto solo da un'unica spline, ma si possono altresì trovare degli intervalli in cui ad esempio un macchinario è in fase di fermata, quindi a velocità nulla, oppure procede con una velocità costante. Queste tipologie di moto non sono contemplate in una legge polinomiale qual è la spline, e si dovrà quindi ricorrere a funzioni che danno in uscita una legge oraria descrittiva dei due moti appena citati. Fatto ciò è intuitivo pensare che concatenando molteplici tratti, ognuno descritto da una legge che può essere la spline 4-3-4 o quella a velocità nulla o costante si può descrivere in modo completo la traiettoria desiderata. Per prima cosa creiamo la funzione che ci darà in uscita la legge oraria relativa alle casistiche di velocità costante o nulla. L'implementazione di tale funzione proposta qui di seguito è semplice e contempla entrambi i casi; se i valori di posizione iniziale e finale dati in ingresso sono diversi verrà creato un tratto a velocità costante; se i valori di posizione iniziale e finale dati in ingresso sono uguali verrà creato un tratto a velocità nulla.



```
%% Velocità costante o nulla
function[pos,vel,acc]=vcost(qi,qf,vi,Tt)
t0=Tt(1);
tr=Tt-t0;
if qi==qf
    pos=qi.*ones(length(Tt),1);
    vel=zeros(length(Tt),1);
    acc=zeros(length(Tt),1);
elseif ne(qi,qf)
    for k=1:length(Tt)
          pos(:,1)=qi+vi*tr;
    end
    vel=vi.*ones(length(Tt),1);
    acc=zeros(length(Tt),1);
else
    warning('cost è 0(velocità nulla) o 1(velocità costante');
end
return
```

2.2 Main

Ora che si hanno tutte le funzioni base per creare la legge di moto completa del movimento si può creare in MATLAB un ".m" file nel quale si andranno ad inserire i dati da analizzare, ossia: il numero di tratti, e quindi di leggi da usare, i punti da interpolare, la lunghezza degli intervalli e, dove necessario, le velocità e le accelerazioni imposte. Fatto ciò si andranno a concatenare tutti i risultati ottenuti per ogni tratto in un'unica legge descrittiva di tutto il movimento. Infine si passerà ad analizzare degli specifici punti di interesse e a graficare i risultati ottenuti.

2.2.1 Inserimento dati

Per prima cosa nel main si vanno a creare *ad hoc* i dati che serviranno per creare la legge di moto. Si avrà quindi un input che richiederà in ingresso il numero di tratti di cui si compone il movimento; ossia il numero di spline o tratti a velocità costante o nulla che lo compongono. Successivamente si andrà a definire il numero n di punti da interpolare e gli stessi saranno quindi inseriti manualmente. Infine si andrà a scegliere la lunghezza di ciascuno degli n-1 intervalli temporali fra ogni punto.

```
%% Dati ricavati da input a video
dt=0.01;
trt=input('Inserire il numero di tratti  ');
n=input('Quanti punti da interpolare?  ');
disp('Si avrà un numero di punti pari a:');disp(n);
disp('e un numero di intervalli pari a:');disp(n-1);
for k=1:n
    P(k)=input('Inserire i punti da interpolare  ');
end
P=round(P,2);
m(1)=1;
for k=1:trt
    type(k)=input('Inserire il tipo di tratto(434->SPLINE434,1->VEL
COSTANTE,0->VEL NULLA)  ');
    if type(k)==434
        ps=input('Quanti punti interpola la spline434?  ');
        m(k+1)=m(k)+ps-1;
        manual_time=input('Si desidera inserire manualmente la lunghezza
di ogni intervallo temporale  ');
        if manual_time==1
            for j=m(k):m(k+1)-1
            T(j)=input('Inserire la lunghezza di ogni intervallo
temporale  ');
            end
        elseif manual_time==0
            Tt(k)=input('Quanto è la durata del moto? ');
            disp('Scegliere il metodo di spaziatura fra i punti');
            ind=input('1->equidistate 2->proporzionale alla posizione 3-
>proporzionale alla posizione alla 1/2 ');
            T(m(k):m(k+1)-1)=t_ist(P(m(k):m(k+1)),Tt(k),ind);
        end
    elseif type(k)==1 || type(k)==0
        m(k+1)=m(k)+1;
        T(m(k))=input('Inserire lunghezza temporale tratto a velocità
costante o nulla ');
    end
end

T=round(T,2);
t=(0:dt:sum(T))';
```

```
tt(1)=0;
for k=2:n
    tt(k)=tt(k-1)+T(k-1);
end
tt=tt';
tc=(ceil(100.*tt))';
```

2.2.2 Definizione automatica intervalli temporali

Spesso però i dati relativi agli intervalli temporali non sono ben definiti, oppure sono totalmente lasciati casuali in modo tale da poterli poi modificare per ottenere un'ottimizzazione delle prestazioni. Quindi l'inserimento manuale può risultare inefficace; si ricorre quindi a dei metodi di spaziatura fra i punti. In questa trattazione ne vengono presentati tre tra cui scegliere: il primo che considera intervalli tutti uguali su una data lunghezza totale; il secondo che proporziona gli intervalli in maniera diretta con il salto di posizione fra i due punti estremi all'intervallo; il terzo che opera come il secondo ma la proporzionalità è scalata di una potenza μ-esima solitamente uguale a 1/2. Il primo metodo solitamente presenta picchi di velocità maggiori, mentre l'ultimo mantiene valori ridotti di accelerazione. In ogni caso viene data la possibilità nel main di scegliere fra l'inserimento manuale o la scelta di uno dei metodi sopra citati. Nel secondo caso si dovrà comunque conoscere la durata totale del moto nell'intervallo. Di seguito l'implementazione dei tre metodi e successivamente la parte di main per la scelta dei punti e degli intervalli temporali.

```
%% 3 tipologie per la scelta degli intervalli temporali
function[T]=t_ist(P,Tt,ind)
n=length(P);
if ind==1 % equidistanti
    for k=1:n-1
        T(k)=(Tt)/(n-1);
    end
elseif ind==2 % proporzionali alla differenza di posizione fra punti
    for k=1:n-1
        d(k)=abs(P(k+1)-P(k));
    end
    T=d./(sum(d)).*Tt;
elseif ind==3 % proporzionali alla differenza di posizione alla 1/2
```

```
    u=1/2;%esponente
    for k=1:n-1
        d(k)=abs(P(k+1)-P(k))^u;
    end
    T=d./(sum(d)).*Tt;
end
return
```

2.2.3 Concatenazione funzioni

Avendo ora tutti i dati di ingresso si passa alla creazione dei vari tratti ed infine alla concatenazione. Il primo passo consta nel definire la tipologia di ciascun tratto, ossia assegnare ad ogni tratto la rispettiva funzione codificante la legge di moto desiderata, scelta fra le tre proposte sopra. Quando si vorrà avere una spline il programma richiederà di definire quanti punti vengono interpolati dalla stessa, mentre per le velocità costante o nulla i punti interpolati sono di default due. Fatto ciò si andranno a definire caso per caso le funzioni appena scelte. In particolare si dovrà tenere conto di quanto segue per creare un programma che risponda in modo ottimale a qualsiasi casistica possibile:

- Se il tratto successivo ad una spline è: a velocità costante bisogna che l'accelerazione finale della spline sia nulla mentre la velocità finale sarà imposta uguale a quella del tratto successivo; a velocità nulla bisogna che sia la velocità che l'accelerazione finale della spline siano nulle. Queste casistiche non sono contemplate se la spline si trova come ultimo tratto in cui la velocità e l'accelerazione finale sono inserite con l'input.
- Se il tratto precedente ad una spline è: a velocità costante bisogna che l'accelerazione iniziale della spline sia nulla mentre la velocità iniziale sarà imposta uguale a quella del tratto precedente; a velocità nulla bisogna che sia la velocità che l'accelerazione iniziale della spline siano nulle. Queste casistiche non sono contemplate se la spline si trova come primo tratto in cui la velocità e l'accelerazione iniziale sono inserite con l'input.
- Se si hanno due spline in successione per avere continuità bisognerà che la velocità e l'accelerazione iniziali della seconda siano uguali a quelle finali della prima, in questo modo da input si andranno a definire solo quelle finali della prima.

- Nel caso di tratti a velocità nulla o costante non si dovrà inserire alcun input in quanto le velocità sono già definite dai punti e gli intervalli temporali, o a priori uguali a zero, e le accelerazioni sono per definizione nulle.

Di seguito il codice di programma relativo a quanto appena descritto.

```
%Input della tipologia di interpolazione da effettuare
pos=cell(1,n-1);
vel=cell(1,n-1);
acc=cell(1,n-1);
for k=1:trt
    if type(k)==434
        if k==1
            vi(k)=input('Quanto vale la velocità iniziale?');
            ai(k)=input('Quanto vale la accelerazione iniziale?');
            if type(k+1)==1
                vf(k)=(P(m(3))-P(m(2)))/T(m(2));
                af(k)=0;
            elseif type(k+1)==0
                vf(k)=0;
                af(k)=0;
            elseif type(k+1)==434
                vf(k)=input('Quanto vale la velocità finale della prima
spline?');
                af(k)=input('Quanto vale la accelerazione finale della
prima spline?');
            end
        elseif k==trt
            if type(k-1)==1
                vi(k)=(P(m(end-1))-P(m(end-2)))/T(m(end-2));
                ai(k)=0;
            elseif type(k-1)==0
                vi(k)=0;
                ai(k)=0;
            elseif type(k-1)==434
                vi(k)=vf(k-1);
                ai(k)=af(k-1);
            end
            vf(k)=input('Quanto vale la velocità finale?');
            af(k)=input('Quanto vale la accelerazione finale?');
        else
```

```
            if type(k-1)==1
                vi(k)=(P(m(k))-P(m(k-1)))/T(m(k-1));
                ai(k)=0;
            elseif type(k-1)==0
                vi(k)=0;
                ai(k)=0;
            elseif type(k-1)==434
                vi(k)=vf(k-1);
                ai(k)=af(k-1);
            end
            if type(k+1)==1
                vf(k)=(P(m(k+2))-P(m(k+1)))/T(m(k+1));
                af(k)=0;
            elseif type(k+1)==0
                vf(k)=0;
                af(k)=0;
            elseif type(k+1)==434
                vf(k)=input('Quanto vale la velocità finale della
spline?');
                af(k)=input('Quanto vale la accelerazione finale della
spline?');
            end
        end
        [pos{k},vel{k},acc{k}]=spline434(P(m(k):m(k+1)),T(m(k):m(k+1)-
1),...
            vi(k),vf(k),ai(k),af(k));
    elseif type(k)==1
        v=(P(m(k+1))-P(m(k)))/T(m(k));
        if k==trt
            [pos{k},vel{k},acc{k}]=vcost(P(m(k)),P(m(k+1)),v,...
                0:dt:T(end));
        else
            [pos{k},vel{k},acc{k}]=vcost(P(m(k)),P(m(k+1)),v,...
                0:dt:T(m(k))-dt);
        end
    elseif type(k)==0
        if k==trt
            [pos{k},vel{k},acc{k}]=vcost(P(m(k)),P(m(k+1)),0,...
                0:dt:T(end));
        else
            [pos{k},vel{k},acc{k}]=vcost(P(m(k)),P(m(k+1)),0,...
```

```
                0:dt:T(m(k))-dt);
        end
    end
end
pos=cat(1,pos{:});
vel=cat(1,vel{:});
acc=cat(1,acc{:});
```

Abbiamo così ottenuto la nostra legge che descrive completamente la traiettoria che interpola i punti desiderati con le specifiche imposte. I vettori di posizione, velocità e accelerazione contengono tutti i punti per cui passa la nostra curva. Si passerà quindi ad analizzare il risultato ottenuto per poterne ricavare le informazioni utili per il suo utilizzo applicativo.

2.2.4 Analisi e warning

Una volta che si ha la traiettoria completa che interpola i punti desiderati è importante capire se effettivamente essa resta all'interno di determinati parametri. Se fosse stata usata una polinomiale di grado n-1 per i punti avremmo ottenuto una notevole differenza fra la posizione di passaggio e la posizione massima e minima raggiunte dalla traiettoria; questo crea problemi legati soprattutto all'ingombro ed inoltre ne consegue che anche velocità e accelerazione avrebbero raggiunto picchi indesiderati ed inaspettati. Sulla base di queste considerazioni si decide di operare un'analisi basata sul calcolo dei punti e dei valori di massimo della posizione, velocità e accelerazione e successivamente inserire un messaggio di allarme (warning) nel caso in cui una di esse superi un certo valore limite imposto. Per un'ancor più chiara lettura dei dati si va anche ad estrapolare l'intervallo di punti entro i quali si hanno la velocità e l'accelerazione massima. Si va inoltre a calcolare il valore medio efficacie o *Root Mean Square* ("radice della media del quadrato") di posizione, velocità e accelerazione, ossia il valore che avrebbe la nostra funzione se fosse costante. La funzione che restituisce il valore medio efficace è fra le funzioni predefinite in MATLAB col nome di "rms", in questa trattazione è stata creata una funzione che svolge la medesima funzione che è la seguente.

```
%% Valore Medio Efficace / Root Mean Square
function [eff] = vme(y,t)
eff=sqrt(trapz(t,y.^2)/(t(end)-t(1)));
return
```

La parte del main che implementa l'analisi e il warning è la seguente.

```
%% Analisi e warning
%calcolo del valore medio efficace(root mean square) di posizione,
velocità
%e accelerazione
rmsp=vme(pos,t);
rmsv=vme(vel,t);
rmsa=vme(acc,t);
%calcolo dei valori e della posizione dei massimi e dei minimi della
%posizione, velocità e accelerazione
[max_p,IM_p]=max(pos);
[max_v,IM_v]=max(vel);
[max_a,IM_a]=max(acc);
[min_p,Im_p]=min(pos);
[min_v,Im_v]=min(vel);
[min_a,Im_a]=min(acc);
max_posizione=max(abs(max_p),abs(min_p));
if max_posizione==max_p
    ind_max_pos=IM_p;
    max_posizione=max_p;
elseif max_posizione==abs(min_p)
    ind_max_pos=Im_p;
    max_posizione=min_p;
end
max_velocita=max(abs(max_v),abs(min_v));
if max_velocita==max_v
    ind_max_vel=IM_v;
    max_velocita=max_v;
elseif max_velocita==abs(min_v)
    ind_max_vel=Im_v;
    max_velocita=min_v;
end
max_accelerazione=max(abs(max_a),abs(min_a));
if max_accelerazione==max_a
    ind_max_acc=IM_a;
    max_accelerazione=max_a;
elseif max_accelerazione==abs(min_a)
    ind_max_acc=Im_a;
    max_accelerazione=min_a;
```

```
end
%posizione della massima velocità e accelerazione
t_vmax=t(ind_max_vel);
t_amax=t(ind_max_acc);
sv=1;
while tt(sv)<t_vmax
    sv=sv+1;
end
disp(['La velocità massima si trova fra i punti ',num2str(P(sv-1)),' e
',num2str(P(sv))]);
sa=1;
while tt(sa)<t_amax
    sa=sa+1;
end
disp(['La accelerazione massima si trova fra i punti ',num2str(P(sa-
1)),' e ',num2str(P(sa))]);
%mostra a video i valori di velocità massima e minima
disp(['La velocità massima è:',num2str(abs(max_velocita))]);
disp(['La accelerazione massima è:',num2str(abs(max_accelerazione))]);
% Warning nel caso in cui la posizione o la velocità superino dei valori
% prefissati
sm=0.5;%scotamento massimo ammissibile dal valore limite
if max_p>max(P)+sm
    warning('Interpolazione supera la massima posizione prevista');
end
if min_p<min(P)-sm
    warning('Interpolazione supera la minima posizione prevista');
end
```

2.2.5 Grafici

Per poter avere un riscontro effettivo e un'idea di massima sulle posizioni assunte dalla traiettoria e sull'andamento di velocità e accelerazione, risulta molto utile graficare la legge di moto. Per una miglior comprensione, oltre ad una figura che rappresenta una sotto l'altra tutte e tre le componenti della legge, si andranno anche a plottare una ad una la posizione, la velocità e l'accelerazione in funzione del tempo. Per una lettura migliore del grafico vengono identificati i punti di massimo calcolati al punto precedente e una linea rossa tratteggiata che identifica il valore medio efficace. Nel grafico della posizione vengono poi

indicati i punti inseriti nei dati iniziali da input in modo tale da verificare l'effettiva efficacia dell'interpolazione ed eventuali anomalie.

```
%% Grafici
figure(1)
subplot(3,1,1)
plot(t,pos,'k');
title('Legge di moto');
xlabel('t[s]');
ylabel('posizione[m]');
axis equal
grid on
subplot(3,1,2)
plot(t,vel,'k');
xlabel('t[s]');
ylabel('velocità[m/s]');
axis equal
grid on
subplot(3,1,3)
plot(t,acc,'k');
xlabel('t[s]');
ylabel('accelerazione[m/s^2]');
grid on

figure(2)
plot(t,pos,'b','Linewidth',1.25);
xlabel('t[s]');
ylabel('posizione[m]');
text(t(ind_max_pos)+0.1,max_posizione+0.2,'Posizione massima');
rectangle('Position',[t(ind_max_pos)-0.15,max_posizione-
0.15,0.3,0.3],...
    'Curvature',[1,1],'FaceColor','r');
grid on
axis equal
hold on
plot(t,rmsp.*ones(length(t),1),'r--');
legend('Posizione','rms posizione','Location','Southeast');
hold on
plot(t,zeros(length(t),1),'k--','Linewidth',0.1);
hold on
plot(tt,P,'x','Color','c');
```

```
figure(3)
plot(t,vel,'b','Linewidth',1.25);
xlabel('t[s]');
ylabel('velocità[m/s]');
text(t(ind_max_vel)+0.1,max_velocita+0.2,'Velocità massima');
rectangle('Position',[t(ind_max_vel)-0.15,max_velocita-0.15,0.3,0.3],...
    'Curvature',[1,1],'FaceColor','r');
grid on
axis equal
hold on
plot(t,rmsv.*ones(length(t),1),'r--');
legend('Velocità','rms velocità','Location','Southeast');
hold on
plot(t,zeros(length(t),1),'k--','Linewidth',0.1);

figure(4)
plot(t,acc,'b','Linewidth',1.25);
xlabel('t[s]');
ylabel('accelerazione[m/s^2]');
text(t(ind_max_acc)+0.1,max_accelerazione+0.2,'Accelerazione massima');
rectangle('Position',[t(ind_max_acc)-0.15,max_accelerazione-
0.15,0.3,0.3],...
    'Curvature',[1,1],'FaceColor','r');
grid on
axis equal
hold on
plot(t,rmsa.*ones(length(t),1),'r--');
legend('Accelerazione','rms accelerazione','Location','Southeast');
hold on
plot(t,zeros(length(t),1),'k--','Linewidth',0.1);
```

2.3 Esempio

Poiché quello delle spline 4-3-4 e della creazione di traiettorie è l'argomento più importante della trattazione, e quello per il quale si son studiate più casistiche per implementare al meglio l'm-file descritto fin qui, si è deciso di dedicare un paragrafo intero mostrando il funzionamento del file "Main.m" e come questo si presenta al momento in cui viene lanciato in MATLAB. L'esempio è del tutto generico e si andranno a coprire un po' tutte

le combinazioni possibili, tutte all'interno della stessa traiettoria. Verrà mostrata la schermata della *Command window* relativa agli input e successivamente agli output ed infine i grafici relativi alla legge di moto creata. Si è scelto di creare 11 tratti per interpolare 30 punti. I punti saranno scelti da input mentre gli intervalli temporali saranno scelti con una delle tre tipologie descritte al §2.2.1 e in un caso inseriti da input. Le spline interpoleranno numeri di punti diversi e si affronteranno i casi in cui si hanno due spline consecutive. Di seguito viene riportato quanto mostrato nella *Command window*:

Inserire il numero di tratti 11
Quanti punti da interpolare? 30
Si avrà un numero di punti pari a:
30

e un numero di intervalli pari a:
29

Inserire i punti da interpolare
1°-->0
2°-->0.6
3°-->0.8
4°-->1.5
5°-->0.3
6°-->-0.8
7°-->-1.3
8°-->0.1
9°-->0.5
10°-->1
11°-->1
12°-->1.6
13°-->2.2
14°-->2.5
15°-->0
16°-->-0.8
17°-->-1.9

18°-->-2

19°-->-0.5

20°-->0

21°-->0.2

22°-->1.6

23°-->1.9

24°-->2.5

25°-->2

26°-->2

27°-->2

28°-->1.1

29°-->1.1

30°-->0

Inserire tipologia del 1° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 1

Inserire lunghezza temporale tratto a velocità costante o nulla 1

Inserire tipologia del 2° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 5

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)

0

Quanto è la durata del moto nel tratto? 3

Scegliere il metodo di spaziatura fra i punti

1->equidistate 2->proporzionale alla posizione 3->proporzionale alla posizione alla 1/2 2

Inserire tipologia del 3° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 5

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)

0

Quanto è la durata del moto nel tratto? 3

Scegliere il metodo di spaziatura fra i punti

1->equidistate 2->proporzionale alla posizione 3->proporzionale alla posizione alla 1/2 3

Inserire tipologia del 4° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 0

Inserire lunghezza temporale tratto a velocità costante o nulla 2

Inserire tipologia del 5° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 4

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)

0

Quanto è la durata del moto nel tratto? 4

Scegliere il metodo di spaziatura fra i punti

1->equidistate 2->proporzionale alla posizione 3->proporzionale alla posizione alla 1/2 1

Inserire tipologia del 6° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 1

Inserire lunghezza temporale tratto a velocità costante o nulla 1.2

Inserire tipologia del 7° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 8

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)

0

Quanto è la durata del moto nel tratto? 5

Scegliere il metodo di spaziatura fra i punti

1->equidistate 2->proporzionale alla posizione 3->proporzionale alla posizione alla 1/2 3

Inserire tipologia del 8° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 4

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)

0

Quanto è la durata del moto nel tratto? 4

Scegliere il metodo di spaziatura fra i punti

1->equidistate 2->proporzionale alla posizione 3->proporzionale alla posizione alla 1/2 2

Inserire tipologia del 9° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 0

Inserire lunghezza temporale tratto a velocità costante o nulla 1.5

Inserire tipologia del 10° tratto

434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 434

Quanti punti interpola la spline? 4

Si desidera inserire manualmente la lunghezza di ogni intervallo temporale: (Sì->1 No->0)
1
Inserire la lunghezza di ogni intervallo temporale 1.3
Inserire la lunghezza di ogni intervallo temporale 1.2
Inserire la lunghezza di ogni intervallo temporale 1.5
Inserire tipologia del 11° tratto
434->SPLINE434,1->VEL COSTANTE,0->VEL NULLA: 1
Inserire lunghezza temporale tratto a velocità costante o nulla 1.3
Quanto vale la velocità finale della spline del 2° tratto?0.5
Quanto vale la accelerazione finale della spline del 2° tratto?0
Quanto vale la velocità finale della spline del 7° tratto?1
Quanto vale la accelerazione finale della spline del 7° tratto?1.2

E dà come risultati:

La velocità massima si trova fra i punti 0.3 e -0.8

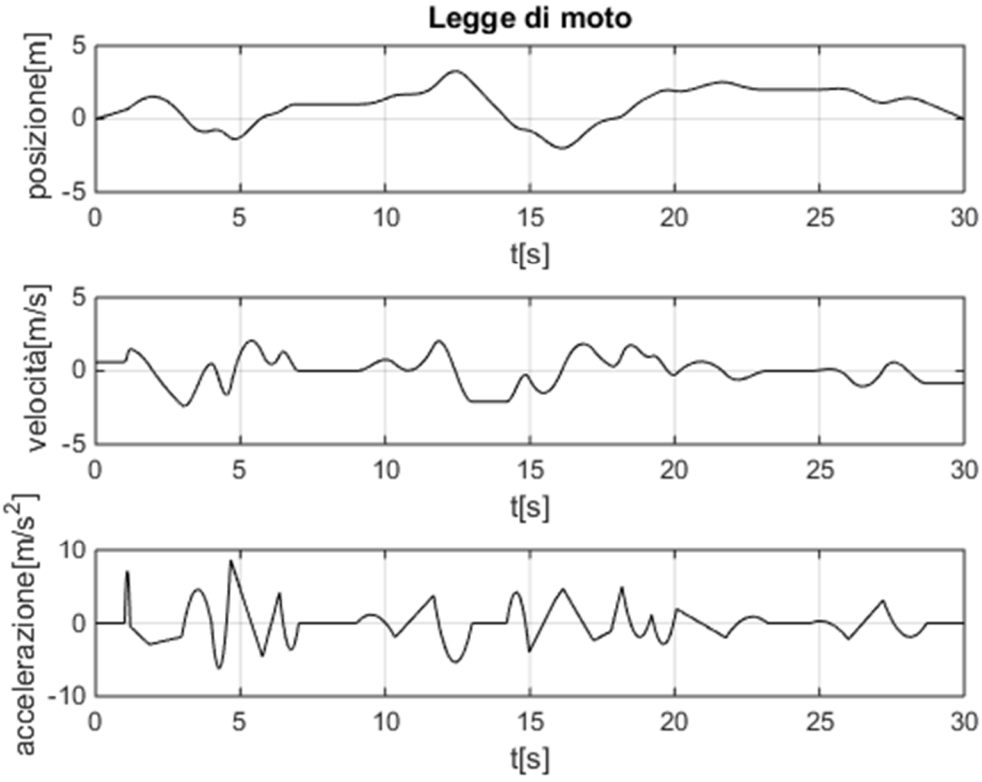
La accelerazione massima si trova fra i punti -0.8 e -1.3

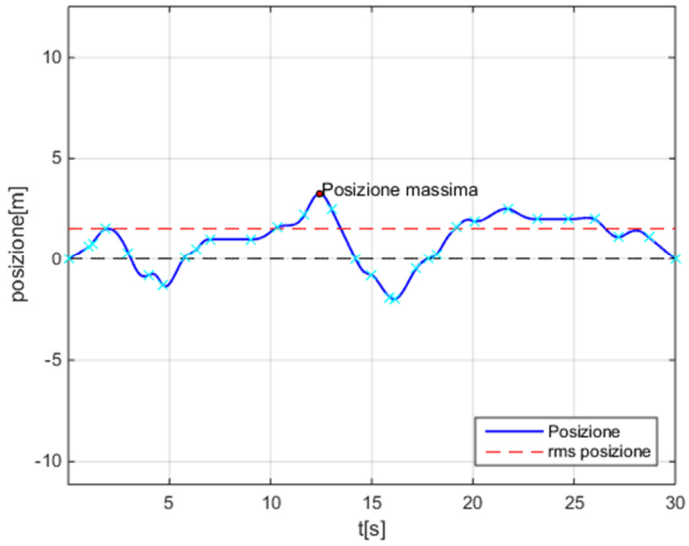
La velocità massima è:2.3941m/s

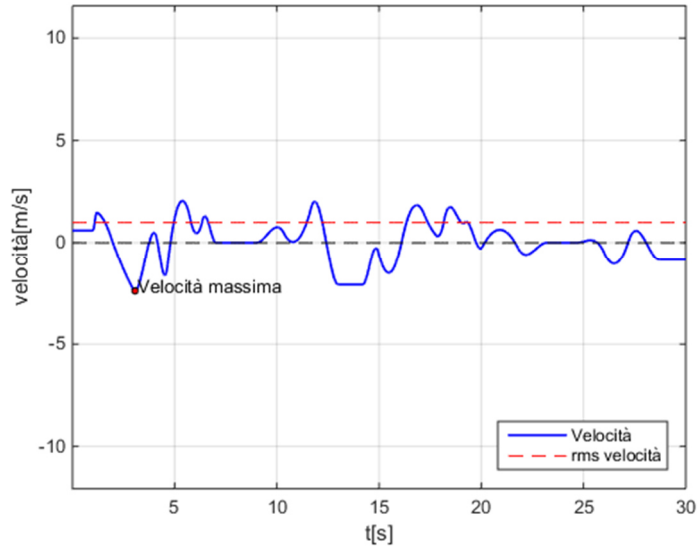
La accelerazione massima è:8.6718m/s^2

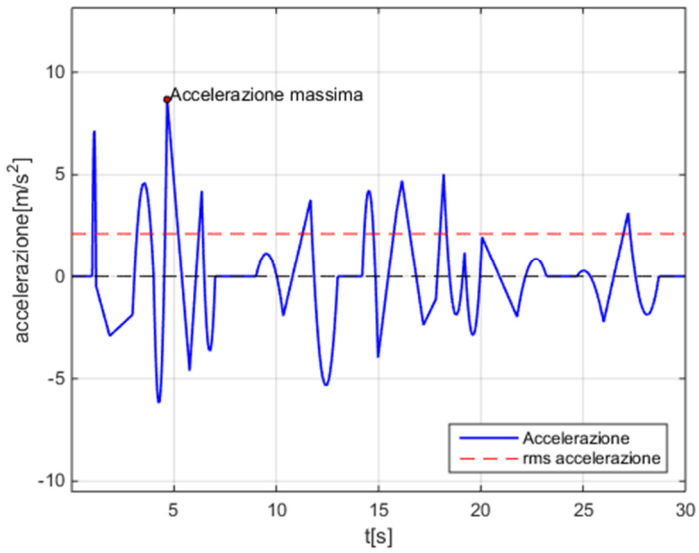
Warning: Interpolazione supera la massima posizione prevista

E i seguenti grafici:




Posizione massima
posizione[m]
t[s]
Posizione
rms posizione

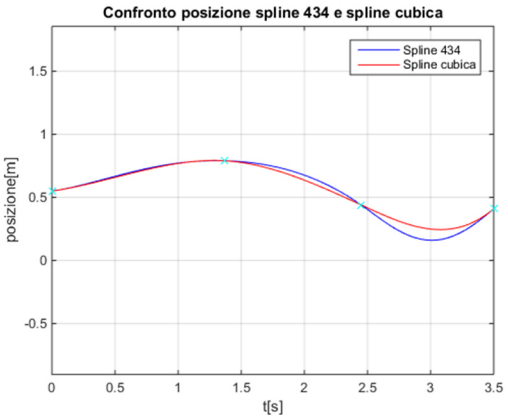

Velocità massima
velocità[m/s]
t[s]
Velocità
rms velocità

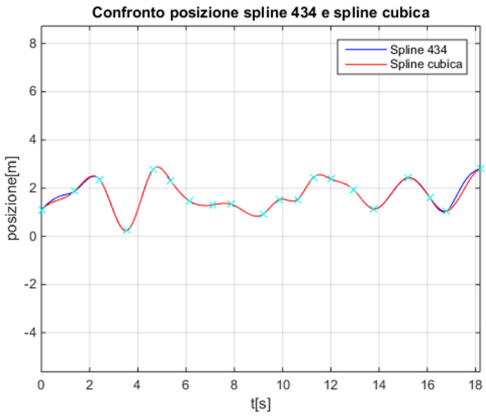

Accelerazione massima
accelerazione[m/s^2]
t[s]
Accelerazione
rms accelerazione

2.4 Confronto Spline 4-3-4 e Spline Cubica

Come detto ad inizio trattazione si è utilizzata nell'interpolazione dei punti una tipologia di spline detta 4-3-4 in quanto agli istanti iniziali e finali utilizza una legge polinomiale di 4° grado mentre nei tratti centrali una di 3° grado. Le spline che vengono di norma usate però utilizzano un polinomio di 3° grado anche nei tratti iniziali e finali e di seguito si andranno ad analizzare le differenze che ciò comporta nella creazione della traiettoria. Per farlo si è dovuto creare una nuova implementazione della function per i coefficienti che viene derivata nello stesso modo con cui si è trovata quella per la 4-3-4. Fatto ciò è semplice implementare una function per la spline cubica, che per motivi di ridondanza non viene presentata vista la somiglianza all'altra tipologia di spline. Per poter studiare il confronto si son provate diverse soluzioni utilizzando punti e intervalli random e da tutte le prove se ne sono estrapolate delle considerazioni. Ci si servirà di due casi fra i tanti trovati che saranno significativi nella spiegazione, dei quali il primo prende a riferimento la più semplice delle spline, formata da tre tratti, mentre il secondo contiene diciannove tratti. Si userà un intervallo di punti di [0;1] e di lunghezza degli istanti temporali [0.5;1.5] sia per il primo che il secondo caso. In sequenza si andranno a vedere e commentare i risultati relativi a posizione, velocità e accelerazione.

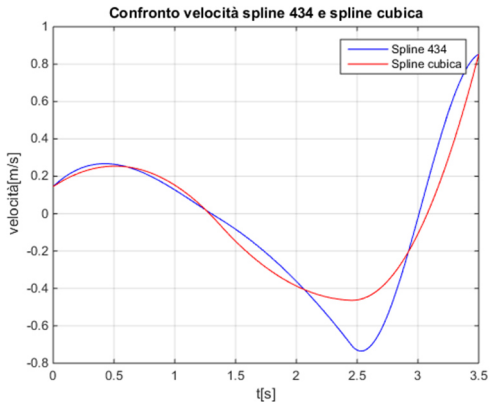
2.4.1 Posizione

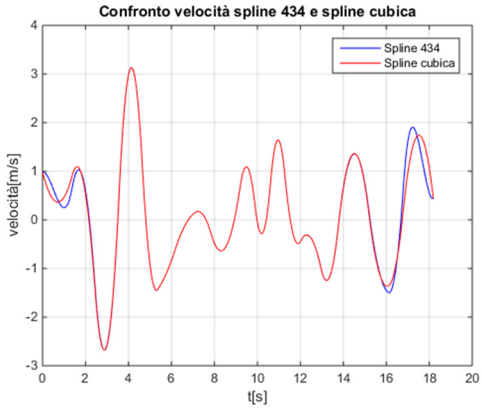




Come già accennato all'inizio della trattazione si può subito notare in entrambe come nei tratti agli estremi la spline cubica abbia un andamento più dolce rispetto alla 4-3-4 dato appunto dal fatto che vengono usati polinomi di grado inferiore per l'interpolazione. Ciò permette quindi di non avere picchi che si discostano troppo dai punti prefissati. Tuttavia le differenze fra le due non sono tali da preferire una all'altra per ora.

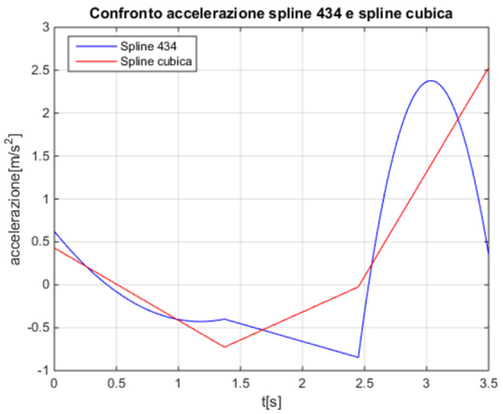
2.4.2 Velocità

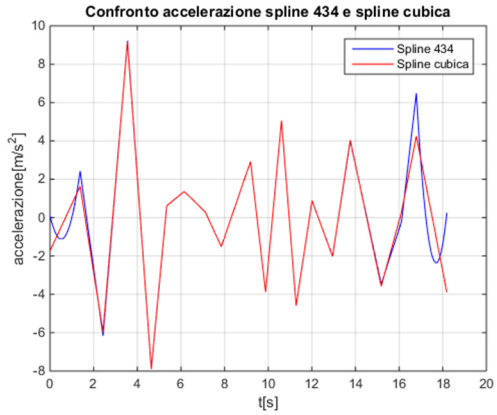




Partendo dal fatto che le velocità iniziali e finali di entrambe le spline sono uguali ed imposte, è immediato notare, com'era del resto prevedibile dalle considerazioni sui grafici della posizione, come la 4-3-4 presenti picchi di velocità maggiori rispetto alla spline cubica; ciò potrebbe essere fonte di problemi nelle applicazioni reali. Viene però trascurato questo fatto in quanto la differenza dei picchi non è così significativa. Si fa inoltre notare che dall'analisi dell'espressione del polinomio di 4° grado rispetto a quello di 3°, il primo presenta una velocità espressa da un polinomio di grado 2 e quindi avrà al suo interno un massimo e anche un minimo locali, mentre il secondo ha velocità parabolica e quindi ha solamente un massimo (o minimo) locale. Come si vedrà in seguito, si possono utilizzare metodi di ottimizzazione delle spline imponendo velocità massime ammissibili e quindi eliminare questo problema di picchi inaspettati.

2.4.3 Accelerazione





Per l'accelerazione le differenze fra le due spline sono molto marcate. Partendo analizzando la spline cubica si vede chiaramente che l'accelerazione agli estremi è totalmente

incontrollata, e ciò deriva dal fatto che con un polinomio di 3° grado si ha la possibilità di imporre le condizioni che interessano solo la derivata prima. Il problema sorge quando si è in applicazioni reali poiché l'accelerazione potrebbe raggiungere valori incompatibili con quelli di progetto, in particolar modo perché questi picchi incontrollati sono presenti ad inizio e fine movimento, che in gran parte degli utilizzi sono tratti critici. Per ovviare a questo problema si passa quindi alla spline 4-3-4 nella quale i polinomi di 4° grado permettono di imporre l'accelerazione iniziale e finale, che nei casi trattati è scelta random in un intervallo. Da un'analisi grafica si ricava che la differenza di accelerazione agli estremi è superiore anche di un fattore 10 fra le tue tipologie di spline. Come prevedibile però la spline 4-3-4 raggiunge successivamente picchi più elevati rispetto a quella cubica. Con identico ragionamento fatto per le velocità si nota che con la polinomiale di 4° grado l'accelerazione è parabolica e quindi presenterà un massimo (o minimo) locale. Come si è fatto finora, il controllo sull'accelerazione iniziale e finale fa propendere la scelta sulla spline 4-3-4 che soddisfa maggiormente i vincoli delle applicazioni reali.

Capitolo 3

Ottimizzazione tempo totale spline

Come si era accennato nel capitolo precedente, vi è la possibilità, data una spline qualsiasi, di ottimizzare il tempo di percorrenza del movimento attraverso i punti fissati, mantenendo la velocità e l'accelerazione all'interno di limiti imposti. Di seguito quindi si andranno ad esporre le fasi sia teoriche e analitiche che quelle legate all'implementazione MATLAB del processo di ottimizzazione. Si precisa che a scopo esemplificativo si considerano spline che interpolano quattro punti; per avere lo stesso risultato con un maggior numero di punti le considerazioni da fare sono comunque le stesse. Si avrà infine modo di confrontare l'ottimizzazione eseguita su una spline cubica e quella su una spline 4-3-4; inoltre un confronto sarà eseguito anche per le tre tipologie di scelta degli intervalli temporali introdotte al capitolo relativo alla creazione dei dati sul main (§2.2.1).

Lo scopo sarà di pianificare una traiettoria spline che interpola quattro punti in modo tale da minimizzarne il tempo totale T, dato dalla somma dei tempi parziali di ogni tratto, restando all'interno dei vincoli imposti alla velocità e all'accelerazione. Tale problema di ottimizzazione è di tipo non lineare. Si può scrivere la funzione da minimizzare come:

$$f(T)=T_1+T_2+T_3$$

Quindi il problema sarà: min $\{T= T_1+T_2+T_3\}$.

Ora si andranno a definire tutti i vincoli che faranno in modo che la spline resti all'interno della velocità e l'accelerazione imposte, che saranno chiamate nell'esempio v_{max} (=3m/s) e a_{max} (=2m/s^2). Per primi saranno esposti quelli della spline cubica, e a seguire quelli della spline 4-3-4; i richiami teorici iniziali necessari alla stesura delle condizioni in forma di disequazioni ed equazioni saranno validi però per entrambi i casi. Si precisa già da ora che tali vincoli non saranno sufficienti per l'implementazione MATLAB, ma sono quelli necessari per la comprensione del problema.

Innanzitutto si partirà imponendo che le velocità e le accelerazioni iniziali e finali di ogni tratto dovranno essere minori o uguali ai valori massimi prefissati. Come analizzato nei

precedenti capitoli si ricorda che: sulla spline cubica si ha la libertà di scegliere le velocità agli estremi (v_i e v_f); sulla spline 4-3-4 si ha la libertà di scegliere la velocità e l'accelerazione agli estremi (v_i, v_f, a_i, a_f); tutte nell'esempio assumeranno valore nullo. Da ciò se ne deduce che si avranno per la prima: 2 uguaglianze e 13 disuguaglianze; per la seconda: 4 uguaglianze e 13 disuguaglianze. Quindi si dovrà imporre che la velocità e l'accelerazione all'interno dei tratti siano minore o uguale a quelle consentite. Basterà che nel punto di massimo (o minimo) delle rispettive curve il valore sia inferiore o al più uguale a quello limite; sappiamo che nei punti di massimo (o minimo) locale di una curva la derivata è sempre nulla, quindi basterà che il vincolo sia posto in quel determinato istante temporale per verificare così tutto il tratto. Si vedranno i due casi applicati alle spline in modo separato poiché come già si accennava le relative curve hanno andamenti e punti critici diversi.

Per poter risolvere il problema di ottimizzazione in MATLAB si userà la funzione già presente nel software *fmincon* (o in alternativa *l'Optimization Toolbox*). In ingresso vengono richiesti:

1) La funzione da minimizzare espressa in forma di parametri x, ossia $x(1)=T_1$ e così via; quindi si creerà la *function* seguente:

```
function[y]=funzione(x)
y=x(1)+x(2)+x(3);
end
```

2) La soluzione di primo tentativo nella quale i primi parametri "x" saranno gli intervalli temporali ricavati da uno dei tre metodi descritti nel paragrafo iniziale del main. I successivi parametri saranno i coefficienti della funzione polinomiale di ogni tratto calcolata con gli intervalli ricavati al passo precedente. Quindi si avrà nel nostro caso che $x(4)=a_{11}$, $x(5)=a_{12}$ e così via, prestando attenzione che nella spline 4-3-4 si hanno due parametri in più che nella spline cubica. I parametri dati in ingresso saranno gli stessi che alla fine delle iterazioni daranno il risultato trovato.
3) Le matrici A, b e Aeq, beq rispettivamente codificanti i vincoli di disuguaglianza e uguaglianza lineare.
4) Il limite superiore e inferiore di ciascun parametro. Nel nostro caso viene imposto ai primi tre parametri (gli intervalli temporali) di essere positivi, mentre per gli altri si pone limite ad infinito.

5) La funzione contenente tutti i vincoli di uguaglianza e disuguaglianza, che sarà diversa nei due casi trattati e sarà descritta dettagliatamente in seguito.

3.1 Spline Cubica

Poiché l'accelerazione è costante (la velocità è lineare), non si avranno vincoli legati all'accelerazione massima all'interno dei tratti, ma solo quelli nei punti iniziali e finali di ogni tratto. La velocità nel primo tratto è: $a_{11}+2*a_{12}(t-t_1)+3*a_{13}(t-t_1)^2$, quindi per trovare $(t-t_1)$ nel punto di massimo basterà derivare tale espressione e porla uguale a zero. Si ottiene: $2*a_{12}+6*a_{13}(t-t_1)=0$ e quindi: $(t-t_1)=-a_{12}/a_{13}$. Da cui si ricava che la velocità massima nel primo tratto sarà: $a_{11}+2*a_{12}*(-a_{12}/a_{13})+3*a_{13}(-a_{12}/a_{13})^2$. Allo stesso modo per il secondo e terzo tratto. Scrivendo quindi tutti i vincoli da considerare si avrà il seguente sistema di disequazioni:

$a_{21}\leq v_{max}$ (velocità iniziale 2° tratto)

$a_{31}\leq v_{max}$ (velocità iniziale 3° tratto)

$2*a_{12}\leq a_{max}$ (accelerazione iniziale 1° tratto)

$2*a_{22}\leq a_{max}$ (accelerazione iniziale 2° tratto)

$2*a_{32}\leq a_{max}$ (accelerazione iniziale 3° tratto)

$a_{11}+2*a_{12}*T_1+3*a_{13}T_1^2\leq v_{max}$ (velocità finale 1° tratto)

$a_{21}+2*a_{22}*T_2+3*a_{23}T_2^2\leq v_{max}$ (velocità finale 2° tratto)

$2*a_{12}+6*a_{13}T_1\leq a_{max}$ (accelerazione finale 1° tratto)

$2*a_{22}+6*a_{23}T_2\leq a_{max}$ (accelerazione finale 2° tratto)

$2*a_{32}+6*a_{33}T_3\leq a_{max}$ (accelerazione finale 3° tratto)

$a_{11}+2*a_{12}*(-a_{12}/a_{13})+3*a_{13}(-a_{12}/a_{13})^2\leq v_{max}$ (velocità massima nel 1° tratto)

$a_{21}+2*a_{22}*(-a_{22}/a_{23})+3*a_{23}(-a_{22}/a_{23})^2\leq v_{max}$ (velocità massima nel 2° tratto)

$a_{31}+2*a_{32}*(-a_{32}/a_{33})+3*a_{33}(-a_{32}/a_{33})^2\leq v_{max}$ (velocità massima nel 3° tratto)

e le equazioni:

$a_{11}=v_i$ (velocità iniziale 1° tratto)

$a_{31}+2*a_{32}*T_3+3*a_{33}T_3^2=v_f$ (velocità finale 3° tratto)

Per implementare in MATLAB queste condizioni basterà esprimerle sostituendo ai coefficienti e ai tempi i relativi parametri "x" ed esprimere ogni dis/equazione tale che a destra dell'operatore ci sia zero. Perché l'ottimizzazione sia efficace essa dovrà avere all'interno anche le condizioni proposte nel paragrafo riguardante il calcolo dei coefficienti (§2.1.1) a proposito del passaggio per i punti dati e la continuità di velocità e accelerazione fra i tratti. Per motivi di ridondanza vengono omesse le espressioni in quanto già presenti nella trattazione. Di seguito la function che implementa quanto appena spiegato.

```
%% Vincoli non lineari di uguaglianza e disuguaglianza spline classica
function[c,ceq]=nonlincon(x)
%disuguaglianze
c(1)=x(4)+2*x(5)*x(1)+3*x(6)*x(1)^2-3;
c(2)=x(7)+2*x(8)*x(2)+3*x(9)*x(2)^2-3;
c(3)=x(4)-1/3*(x(5)^2/x(6))-3;
c(4)=x(7)-1/3*(x(8)^2/x(9))-3;
c(5)=x(10)-1/3*(x(11)^2/x(12))-3;
c(6)=2*x(5)+6*x(6)*x(1)-2;
c(7)=2*x(8)+6*x(9)*x(2)-2;
c(8)=2*x(11)+6*x(12)*x(3)-2;
%uguaglianze
ceq(1)=x(10)+2*x(11)*x(3)+3*x(12)*x(3)^2;
%condizioni di passaggio
ceq(2)=x(4)*x(1)+x(5)*x(1)^2+x(6)*x(1)^3-2;
ceq(3)=2+x(7)*x(2)+x(8)*x(2)^2+x(9)*x(2)^3-12;
ceq(4)=12+x(10)*x(3)+x(11)*x(3)^2+x(12)*x(3)^3-5;
%condizioni di continuità
ceq(5)=x(4)+2*x(5)*x(1)+3*x(6)*x(1)^2-x(7);
ceq(6)=x(7)+2*x(8)*x(2)+3*x(9)*x(2)^2-x(10);
ceq(7)=2*x(5)+6*x(6)*x(1)-2*x(8);
ceq(8)=2*x(8)+6*x(9)*x(2)-2*x(11);
end
```

E quindi il codice per l'ottimizzazione:

```
f=@funzione;
nonlcon=@nonlincon;
vma=3;%[m/s]
ama=2;%[m/s^2]
%vincoli lineari di disuguaglianza
A=zeros(10,12);
A(1,7)=1; A(2,10)=1;
```

```
A(3,5)=2; A(4,8)=2; A(5,11)=2;
A(6,7)=-1; A(7,10)=-1;
A(8,5)=-2; A(9,8)=-2;A(10,11)=-2;
b=[3,3,2,2,2,3,3,2,2,2]';
%vincoli lineari di uguaglianza
Aeq=[0,0,0,1,0,0,0,0,0,0,0,0];
beq=0;
%limiti
lbc=(-Inf).*ones(9,1);
lb=[0.1;0.1;0.1;lbc];
ub=[];
[x,fval,exitflag,output]=fmincon(f,x0,A,b,Aeq,beq,lb,ub,nonlcon);
```

3.2 Spline 4-3-4

In questo caso l'accelerazione nel primo e nell'ultimo tratto ha andamento parabolico e quindi presenta un massimo (o minimo) locale in quelle zone. Nel tratto centrale invece si comporta come visto nel caso precedente. L'accelerazione nel 1° tratto è: $2*a_{12}+6*a_{13}(t-t_1)+12*a_{14}(t-t_1)^2$ che derivato per trovare il valore di $(t-t_1)$ dà: $6*a_{13}(t-t_1)+24*a_{14}(t-t_1)^2=0$ e quindi: $(t-t_1)=-1/4*(a_{13}/a_{14})$ e allo stesso modo per trovare il valore relativo al 3° tratto. La velocità nel 1° tratto è: $a_{11}+2*a_{12}(t-t_1)+3*a_{13}(t-t_1)^2+4*a_{14}(t-t_1)^3$ che derivata e posta uguale a zero restituisce un equazione di grado 2: $2*a_{12}+6*a_{13}(t-t_1)+12*a_{14}(t-t_1)$ e quindi avrà sia un massimo che un minimo locale e si dovranno calcolare due valori di $(t-t_1)$ da sostituire nei vincoli. Con un semplice passaggio si ricavano i valori:

$$(t-t_1)_{1,2}=\frac{-3*a_{13}\pm\sqrt{9*a_{13}^2-24*a_{12}*a_{14}}}{12*a_{14}}$$

Allo stesso modo si esegue il calcolo per il 3° tratto. Si può dunque scrivere il sistema disequazioni corrispondenti ai vincoli:

$a_{21}\leq v_{max}$ (velocità iniziale 2° tratto)

$a_{31}\leq v_{max}$ (velocità iniziale 3° tratto)

$2*a_{22}\leq a_{max}$ (accelerazione iniziale 2° tratto)

$2*a_{32}\leq a_{max}$ (accelerazione iniziale 3° tratto)

$a_{11}+2*a_{12}*T_1+3*a_{13}T_1^2+4*a_{14}T_1^3\leq v_{max}$ (velocità finale 1° tratto)

$a_{21}+2*a_{22}*T_2+3*a_{23}T_2^2\leq v_{max}$ (velocità finale 2° tratto)

$2*a_{12}+6*a_{13}T_1+12*a_{14}T_1^2 \leq a_{max}$ (accelerazione finale 1° tratto)

$2*a_{22}+6*a_{23}T_2 \leq a_{max}$ (accelerazione finale 2° tratto)

$a_{11}+2*a_{12}*\left(\frac{-3*a_{13}\pm\sqrt{9*a_{13}^2-24*a_{12}*a_{14}}}{12*a_{14}}\right)+3*a_{13}\left(\frac{-3*a_{13}\pm\sqrt{9*a_{13}^2-24*a_{12}*a_{14}}}{12*a_{14}}\right)^2 \leq v_{max}$ (velocità massima nel 1° tratto)

$a_{21}+2*a_{22}*(-a_{22}/a_{23})+3*a_{23}(-a_{22}/a_{23})^2 \leq v_{max}$ (velocità massima nel 2° tratto)

$a_{31}+2*a_{32}*\left(\frac{-3*a_{33}\pm\sqrt{9*a_{33}^2-24*a_{32}*a_{34}}}{12*a_{34}}\right)+3*a_{33}\left(\frac{-3*a_{33}\pm\sqrt{9*a_{33}^2-24*a_{32}*a_{34}}}{12*a_{34}}\right)^2 \leq v_{max}$ (velocità massima nel 3° tratto)

$2*a_{12}+6*a_{13}(-1/4*(a_{13}/a_{14}))+12*a_{14}(-1/4*(a_{13}/a_{14}))^2 \leq a_{max}$ (accelerazione massima nel 1° tratto)

$2*a_{32}+6*a_{33}(-1/4*(a_{33}/a_{34}))+12*a_{34}(-1/4*(a_{33}/a_{34}))^2 \leq a_{max}$ (accelerazione massima nel 3° tratto)

E le equazioni:

$a_{11}=v_i$ (velocità iniziale 1° tratto)

$a_{31}+2*a_{32}*T_3+3*a_{33}T_3^2+4*a_{34}T_3^3=v_f$ (velocità finale 3° tratto)

$2*a_{12}=a_i$ (accelerazione iniziale 1° tratto)

$2*a_{32}+6*a_{33}T_3+12*a_{34}T_3^2=a_f$ (accelerazione finale 3° tratto)

Implementandole in MATLAB si avrà la function:

```
%% Vincoli non lineari di uguaglianza e disuguaglianza spline 434
function[c,ceq]=nonlincon434(x)
%disuguaglianze
c(1)=x(4)+2*x(5)*x(1)+3*x(6)*x(1)^2+4*x(7)*x(1)^3-3;
c(2)=x(8)+2*x(9)*x(2)+3*x(10)*x(2)^2-3;
c(3)=2*x(5)+6*x(6)*x(1)+12*x(7)*x(1)^2-2;
c(4)=2*x(9)+6*x(10)*x(2)-2;
c(5)=x(4)+2*x(5)*((-3*x(6)+sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))...
    +3*x(6)*((-3*x(6)+sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))^2 ...
    +4*x(7)*((-3*x(6)+sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))^3-3;
c(6)=x(4)+2*x(5)*((-3*x(6)-sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))...
    +3*x(6)*((-3*x(6)-sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))^2 ...
    +4*x(7)*((-3*x(6)-sqrt(9*x(6)^2-24*x(5)*x(7)))/12/x(7))^3-3;
c(7)=x(8)-1/3*(x(9)^2/x(10))-3;
```

```
c(8)=x(11)+2*x(12)*((-3*x(13)+sqrt(9*x(13)^2-
24*x(12)*x(14)))/12/x(14))...
    +3*x(13)*((-3*x(13)+sqrt(9*x(13)^2-24*x(12)*x(14)))/12/x(14))^2 ...
    +4*x(14)*((-3*x(13)+sqrt(9*x(13)^2-24*x(12)*x(14)))/12/x(14))^3-3;
c(9)=x(11)+2*x(12)*((-3*x(13)-sqrt(9*x(13)^2-
24*x(12)*x(14)))/12/x(14))...
    +3*x(13)*((-3*x(13)-sqrt(9*x(13)^2-24*x(12)*x(14)))/12/x(14))^2 ...
    +4*x(14)*((-3*x(13)-sqrt(9*x(13)^2-24*x(12)*x(14)))/12/x(14))^3-3;
c(10)=2*x(5)+6*x(6)*(-x(6)/4/x(7))+12*x(7)*(-x(6)/4/x(7))^2-2;
c(11)=2*x(12)+6*x(13)*(-x(13)/4/x(14))+12*x(14)*(-x(13)/4/x(14))^2-2;
%uguaglianze
ceq(1)=x(11)+2*x(12)*x(3)+3*x(13)*x(3)^2+4*x(14)*x(3)^3;
ceq(2)=2*x(12)+6*x(13)*x(3)+12*x(14)*x(3)^2;
%condizioni di passaggio
ceq(3)=x(4)*x(1)+x(5)*x(1)^2+x(6)*x(1)^3+x(7)*x(1)^4-2;
ceq(4)=2+x(8)*x(2)+x(9)*x(2)^2+x(10)*x(2)^3-12;
ceq(5)=12+x(11)*x(3)+x(12)*x(3)^2+x(13)*x(3)^3+x(14)*x(3)^4-5;
%condizioni di continuità
ceq(6)=x(4)+2*x(5)*x(1)+3*x(6)*x(1)^2+4*x(7)*x(1)^3-x(8);
ceq(7)=x(8)+2*x(9)*x(2)+3*x(10)*x(2)^2-x(11);
ceq(8)=2*x(5)+6*x(6)*x(1)+12*x(7)*x(1)^2-2*x(9);
ceq(9)=2*x(9)+6*x(10)*x(2)-2*x(12);
end
```

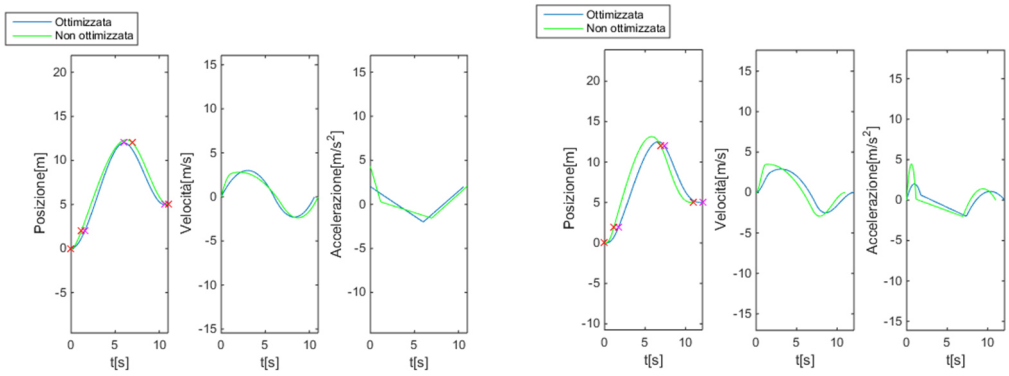
E quindi il codice per l'ottimizzazione:

```
f=@funzione;
nonlcon=@nonlincon434;
%vincoli lineari di disuguaglianza
A=zeros(8,14);
A(1,8)=1; A(2,11)=1;
A(3,9)=2; A(4,12)=2;
A(5,8)=-1; A(6,11)=-1;
A(7,9)=-2; A(8,12)=-2;
b=[3,3,2,2,3,3,2,2]';
%vincoli lineari di uguaglianza
Aeq=[0,0,0,1,0,0,0,0,0,0,0,0,0,0
     0,0,0,0,2,0,0,0,0,0,0,0,0,0];
beq=[0;0];
%limiti
lbc=(-Inf).*ones(11,1);
lb=[1;1;1;lbc];
```

```
ub=[];

[x,fval,exitflag,output]=fmincon(f,x0,A,b,Aeq,beq,lb,ub,nonlcon);
```

3.3 Confronto

Si andrà ora a fare un confronto prima sui risultati dell'ottimizzazione per entrambi i tipi di spline, in modo da concludere quanto detto al capitolo precedente; poi si andranno a confrontare i valori ottenuti partendo da diverse condizioni iniziali, ed in particolare considerando i tre tipi di scelta degli intervalli temporali. Nella trattazione è stata omessa la parte di codice MATLAB che codifica con l'inserimento dei dati, in quanto è di semplice sviluppo, e la parte di analisi e grafici, anch'essa di immediata stesura. I punti che andrà ad interpolare la spline saranno nell'ordine: 0, 2, 12, 5; e il tempo di percorrenza totale è stato scelto di 11 secondi. Si può ipotizzare da quanto detto al capitolo precedente che la spline 4-3-4 avrà una velocità massima maggiore e quindi necessiterà di un tempo totale ottimizzato maggiore della spline cubica per rientrare nei limiti di velocità imposti. Di seguito i grafici di posizione, velocità e accelerazione relativi rispettivamente alla spline cubica e alla spline 4-3-4.



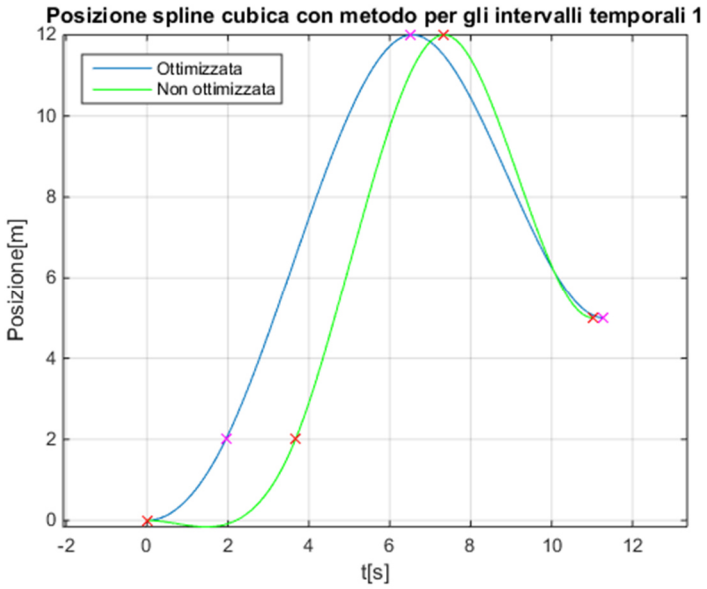
Come si era giustamente ipotizzato la spline 4-3-4 raggiunge una velocità massima più elevata. Dal confronto fra le due si può ben notare dalla figura come nel primo caso il tempo totale dopo l'ottimizzazione sia minore del tempo iniziale, dovuto al fatto che la spline cubica non superava il limite di velocità imposto, mentre nel secondo caso avviene l'inverso, ossia che il tempo totale ottimizzato risulta essere maggiore del tempo iniziale. Questo fatto è particolarmente rilevante in applicazioni reali in quanto bisogna considerare che non sempre l'ottimizzazione porta a una riduzione del tempo rispetto a quella scelta di

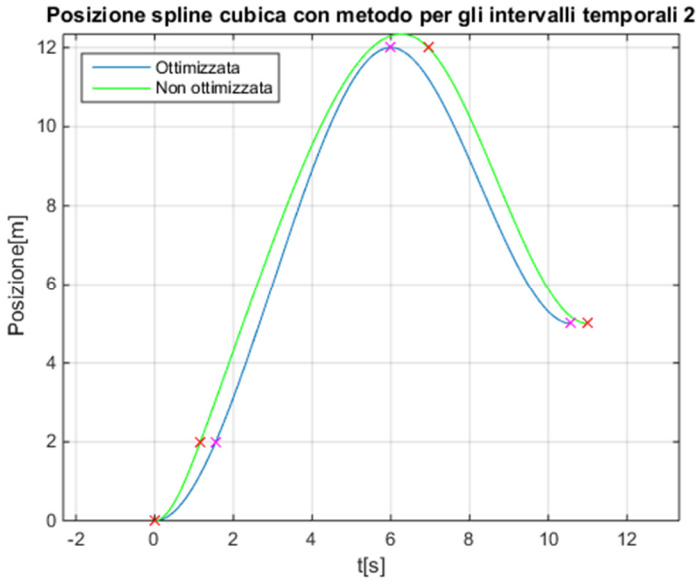
progetto; si dovrà quindi fare attenzione alle velocità e accelerazioni massime che si vanno ad imporre per trovare il compromesso giusto con il tempo che sarà calcolato.

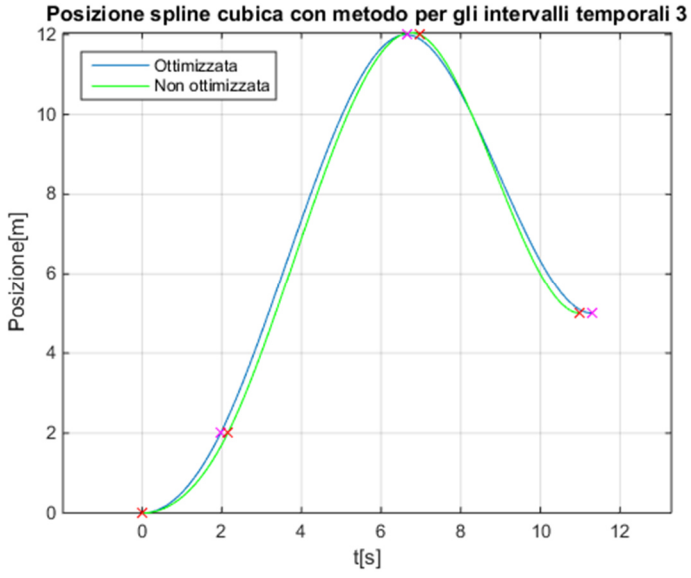
Si passa ora ad analizzare la differenza di valori post ottimizzazione al variare dei parametri di primo tentativo inseriti. Nei tre casi si useranno i tre metodi di spaziatura fra intervalli temporali già visti, ossia: equi spaziati, proporzionali alla distanza fra i punti e proporzionali alla radice della distanza fra i punti. Di seguito la tabella con i risultati e

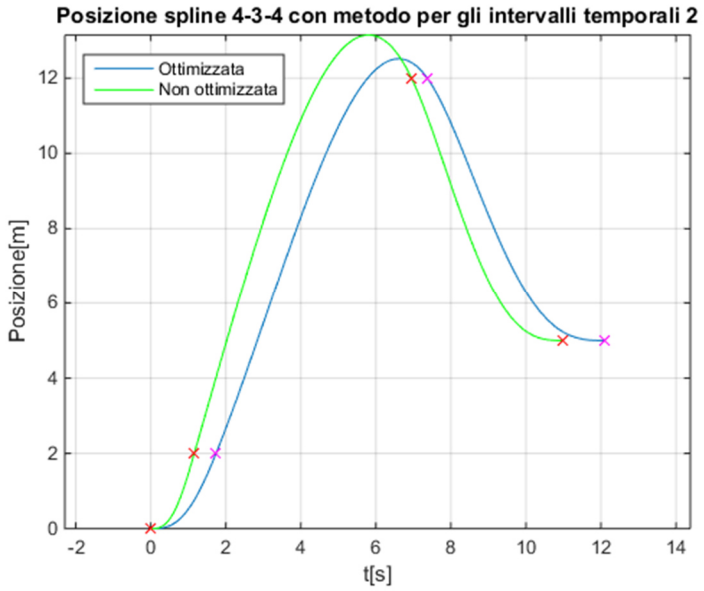
Tipologia Spline / Tipologia Spaziatura temporale	Tempo ottimizzato Spline cubica [s]	Tempo ottimizzato Spline 4-3-4 [s]
1) Equi spaziato	11.2859	14.4242
2) Proporzionale alla distanza fra punti	10.5826	12.0896
3) Proporzionale alla radice della distanza fra i punti	11.3011	12.0896

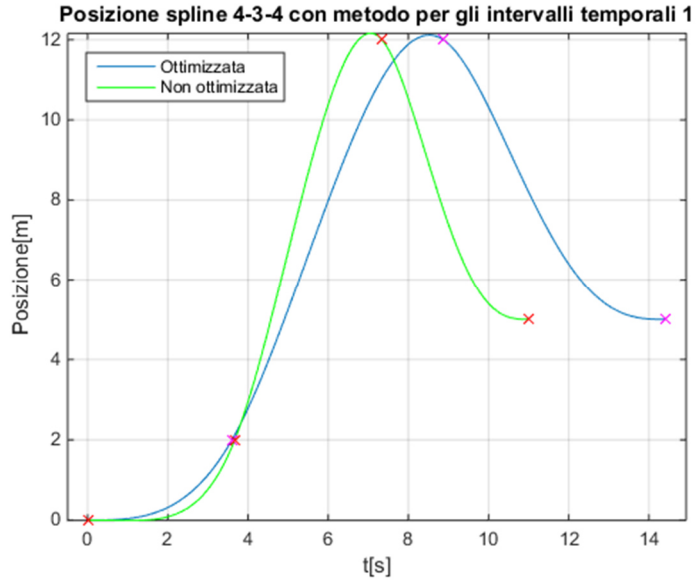
successivamente i grafici di posizione delle spline per ognuno dei casi. Si ricorda che il tempo iniziale è per tutte di 11 secondi. Si vede chiaramente che solo in un'occasione delle 6 proposte il tempo viene ridotto rispetto al valore iniziale. Il secondo metodo si spaziatura degli intervalli sembra essere il migliore quindi nel mantenere ridotte le velocità e le accelerazioni, mentre il primo risulta abbastanza inefficace soprattutto nella 4-3-4. Il fatto che due valori ottimizzati della spline 4-3-4 vengano uguali è un puro caso, e non è frutto di errori.

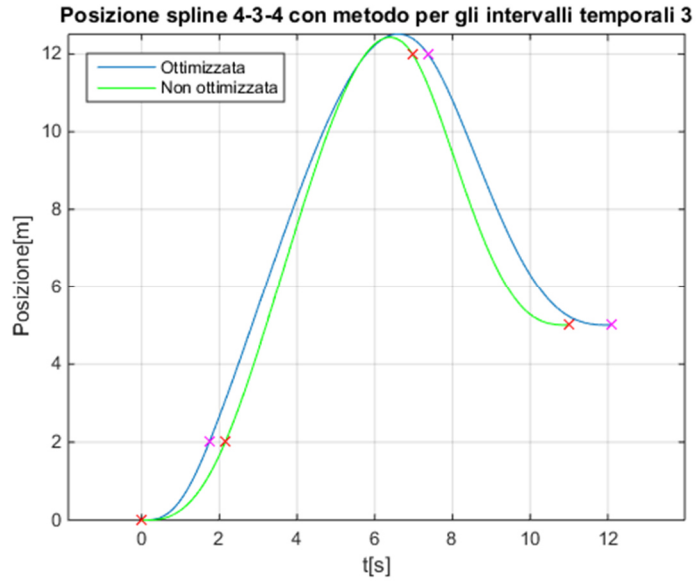

Posizione spline cubica con metodo per gli intervalli temporali 1
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]


Posizione spline cubica con metodo per gli intervalli temporali 2
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]


Posizione spline cubica con metodo per gli intervalli temporali 3
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]


Posizione spline 4-3-4 con metodo per gli intervalli temporali 2
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]


Posizione spline 4-3-4 con metodo per gli intervalli temporali 1
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]


Posizione spline 4-3-4 con metodo per gli intervalli temporali 3
Ottimizzata
Non ottimizzata
Posizione[m]
t[s]

Capitolo 4

B-spline

Finora si è sempre discusso di curve spline con grado di continuità pari a 3, ossia con posizione e velocità continue, accelerazione continua ma con jerk non continuo. Si vuole ora fare un passo avanti e cercare di esprimere una legge di moto che abbia un grado di continuità 4, e quindi presenti un jerk continuo. Può risultare particolarmente utile questo studio in quanto permette di avere un'accelerazione regolare a differenza di quanto visto fin qui. Non si avranno più quindi grafici di accelerazione come quelli visti nei capitoli precedenti, formati da segmenti rettilinei che presentavano punti angolosi ossia di non derivabilità. Con le B-spline ci si può spingere oltre quanto sarà trattato di seguito, in particolare si può salire col grado di continuità desiderato fino ad avere uno snap continuo (grado 5) e oltre; ai fini della nostra trattazione non è di particolare interesse spingersi oltre il grado 4, anche perché l'implementazione MATLAB diventerebbe molto più pesante e laboriosa.

Una B-spline o *basic spline* è una particolare tipologia di spline ottenuta come una combinazione lineare di un certo numero di funzioni base che chiameremo $B_j^p(u)$ in cui p è il grado (nel nostro caso è 4), definite per un *knot vector* o vettore dei nodi $\mathbf{u}=[u_0,\ldots,u_{nknot}]$ che sarà definito più avanti. La forma della B-spline sarà:

$$s(u)=\sum_{j=0}^{m} P_j * B_j^p(u)$$

In cui P_j, j=0,…,m è il vettore dei *control points* o punti di controllo che definiscono la curva e sono calcolati in modo tale da garantire le condizioni di passaggio della curva per i punti desiderati. Tutte le proprietà delle B-spline non vengono riportate in questa trattazione per non renderla troppo pesante e possono essere velocemente lette da testi che trattano tale argomento[1]; quanto necessario per la comprensione e l'implementazione in MATLAB verrà in ogni caso spiegato, anche richiamando determinate proprietà e formule.

[1] L. Biagiotti, C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots,* Springer, 2008

L'unica che si vuole riportare subito e che sarà fondamentale soprattutto per l'implementazione MATLAB è: $m=n_{knot}-p-1$ (4.1). Si partirà scrivendo i dati iniziali del problema e quelli per il calcolo delle funzioni base, che una volta trovate permetteranno di trovare i punti di controllo ed infine la curva desiderata. Fatto tutto ciò si passa al *case study* visto nell'ottimizzazione e si andranno a confrontare i risultati ottenuti con la B-spline con quelli di una spline 4-3-4.

4.1 Inserimento dati, vettore dei nodi e dei termini noti

Come per tutte le curve viste finora si avranno q_k, k=0,…,n punti da interpolare, ognuno ad un determinato t_k istante temporale. Inoltre si dovranno imporre le velocità e le accelerazioni iniziali e finali necessarie per avere un adeguato numero di condizioni che saranno descritte in seguito. Una volta scelto il grado di continuità (i.e. 4) si dovrà trovare una formulazione per definire il *knot vector*. Il modo più semplice di definire gli $n_{knot}+1$ nodi del vettore è quello di porre pari a t_0 i primi p+1 elementi e pari a t_n gli ultimi p+1. Gli altri punti saranno rispettivamente $t_2,\ldots,t_{n-1}$. Si avrebbe un vettore formato da n+2*p+1 elementi che ricordando la relazione (4.1) scritta sopra daranno luogo a (n+1)+p-1 punti di controllo. Questa scelta è ottimale se il grado p è dispari, ma questo non è il nostro caso e quindi si dovrà trovare un modo diverso per definire i nodi. La scelta per gradi p pari ricade sulla seguente formulazione, simile alla precedente ma con un numero di elementi pari a n+2*p+2 e quindi (n+1)+p *control points*. La scelta dei primi e degli ultimi p+1 elementi è uguale al caso precedente, mentre per tutti gli altri punti si avrà: $(t_0+t_1)/2,\ldots,(t_{k-1}+t_k)/2,\ldots,(t_{n-1}+t_n)/2$. Il vettore dei nodi sarà quindi:

$$\mathbf{u}=[t_0, t_0, t_0, t_0, t_0, (t_0+t_1)/2,\ldots,(t_{k-1}+t_k)/2,\ldots,(t_{n-1}+t_n)/2,t_n,t_n,t_n,t_n,t_n).$$

Facendo una rapida analisi dimensionale si osserva che per trovare gli m+1 punti di controllo si userà un sistema lineare che per ora è formato da n+1 equazioni ricavate imponendo il passaggio per i punti di q. Per avere un'unica soluzione dovremo però avere un sistema (m+1)x(m+1) e quindi avere dei vincoli extra; in particolare dovremmo avere ancora un numero p di condizioni. Verranno usati come 4 vincoli le velocità e accelerazioni iniziali e finali della curva: $s^{(1)}(t_0)=v_0$, $s^{(1)}(t_n)=v_n$, $s^{(2)}(t_0)=a_0$, $s^{(2)}(t_n)=a_n$. In alternativa si potevano porre le cosiddette condizioni periodiche o cicliche in cui le derivate i-esime dovevano avere lo stesso valore ad inizio e fine della curva. La nostra però è stata una scelta logica in quanto analoga a quella fatta per ricavare la spline 4-3-4 nel capitolo 2. Una volta

scritto **u** si possono calcolare le funzioni base della B-spline, ma questo si vedrà interamente nel prossimo paragrafo. Fatto ciò si potrà iniziare a comporre il sistema lineare per il calcolo del vettore **p** contenente i P_j. La formulazione sarà del tipo: **A*p=c** dove: **A** è la matrice quadrata delle funzioni base ricavate dalle condizioni descritte sopra, e può essere scritta come:

$$\begin{bmatrix} B_0^p(t_0) & B_1^p(t_0) & & B_m^p(t_0) \\ B_0^{p^{(1)}}(t_0) & B_1^{p^{(1)}}(t_0) & \cdots & B_m^{p^{(1)}}(t_0) \\ B_0^{p^{(2)}}(t_0) & B_1^{p^{(2)}}(t_0) & & B_m^{p^{(2)}}(t_0) \\ B_0^p(t_1) & B_1^p(t_1) & & B_m^p(t_1) \\ \vdots & & \ddots & \vdots \\ B_0^p(t_{n-1}) & B_1^p(t_{n-1}) & & B_m^p(t_{n-1}) \\ B_0^{p^{(2)}}(t_n) & B_1^{p^{(2)}}(t_n) & \cdots & B_m^{p^{(2)}}(t_n) \\ B_0^{p^{(1)}}(t_n) & B_1^{p^{(1)}}(t_n) & & B_m^{p^{(1)}}(t_n) \\ B_0^p(t_n) & B_1^p(t_n) & & B_m^p(t_n) \end{bmatrix} = \mathbf{A}$$

Mentre **c** è il vettore dei termini noti che si ricavano sempre dai vincoli:

$$\mathbf{c}=[q_0, v_0, a_0, q_1, \ldots, q_{n-1}, a_n, v_n, q_n]^T$$

E il vettore dei punti di controllo si trova invertendo la matrice **A** e moltiplicando per **c**, quindi si avrà: $\mathbf{p}=\mathbf{A}^{-1}\mathbf{c}$.

4.2 Funzioni base

Il punto più critico nella creazione di una B-spline è ricavare le funzioni base, e le loro derivate, che sono indispensabili sia per il calcolo dei P_j sia per la creazione della legge di moto vera e propria. La formulazione più semplice per ricavare le funzioni base è di tipo ricorsivo, quindi risulta fondamentale l'aiuto di un software come MATLAB per rendere veloce il calcolo.

4.2.1 Posizione

La funzione ricorsiva per la funzione base di posizione può essere scritta nel seguente modo:

$$B_0^p(u)=\begin{cases} 1, se\ u_j \le u < u_{j+1} \\ 0, altrimenti \end{cases}$$

$$B_j^p(u) = \frac{u - u_j}{u_{j+p}} B_j^{p-1}(u) + \frac{u_{j+p+1} - u}{u_{j+p+1} - u_{j+1}} B_{j+1}^{p-1}(u)$$

Implementata in una *function* di MATLAB che riceve in ingresso: un indicatore del grado della derivata che si vuole ottenere; il grado di continuità; l'indice j; il vettore dei nodi; l'istante temporale in cui è calcolate la funzione. La *function* calcolerà come uscita uno scalare che sarà il nostro $B_j^p(t)$. Si avrà il seguente codice:

```
function[B]=b_fun(d,p,j,u,t)
if d==0%posizione
    if t==u(end)
        if j==length(u)-p-1
            B=1;
            return
        else
            B=0;
            return
        end
    end
    if p==0
        if u(j)<=t && t<u(j+1)
            B=1;
        else
            B=0;
        end
    elseif p>0
        if (u(j+p)-u(j))==0
            B=((u(j+p+1)-t)/(u(j+p+1)-u(j+1))*b_fun(d,p-1,j+1,u,t));
        elseif (u(j+p+1)-u(j+1))==0
            B=((t-u(j))/(u(j+p)-u(j))*b_fun(d,p-1,j,u,t));
        elseif (u(j+p)-u(j))==0 && (u(j+p+1)-u(j+1))==0
            B=0;
        else
            B=((t-u(j))/(u(j+p)-u(j))*b_fun(d,p-1,j,u,t))+...
            ((u(j+p+1)-t)/(u(j+p+1)-u(j+1))*b_fun(d,p-1,j+1,u,t));
        end
    end
end
```

4.2.2 Velocità

Per la derivata prima la funzione ricorsiva da utilizzare per il calcolo analitico è molto simile a quella usata in precedenza, e al suo interno presenta infatti iterazioni che necessitano un richiamo alle funzioni base di posizione. Si avrà quindi:

$$B_j^{p^{(1)}}(u) = \frac{p}{u_{j+p} - u_j} B_j^{p-1}(u) - \frac{p}{u_{j+p+1} - u_{j+1}} B_{j+1}^{p-1}(u)$$

Che codificato in MATLAB sempre all'interno della *function* "b_fun":

```
if d==1 %derivata prima
    if (u(j+p)-u(j))==0
```

```
        B=-(p/(u(j+p+1)-u(j+1))*b_fun(0,p-1,j+1,u,t));
    elseif (u(j+p+1)-u(j+1))==0
         B=(p/(u(j+p)-u(j))*b_fun(0,p,j,u,t));
    else
        B=(p/(u(j+p)-u(j))*b_fun(0,p-1,j,u,t))-...
        (p/(u(j+p+1)-u(j+1))*b_fun(0,p,j+1,u,t));
    end
end
```

4.2.3 Accelerazione

Per la derivata seconda la formulazione analitica e quindi il codice cominciano a diventare parecchio pesanti ed esosi di tempo e risorse; ciò è dato dal fatto che per il calcolo delle funzione base di una generica derivata k-esima esiste un algoritmo che sempre in modo iterativo e con l'utilizzo di fattori ci permette di ricavare la $B_j^{p^{(k)}}(u)$ desiderata. Poiché ai fini pratici di questa trattazione serve la formulazione finale delle funzioni base viene omesso l'algoritmo e i suoi coefficienti che possono comunque essere trovati nei testi a cui si fa riferimento nella bibliografia. Risulterà dopo alcuni passaggi che:

$$B_j^{p^{(2)}}(u) = \frac{p!}{(p-k)!}\Big[\frac{1}{(u_{j+p-1}-u_j)(u_{j+p}-u_j)}B_j^{p-k}(u) - \frac{u_{j+p+1}-u_{j+1}+(u_{j+p}-u_j}{(u_{j+p}-u_{j+1})(u_{j+p+1}-u_{j+1})(u_{j+p}-u_j)}B_{j+1}^{p-k}(u) + \frac{1}{(u_{j+p+1}-u_{j+2})(u_{j+p+1}-u_{j+1})}B_{j+2}^{p-k}(u)\Big]$$

Che implementato in MATLAB dà:

```
if d==2%derivata seconda
    if (u(j+p-1)-u(j))*(u(j+p)-u(j))==0 && (u(j+p+1)-u(j+1))*(u(j+p)-
u(j+1))*(u(j+p)-u(j))==0
        B=(12/(u(j+p+1)-u(j+2))/(u(j+p+1)-u(j+1)))*b_fun(0,p-d,j+2,u,t);
    elseif (u(j+p-1)-u(j))*(u(j+p)-u(j))==0 && (u(j+p+1)-
u(j+2))*(u(j+p+1)-u(j+1))==0
        B=-(12*(u(j+p)-u(j)+u(j+p+1)-u(j+1))/(u(j+p+1)-u(j+1))/(u(j+p)-
u(j+1))/(u(j+p)-u(j)))*b_fun(0,p-d,j+1,u,t);
    elseif (u(j+p+1)-u(j+1))*(u(j+p)-u(j+1))*(u(j+p)-u(j))==0 &&
(u(j+p+1)-u(j+2))*(u(j+p+1)-u(j+1))==0
        B=(12/(u(j+p-1)-u(j))/(u(j+p)-u(j)))*b_fun(0,p,j,u,t);
    elseif (u(j+p-1)-u(j))*(u(j+p)-u(j))==0
        B=-(12*(u(j+p)-u(j)+u(j+p+1)-u(j+1))/(u(j+p+1)-u(j+1))/(u(j+p)-
u(j+1))/(u(j+p)-u(j)))*b_fun(0,p-d,j+1,u,t)...
        +(12/(u(j+p+1)-u(j+2))/(u(j+p+1)-u(j+1)))*b_fun(0,p-d,j+2,u,t);
    elseif (u(j+p+1)-u(j+1))*(u(j+p)-u(j+1))*(u(j+p)-u(j))==0
        B=(12/(u(j+p-1)-u(j))/(u(j+p)-u(j)))*b_fun(0,p-d,j,u,t)...
            +(12/(u(j+p+1)-u(j+2))/(u(j+p+1)-u(j+1)))*b_fun(0,p-
d,j+2,u,t);
```

```
    elseif (u(j+p+1)-u(j+2))*(u(j+p+1)-u(j+1))==0
        B=(12/(u(j+p-1)-u(j))/(u(j+p)-u(j)))*b_fun(0,p-d,j,u,t)...
        -(12*(u(j+p)-u(j)+u(j+p+1)-u(j+1))/(u(j+p+1)-u(j+1))/(u(j+p)-
u(j+1))/(u(j+p)-u(j)))*b_fun(0,p,j+1,u,t);
    else
        B=(12/(u(j+p-1)-u(j))/(u(j+p)-u(j)))*b_fun(0,p-d,j,u,t)...
        -(12*(u(j+p)-u(j)+u(j+p+1)-u(j+1))/(u(j+p+1)-u(j+1))/(u(j+p)-
u(j+1))/(u(j+p)-u(j)))*b_fun(0,p-d,j+1,u,t)...
        +(12/(u(j+p+1)-u(j+2))/(u(j+p+1)-u(j+1)))*b_fun(0,p,j+2,u,t);
    end
end
```

Come si può intuire, aggiungere il calcolo del jerk e dello snap avrebbe necessitato di tempi di calcolo molto lunghi, e i risultati sarebbero stati di scarso interesse in quanto nelle applicazioni reali a cui si fa riferimento in questa trattazione non vengono usate derivate superiori alla seconda.

4.3 Costruzione legge di moto

Ora che si ha la *function* per ricavare le funzioni base di posizione, velocità e accelerazione non ci resta che implementare in MATLAB quando detto al paragrafo §4.1 e successivamente calcolare la legge di moto completa ricordando la seguente formulazione per le B-spline:

$$s^{(k)}(u) = \sum_{j=0}^{m} p_j * B_j^{p(k)}(u)$$

Si è scelto di creare un'apposita funzione chiamata "b_fun" che similmente a quanto fatto per la spline 4-3-4 riceve in ingresso i punti da interpolare, gli istanti temporali e le velocità e accelerazioni iniziali e finali e darà in uscita la legge di moto per posizione, velocità e accelerazione. Questo permetterà poi di far il confronto fra le spline avendo funzioni che operano con gli stessi parametri in ingresso e uscita. Si sottolinea inoltre il fatto che anche per le B-spline la funzione ha carattere totalmente generale e quindi può essere applicata a un numero qualsiasi di punti. Di seguito il codice MATLAB della *function* che crea la B-spline:

```
%% B-SPLINE
function[pos,vel,acc]=B_spline(q,t,vi,vf,ai,af)
n=length(q);
m=n+4;
dt=0.01;
tt=0:dt:t(end);
for ind=1:length(t)-1
    T(ind)=t(ind+1)-t(ind);
end
%% Esecuzione
```

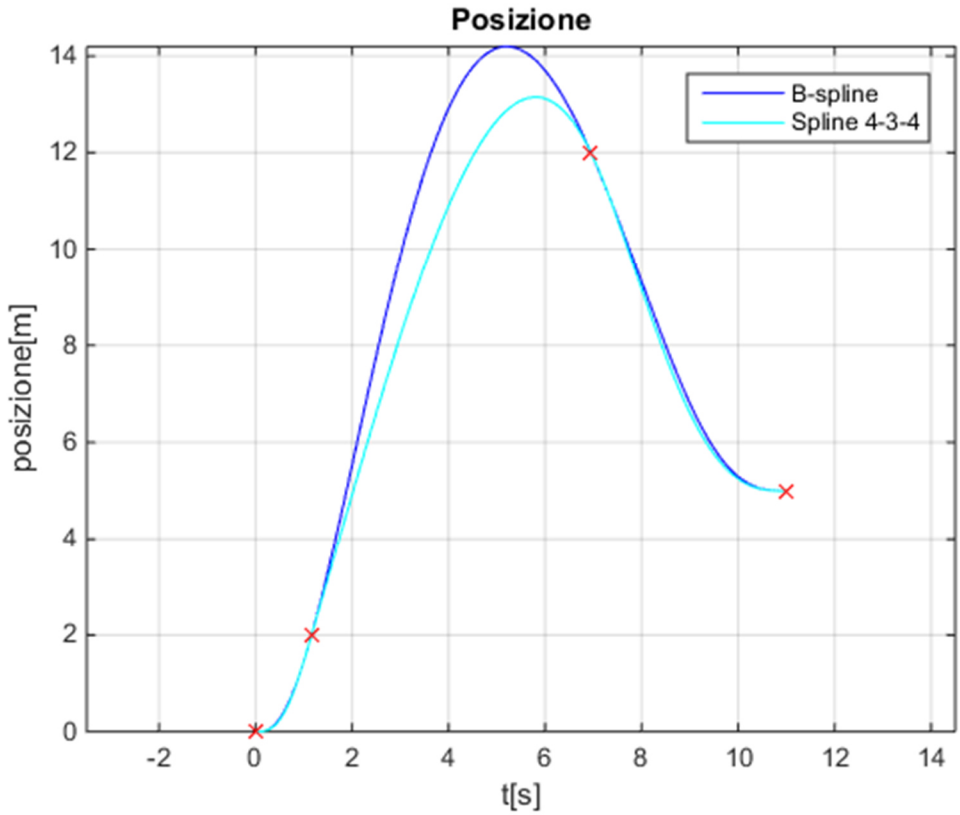
```
p=4;%grado di continuità
%creo il knot vector che sarà lungo n+2p+1 se p è pari
u=zeros(1,n+2*p+1);
u(1,1:p+1)=t(1);
u(1,n+p+1:n+2*p+1)=t(n);
e=2;
for k=p+2:n+p
    u(k)=(t(e-1)+t(e))/2;
    e=e+1;
end
% vettore delle variabili note
c=zeros(m,1);
c(1)=q(1);
c(2)=vi;
c(3)=ai;
c(end)=q(end);
c(end-1)=vf;
c(end-2)=af;
e=2;
for k=p:n+1
    c(k)=q(e);
    e=e+1;
end
% Funzione base
for k=1:n
    for j=1:m
        B(k,j)=b_fun(0,p,j,u,t(k));%posizione
    end
end
for k=1:n
    for j=1:m
        dB(k,j)=b_fun(1,4,j,u,t(k));%velocità
    end
end
for k=1:n
    for j=1:m
        ddB(k,j)=b_fun(2,4,j,u,t(k));%accelerazione
    end
end
A(1,:)=B(1,:);
A(m,:)=B(n,:);
A(2,:)=dB(1,:);
A(m-1,:)=dB(n,:);
A(3,:)=ddB(1,:);
A(m-2,:)=ddB(n,:);
A(4:m-3,:)=B(2:n-1,:);

P=pinv(A)*c;%Control Points
%% Costruzione legge di moto(posizione,velocità,accelerazione)
for j=1:m
    for k=1:length(tt)
        f_base(j,k)=b_fun(0,p,j,u,tt(k));%funzione base
    end
end
for j=1:m
    pos(j,:)=P(j).*f_base(j,:);
end
pos=sum(pos);
for j=1:m
    for k=1:length(tt)
        v_base(j,k)=P(j).*b_fun(1,p,j,u,tt(k));%velocità
```

```
    end
end
vel=sum(v_base);
for j=1:m
    for k=1:length(tt)
        a_base(j,k)=P(j).*b_fun(2,p,j,u,tt(k));%accelerazione
    end
end
acc=sum(a_base);
return
```

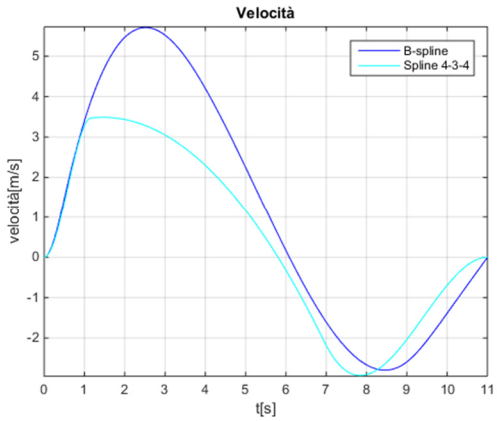
Una volta creata questa funzione basterà richiamarla all'interno di un codice simile al Main della spline 4-3-4 inserendo i dati iniziali e successivamente graficando i risultati trovati. Per motivi di ridondanza omettiamo questi semplici passaggi.

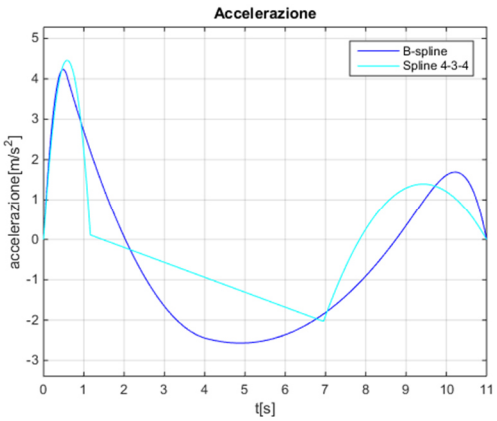
4.4 Confronto con spline 4-3-4

Di notevole interesse a questo punto sarà il fatto di confrontare le due tipologie principali di spline trattate finora, ossia la spline 4-3-4 e la B-spline. In particolare si andrà a vedere se e cosa si dovrà sacrificare per avere un grado maggiore di continuità, e quindi confrontando i grafici dell'accelerazione che saranno molto differenti. In più si vuole porre l'attenzione al tempo che impiega la macchina a calcolare entrambi i metodi; ciò verrà fatto con il "tic-toc" di MATLAB. Si noterà una palese differenza sotto questo aspetto fra le due tipologie. Per semplicità e per una maggiore comprensione verranno plottate entrambe le spline nella stessa figura. Di seguito i grafici e i commenti. Si usano i dati dell'esempio del capitolo 3.



Il grafico che mostra la legge di moto di posizione è il più importante in quanto è quello che fin dal primo capitolo si è cercato di rendere il più dolce possibile; proprio partendo da questo presupposto si può fare una prima considerazione. Si nota infatti che nel tratto centrale la B-spline assume valori di posizione maggiori che nella 4-3-4 e questo può essere un problema di ingombro non indifferente. Questo inconveniente viene riscontrato in maniera così evidente solo in questo caso particolare; infatti dalle molteplici prove effettuate con dati diversi da quelli proposti sopra la differenza di posizione fra le due spline è inesistente o ininfluente, sempre comunque con una maggiore oscillazione da parte della B-spline. In ogni caso se si vuole dare una spiegazione a questa cosa, si può ipotizzare con buona probabilità che sia causata dal fatto che le B-spline per poter avere una continuità dell'accelerazione deve necessariamente raggiungere picchi più elevati di accelerazione per potersi raccordare senza discontinuità e questo causa un effetto a catena nella velocità e quindi nella posizione. Si capirà meglio questo discorso dai seguenti grafici.





Come si può vedere l'obiettivo di avere anche l'accelerazione continua è stato raggiunto; ciò però ha portato alcune differenze in confronto agli altri tipi di spline. Come ipotizzato in precedenza l'accelerazione arriva ad assumere valori massimi in valore assoluto molto superiori alla 4-3-4 proprio perché forzata a mantenere la continuità nel grafico; questo porta ad avere quindi un picco di velocità.

Come ultimo campo per confrontare le due tipologie di spline andiamo a vedere i tempi impiegati dal calcolatore ad eseguire il codice MATLAB di creazione della legge di moto. Per farlo si utilizzerà il "tic-toc" presente nel software. I tempi saranno calcolati nel main del programma delle B-spline come segue:

```
%% B_SPLINE
tic;
[pos,vel,acc]=B_spline(q,t,vi,vf,ai,af);
time_B=toc;
%% Confronto con Spline 4-3-4
tic;
[s_pos,s_vel,s_acc]=spline434f(q,T,vi,vf,ai,af);
time_434=toc;

disp(['La B-spline impiega ',num2str(time_B),'s per il calcolo']);
disp(['La spline 434 impiega ',num2str(time_434),'s per il calcolo']);
```

Che darà a video i seguenti risultati:

La B-spline impiega 17.9675s per il calcolo

La spline 434 impiega 0.88489s per il calcolo

Il divario fra i due tempi è abissale, se si pensa che si sta utilizzando un pc per il calcolo. La B-spline impegna venti volte il tempo di calcolo di una 4-3-4, e si pensi che sono stati interpolati solamente 4 punti. Per portare un ulteriore esempio si riporta che per interpolare 7 punti sono stati necessari 1.2768 secondi per avere una spline 4-3-4, e ben 38.7283 secondi per la B-spline, portando il rapporto di tempistiche fra le due a 1:30. La causa di questo elevatissimo tempo per creare una B-spline è da imputare al fatto che il metodo usato per ricavarla è iterativo, e quindi molto esoso di risorse di calcolo, mentre per la spline 4-3-4 si tratta solamente di risolvere un sistema lineare.

Si dovrà sempre e comunque valutare caso per caso quale delle due spline sia migliore, ossia se preferire una continuità dell'accelerazione sapendo che i tempi di calcolo saranno significativi oppure una maggior regolarità della traiettoria, con un codice più snello e veloce da calcolare. Poiché la seconda delle due opzioni è quella che ci si era prefissata all'inizio della trattazione, si rimane fermi nel concludere che una spline 4-3-4 è un ottimo modo per poter esprimere una traiettoria che sia dolce e regolare e che permetta di avere in tempi rapidi un'idea di quello che sarà il movimento della macchina a cui è applicato.

Conclusioni

L'obiettivo che ci si era posti ad inizio trattazione era quello di analizzare e confrontare diversi modi per interpolare un numero generico di punti tale che la traiettoria così creata avesse un andamento privo di eccessive oscillazioni e che fosse possibile implementarlo in MATLAB. Nei vari capitoli si è visto che l'utilizzo della spline è uno dei migliori metodi per fare ciò, per questo si è scelto di studiare varie tipologie di spline, confrontandole fra loro sotto il punto di vista dei risultati ottenuti, e anche dando uno sguardo al codice implementato. Si può concludere che la spline 4-3-4 è la soluzione che rappresenta un ottimo compromesso tra tutte le specifiche. Infatti la traiettoria ottenuta è molto regolare, senza presentare picchi eccessivi di velocità e accelerazione. Anche nel caso ciò accadesse si è visto che è possibile far rientrare tali grandezze all'interno di limiti prefissati andando anche ad ottimizzare il tempo di percorrenza. Un altro punto a favore della spline 4-3-4 è stato il fatto che si è riusciti a creare un codice MATLAB in grado di contemplare un problema totalmente generico, con inserimento dei dati lasciato completamente libero all'utilizzatore, e con tempi di calcolo rapidi che consentono un facile ricalcolo on-line. Lo stesso si è riusciti a fare anche con la B-spline, che riusciva anche ad aumentare il grado di continuità della curva. Si è visto però che messi a confronto i due codici di calcolo, quello della B-spline richiedeva tempi di calcolo molto lunghi se confrontati con quelli della spline 4-3-4. In definitiva sia per snellezza e rapidità del codice, sia per la sua generalità e per gli ottimi risultati che si ottengono viene suggerita la spline 4-3-4 come metodo per creare traiettorie per il movimento, a meno di non avere applicazioni particolari in cui è necessario rendere continuo il jerk, date le elevate deformabilità del componente da movimentare.

Bibliografia

L. Biagiotti, C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots,* Springer, 2008

C. Melchiorri, *Traiettorie per azionamenti elettrici*, Ed. Esculapio Bologna, 2003

C. Melchiorri, Slide del corso di Robotica Industriale, Università di Bologna

D. Richiedei, Appunti lezione corso di Controllo dei Sistemi Meccanici, Università di Padova

Siti web:

https://it.mathworks.com/help/matlab/

Software utilizzati:

MATLAB R2014b

Ringraziamenti

Desidero ringraziare quanti mi hanno aiutato nella stesura della tesi, sia in modo tangibile che moralmente.

Ringrazio innanzitutto il relatore della tesi, il Professore Dario Richiedei, per avermi fornito i documenti utili alla stesura del testo e per la disponibilità nello spiegarmi gli argomenti trattati negli stessi; lo ringrazio inoltre per aver saputo suscitare il mio interesse nel software MATLAB durante il corso di Fondamenti di Meccanica e Laboratorio.

Ringrazio poi i parenti e gli amici per il supporto morale durante tutto il periodo universitario.