Corso di Laurea Magistrale in

Ingegneria Biomedica

Esame di

Robotica Medica

Lezione del 26/10/23

Inversione della cinematica differenziale III

Esercitazione 5

Anno Accademico 2022/2023

Traccia

Estendere i risultati precedenti per una struttura a 3D, ad esempio uno Jaco^2 a 7 DOFs
Una possibile implementazione è disponibile nel file `esercizio05.zip`

Figura 1: Traccia

Nella scorsa esercitazione (la quarta) è stato aggiunto il quaternione unitario come rappresentazione dell'orientamento. Adesso, si vuole implementare sempre un algoritmo per l'inversione della cinematica differenziale ma ci si vuole spostare su una struttura a 7 gradi di libertà (DOFs), quindi ridondante. Il manipolatore considerato è infatti un *Kinova Jaco 2* a 7 gradi di libertà. Non ci si trova più ad affrontare un problema di tipo planare, ma tridimensionale, ed in più anche ridondante.

Background teorico

Braccio	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0.2755	θ_1
2	0	$\pi/2$	0	θ_2
3	0	$\pi/2$	-0.41	θ_3
4	0	$\pi/2$	-0.0098	θ_4
5	0	$\pi/2$	-0.3072	θ_5
6	0	$\pi/2$	0	θ_6
7	0	0	0.25	θ_7

Figura 2: Kinova Jaco 2

Il Kinova Jaco2 è un manipolatore progettato per applicazioni di servizio e assistenza. Dal punto di vista delle caratteristiche del manipolatore, è noto per avere 7 gradi di libertà, il che significa che può muoversi in sette direzioni indipendenti. Ogni giunto rappresenta un DoF e contribuisce alla flessibilità del manipolatore. La modifica della tavola Denavit-Hartenberg (DH) indica un cambiamento nella convenzione di rappresentazione geometrica dei giunti e dei link del robot. In particolare, le variabili ai nella tavola DH non rappresentano più i bracci del manipolatore ma assumono un significato diverso. Questa modifica è probabilmente correlata alla necessità di adattare la tavola DH alle specifiche cinematiche del Kinova Jaco2, considerando i sette bracci del manipolatore. In termini di movimenti, il robot può eseguire una varietà di task grazie ai suoi 7 gradi di libertà. Questo gli consente di raggiungere posizioni e orientamenti specifici nello spazio tridimensionale. I movimenti possono essere combinati in modo coordinato per eseguire operazioni complesse come la presa e il posizionamento di oggetti, operazioni di montaggio e molte altre attività.

Progettazione dello script

Ad una maggiore complessità del caso analizzato viene fatta corrispondere una necessaria ottimizzazione del codice Matlab di implementazione dell'algoritmo di inversione cinematica. In questa esercitazione, infatti, vengono riservati l'inizializzazione delle variabili e i vari plottaggi al file '*main*' mentre il processo di inversione cinematica viene affidato completamente alla funzione '*inverseKinematics*'.
La stessa funzione '*inverseKinematics*' presenta una prima parte di inizializzazione delle variabili, una seconda di implementazione vera e propria dell'algoritmo di inversione cinematica ed un'ultima parte in cui sono presenti funzioni ausiliarie utile alla parte centrale di inversione.

```
function [q, dq, positionErrors, orientationErrors, spacePath, DHS] = inverseKinematics( ...
    setPoint, ...
    desiredOrientation, ...
    startConfiguration, ...
    simulationTime, ...
    positionGain, ...
    orientationGain, ...
    algorithmType)
```

Figura 3: Funzione *inverseKinematics*

Questa funzione calcola i valori delle variabili di giunzione che devono essere assegnati per ottenere il set point desiderato con un determinato orientamento. Questa funzione è ottimizzata per il manipolatore Kinova Jaco2 7-DOF. Di seguito vengono raffigurati le variabili di input da darle in pasto insieme agli output che restituisce:

```
% INPUT
%
%    - setPoint : 3x1 array in format [x,y,z]
%
%    - desiredOrientation : 3x1 array with ZYZ convention
%
%    - startConfiguration : 7x1 array (Theta angles of Jaco2 spherical
%                           joint)
%
%    - simulationTime : Scalar value in seconds, default 60 s.
%
%    - positionGain : Scalar value for position gain matrix, default 5.
%
%    - orientationGain : Scalar value for orientation gain matrix,
%                        default 5
%
%    - algorithmType : String between "pinv" or "transp", default
%                      "pinv".
```

Variabili di input della *inverseKinematics*

```
% OUTPUT
%
%    - q : Matrix containing the values of the joint variables for each
%          iteration.
%
%    - dq : Matrix containing the values of angular velocity for each
%           iteration.
%
%    - positionErrors : Matrix containing the position error. Each
%                       column of the matrix is formed
%                       as [ex, ey, ez, t].
%
%    - orientationErrors : Matrix containing the orientation error.
%                          Each column of the matrix is formed as
%                          [e_phi, e_theta, e_psi, t].
%
%    - spacePath : Matrix containing the trajectory on the operative
%                  space. Each column of the matrix is formed
%                  as [x, y, z].
%
%    - DHS : 7x4x2 tensor with start and ending DH table.
```

Variabili di output della *inverseKinematics*

Figura 4

Implementazione

inverseKinematics

Il codice della *inverseKinematics* inizia quindi a preparare l'ambiente e le variabili necessarie per la funzione di cinematica inversa, garantendo che le variabili opzionali siano gestite correttamente e che tutte le strutture dati siano inizializzate in modo appropriato per l'esecuzione della cinematica inversa nel seguito del codice.

```
% Checking optional variables
if ~exist('simulationTime', 'var')
    simulationTime = 60;
end

if ~exist('positionGain', 'var')
    positionGain = 5;
end

if ~exist('orientationGain', 'var')
    orientationGain = 5;
end

if ~exist('algorithmType', 'var') || (algorithmType ~= "pinv" && ...
        algorithmType ~= "transp")
    algorithmType = "pinv";
end

% Denavit-Hartenberg table
a = zeros(7,1);
alfa = zeros(7,1);
alfa(1:6) = pi/2;
d = [0.2755 0 -0.41 -0.0098 -0.3072 0 0.25]';

startConfiguration = checkDimension(startConfiguration);

DH = [a alfa d startConfiguration];

% Time parameters [s]
samplingTime = 0.01;
t = 0 : samplingTime : simulationTime;
nPoints = length(t);

% Gain matrices
KP = diag(positionGain*ones(1,3));
KO = diag(orientationGain*ones(1,3));
```

```
% Output variables
q = zeros(7, nPoints);
dq = zeros(7, nPoints);
orientationErrors = zeros(4, nPoints);
orientationErrors(4,:) = t;
positionErrors = zeros(4, nPoints);
positionErrors(4,:) = t;
spacePath = zeros(3, nPoints);
DHS = zeros(7, 4, 2);

% Set point parameters
pD = setPoint;
RD = zyz2rot( ...
    desiredOrientation(1), ...
    desiredOrientation(2), ...
    desiredOrientation(3));
QD = rot2quat(RD);

DHS(:,:,1) = DH;
```

Figura 5: Inizializzazione delle variabili nella funzione *inverseKinematics*

Verifica delle variabili opzionali:

- `simulationTime`: Controlla se la variabile `simulationTime` esiste; se non esiste, le assegna un valore predefinito di 60 secondi.
- `positionGain`: Controlla se la variabile `positionGain` esiste; se non esiste, le assegna un valore predefinito di 5.
- `orientationGain`: Controlla se la variabile `orientationGain` esiste; se non esiste, le assegna un valore predefinito di 5.
- `algorithmType`: Controlla se la variabile `algorithmType` esiste o se è diversa da "pinv" e "transp"; in caso affermativo, le assegna "pinv" come valore predefinito.

Definizione della tavola Denavit-Hartenberg (DH): Inizializza le variabili `a`, `alfa`, e `d` utilizzate per la tavola DH;
Imposta valori specifici per `alfa` nel range da 1 a 6;
Definisce la colonna `d` con valori specifici;
Esegue un controllo delle dimensioni per `startConfiguration` utilizzando la funzione `checkDimension`;
Crea la tavola DH (`DH`) concatenando le variabili `a`, `alfa`, `d`, e `startConfiguration`.

Parametri temporali: Imposta il `samplingTime` a 0.01 secondi;
Genera un vettore `t` da 0 a `simulationTime` con incrementi di `samplingTime`;
Calcola il numero di punti temporali (`nPoints`);

Matrici di guadagno: Crea le matrici di guadagno `KP` e `KO` utilizzando valori diagonali basati su `positionGain` e `orientationGain`, rispettivamente.

Variabili di output inizializzate: Inizializza vettori e matrici (`q`, `dq`, `orientationErrors`, `positionErrors` `spacePath`, `DHS`) con dimensioni appropriate.

Parametri del set point: Definisce `pD` come il punto fisso;
Calcola la matrice di rotazione `RD` e il quaternion corrispondente `QD` a partire dalla `desiredOrientation`;
parte della matrice `DHS` con i valori della tavola DH;

```
for i = 1 : nPoints

    T = DirectKinematics(DH);
    T = T(:,:,end);

    pE = T(1:3, 4);
    spacePath(:,i) = pE;
    RE = T(1:3, 1:3);
    QE = rot2quat(RE);

    ePi = pD - pE;
    eOi = quatError(QE, QD);
    positionErrors(1:3,i) = ePi;
    orientationErrors(1:3,i) = eOi;

    q(:,i) = DH(:,4);

    amplifiedErrors = [KP*ePi; KO*eOi];

    J = Jacobian(DH);

    if algorithmType == "pinv"

        pinvJ = pinv_damped(J);
        dqi = pinvJ*amplifiedErrors;

    else

        transpJ = J';
        dqi = transpJ*amplifiedErrors;

    end

    dq(:,i) = dqi;

    qNext = q(:,i) + samplingTime*dqi;
    DH(:,4) = qNext;

end

DHS(:,:,end) = DH;
```

Figura 6: Algoritmo di inversione cinematica implementato dalla funzione *inverseKinematics*

L'algoritmo di inversione della cinematica diretta effettua le stesse operazioni svolte nelle esercitazioni precedenti ma si trova a lavorare con variabili più onerose. In particolare la posizione desiderata pE prende tutte e 3 le sue componenti, così come anche il quaternione QE per l'orientamento. Inoltre anche per lo Jacobiano è necessario prendere tutte le sue componenti.

Iterazione attraverso i punti temporali:
Utilizza un ciclo `for` da 1 a `nPoints`.

Calcolo della cinematica diretta:
Utilizza la funzione `DirectKinematics` per ottenere la trasformazione `T`.; Estrae la matrice `T` corrispondente all'ultimo elemento della terza dimensione.

Calcolo degli errori di posizione e orientamento:
Estrae la posizione `pE` e la matrice di rotazione `RE` da `T`;
Calcola il quaternione corrispondente `QE` da `RE`;
Calcola gli errori di posizione `ePi` e orientamento `eOi`;
Aggiorna i vettori di errori di posizione e orientamento;

Calcolo degli amplificatori di errore:
Calcola gli errori amplificati (`amplifiedErrors`) moltiplicando gli errori per le matrici di guadagno `KP` e `KO`.

Calcolo del Jacobiano:
Utilizza la funzione `Jacobian` per ottenere il Jacobiano `J`.

Scelta dell'algoritmo di calcolo:
Se `algorithmType` è "pinv", utilizza la pseudoinversa ammorbidita (`pinv_damped`) del Jacobiano per calcolare `dqi`; Altrimenti, utilizza la trasposta del Jacobiano per calcolare `dqi`.

Aggiornamento delle variabili di stato:
Calcola la prossima configurazione delle giunture (`qNext`) e aggiorna la tavola DH;

Alla fine dello script della funzione MATLAB *inverseKinematics* altre funzioni sono state definite all'interno dello stesso script per facilitare la comprensione e l'utilizzo del codice. Ogni funzione svolge un ruolo specifico nella manipolazione e nell'analisi di dati legati alla cinematica di un robot manipolatore.

- **checkDimension(vector):** Questa funzione è progettata per verificare e, se necessario, correggere le dimensioni di un vettore. Questo è cruciale per garantire che i dati in ingresso siano nel formato corretto per le operazioni successive nel codice.
- **zyz2rot(phi, theta, psi):** Trasforma gli angoli di Roll-Pitch-Yaw in una matrice di rotazione 3x3, un passo fondamentale nella descrizione dell'orientamento di un oggetto nello spazio tridimensionale.
- **rot2quat(R):** Converte una matrice di rotazione in un quaternione, una rappresentazione alternativa dell'orientamento utile in molte applicazioni di robotica e grafica computerizzata.
- **quatError(QE, QD):** Calcola l'errore tra due quaternioni, particolarmente utile per il controllo di robot che devono seguire una traiettoria specifica.
- **Homogeneous(DH_row):** Calcola la matrice di trasformazione omogenea 4x4 per un link nella tavola Denavit-Hartenberg. Questo è essenziale per la cinematica diretta di un robot.
- **DirectKinematics(DH):** Calcola la cinematica diretta per una catena cinematica basata sulla tavola Denavit-Hartenberg. Fornisce una serie di matrici di trasformazione che descrivono la posizione e l'orientamento di ciascun link rispetto alla base.
- **Jacobian(DH, jointConfiguration):** Calcola il Jacobiano geometrico per una catena cinematica. Questo è fondamentale per comprendere la relazione tra le velocità delle articolazioni e le velocità del manipolatore.
- **pinv_damped(J):** Calcola la pseudo-inversa di una matrice Jacobiana utilizzando una tecnica di damping. Questo è spesso utilizzato nella risoluzione numerica della cinematica inversa.
- **cosRialzato(sigma, lambda, threshold):** Applica una regolarizzazione cosenoidale a un valore, contribuendo a mantenere la stabilità numerica durante alcune operazioni.

main

```
%% Input parameters

clear
close all
clc

%%%%% INPUT PARAMETERS %%%%%
pD = [0.5 0.3 0.4]';                    % [X Y Z]
oD = [pi/6 pi/4 pi/4]';                 % [phi theta psi] - ZYZ
start = [0 0 0 pi/4 0 0 0]';    % Initial set-up (Joint variables)
simulationTime = 5;                     % Simulation time [s]
positionGain = 3;                       % Position gain for KP matrix
orientationGain = 3;                    % Orientation gain for KO matrix
algorithmType = "transp";                 % Set "transp" for transpose algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[q, dq, pErr, oErr, path, DHS] = inverseKinematics( ...
    pD, ...
    oD, ...
    start, ...
    simulationTime, ...
    positionGain, ...
    orientationGain, ...
    algorithmType);

%% Errors plot

figure

subplot 211
plot(pErr(4,:), pErr(1,:), LineWidth=2);
hold on
plot(pErr(4,:), pErr(2,:), LineWidth=2);
plot(pErr(4,:), pErr(3,:), LineWidth=2);
legend("X error", "Y error", "Z error")
grid on
```

inizializzazione

chiamata della funzione *inverseKinematics*

Plot

Figura 7: Struttura del file *main*

Nel file *main* nella parte iniziale vengono inizializzate le variabili per la funzione *inverseKinematics* e viene richiamata tale funzione. Successivamente vengono eseguiti i vari plot: in questa esercitazione è possibile plottare tutte e tre le componenti (x, y, z) dell'errore di posizione, così come anche per quello di orientamento (ϕ, θ, ψ).

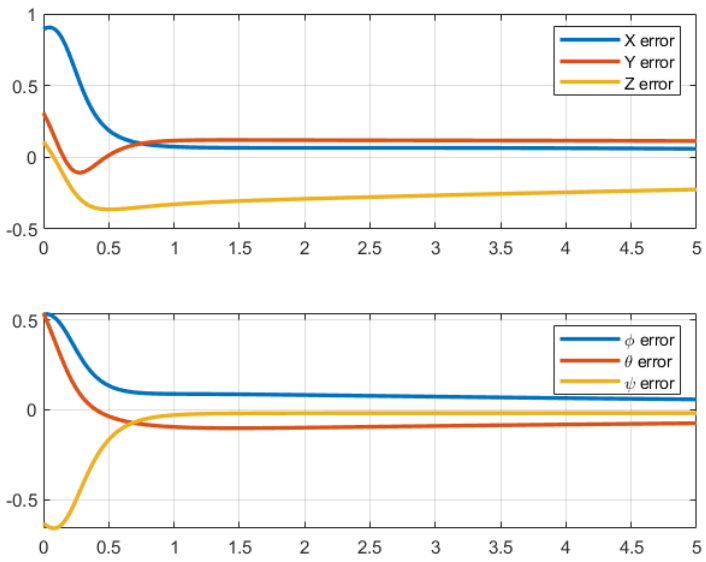

Figura 8: Plot degli errori

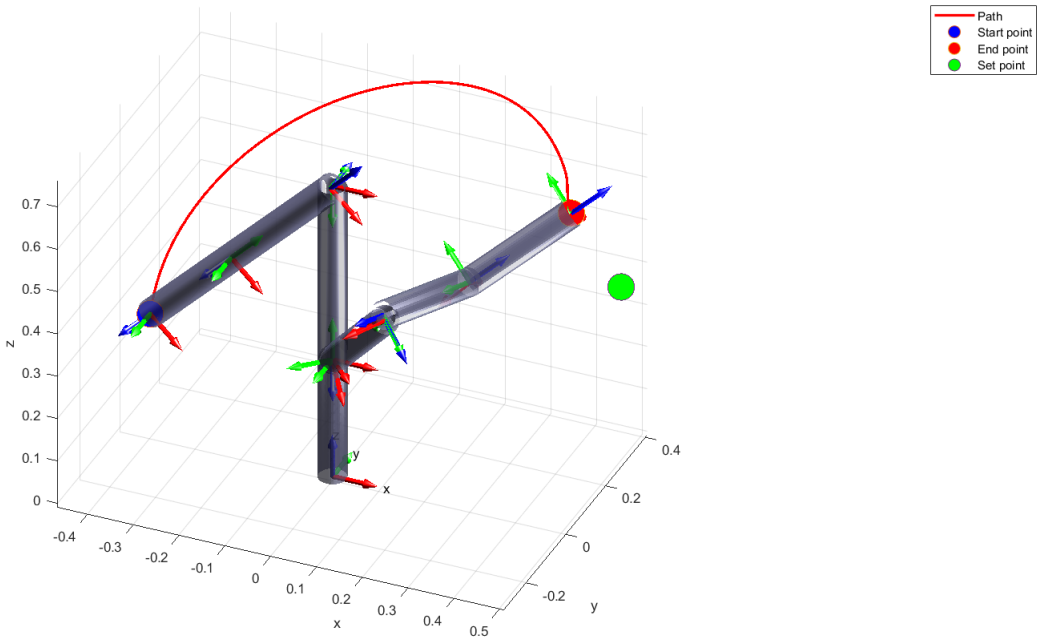

Figura 9: Visualizzazione tramite la funzione *DrawRobot* delle configurazioni iniziali e finali e della traiettoria del manipolatore a 7 bracci imponendo come posizione desiderata pD = [0.5 0.3 0.4]', come orientamento desiderato oD = [pi/6 pi/4 pi/4]' e come configurazione iniziale ai giunti start = [0 0 0 pi/4 0 0 0]'.