UNIVERSITÀ
DELLA CALABRIA

Relazione Robotica

Vaccaro Francesca 239641
Anno accademico 2024/2025

Progetto Laboratorio di automatica

Vaccaro Francesca 239641

L.T. Ingegneria Informatica 24/25

Contents

1 Introduzione

Il seguente progetto si pone l'obiettivo di fare compiere ad un robot fisso, tre differenti percorsi all'interno di uno spazio di lavoro, e questo verrà fatto con l'applicazione della cinematica del robot e di quelle che sono le leggi orarie che contraddistinguono il movimento.

1.1 Composizione del robot

Prima di capire come si muove nei tre scenari richiesti è importante conoscere il tipo di robot con cui si ha a che fare; cominciamo con il dire che ci troviamo all'interno di uno spazio bi-dimensionale(ascisse e coordinate), il robot è **planare**, **fisso**[1], possiede tre giunti **rotoidali** e tre bracci di dimensioni finite. Il primo e il secondo braccio possiedono la lunghezza di *1m*, il terzo, al quale sarà collegato l'**end effector**[2], è un *polso* lungo *0.1m*. Vediamo un'immagine rappresentativa che ci faccia capire meglio la sua struttura.

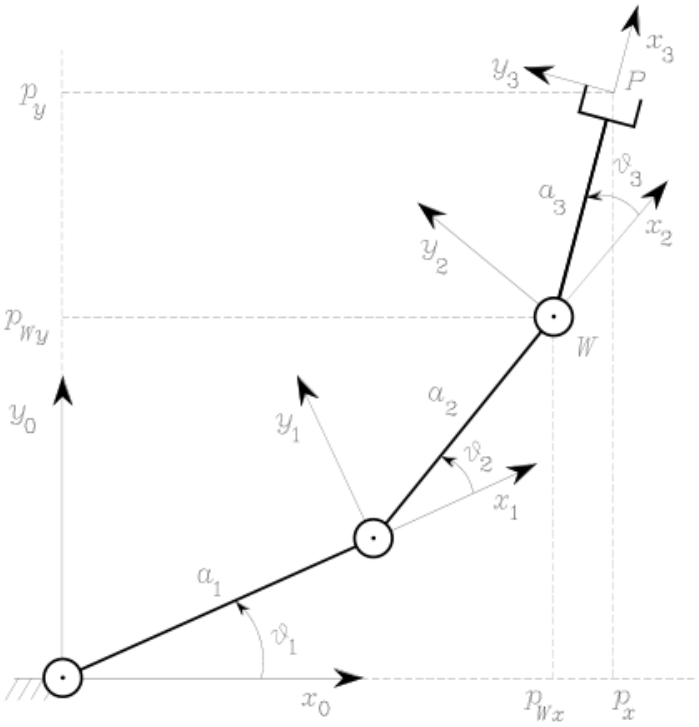


Figure 1: Robot a 3 bracci

Possiamo notare che l'immagine concorda con quanto specificato poco prima, in quanto riporta il primo giunto fissato "a terra", essendo pero' un'immagine rappresentativa, non vi sono le misure date in input dei bracci, difatti dall'immagine sembrano di uguale lunghezza. Inoltre per semplicità di comprensione specifico che nella seguente relazione, quelli che nell'immagine sono evidenziati come $\theta 1, \theta 2, \theta 3$, ovvero le variabili di giunto, saranno **q1, q2, q3**, mentre le lunghezze dei bracci che sono raffigurati come **a1, a2, a3** saranno **L1, L2, L3**.

[1] Uno dei giunti è fissato nel sistema di riferimento
[2] Parte terminale del robot

1.2 Strumenti utilizzati

Gli strumenti a livello software che verranno utilizzati, saranno strumenti propri di matlab, come per esempio live script e plugin relativi alla robotica, e poi per quanto rigurda la raffigurazione di qualche tipo di grafico, è stato utilizzato anche python con il supporto di alcune librerie grafiche come *numpy* e *matplotlib*.

2 Percorso Quadrato

2.1 Richiesta

In questa prima parte del progetto, si vuole che il robot in nostro possesso descriva nel piano la figura di un quadrato, ci vengono date per la seguente figura le coordinate e le direttive sulla posa dell'end effector e di conseguenza dell'ultimo braccio. Le coordinate del quadrato sono :

$$P0 = P4 = (0.3, 0.3)$$

$$P1 = (0.3, 0.7)$$

$$P2 = (0.7, 0.7)$$

$$P3 = (0.7, 0.3)$$

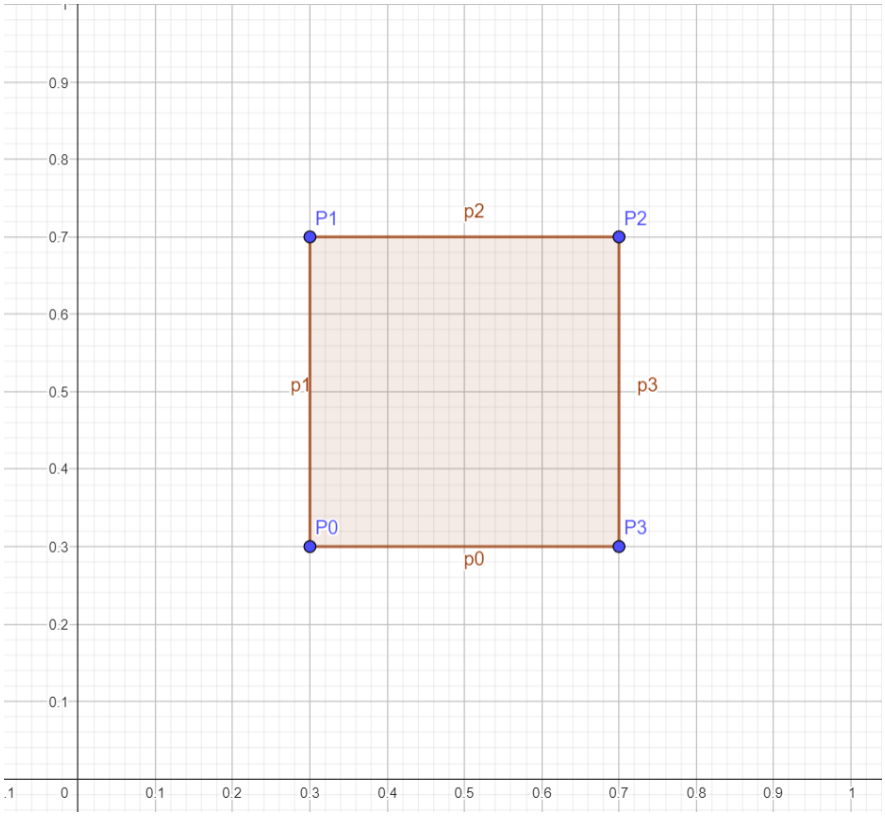


Figure 2: Quadrato nel piano

Il nostro robot dovrà percorrere la figura, dapprima da P0 a P1, poi da P1 a P2, da P2 a P3 e infine da P3 a P4, ci sono però ulteriori richieste che riguardano come dovrà essere orientato l'end effector : da P0 a P1 e da P1 a P2 dovrà essere interno al quadrato, da P2 a P3 e da P3 a P4 dovrà essere esterno; queste direttive che ora verranno graficate, danno anche informazioni sulla disposizione del terzo giunto, dal momento che il polso non ha lunghezza nulla.

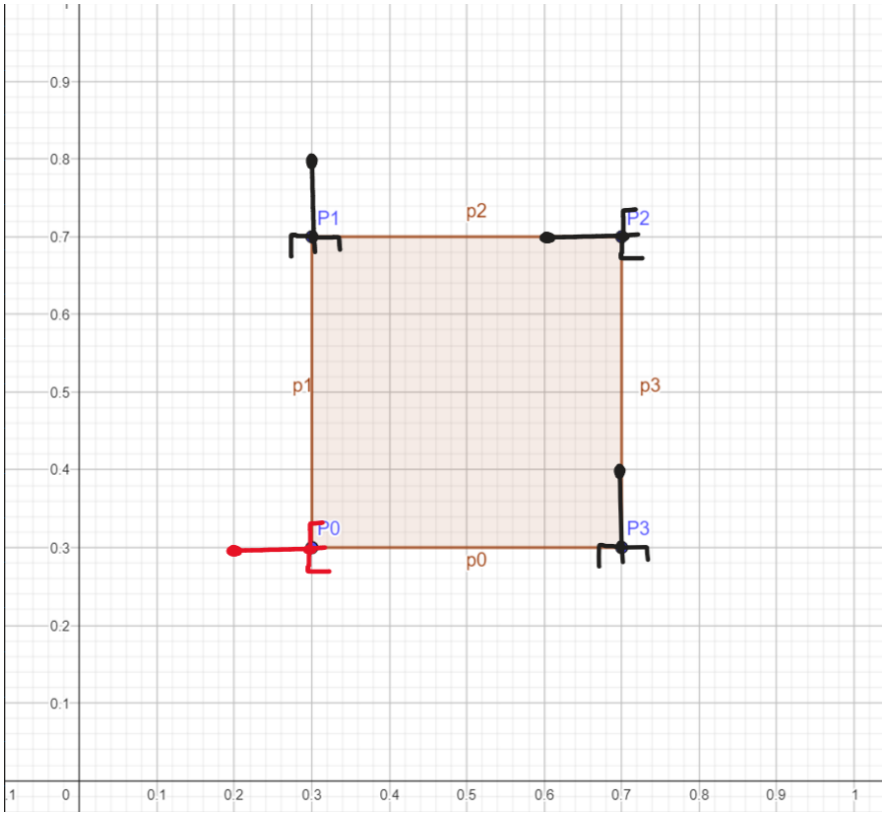


Figure 3: Schema di movimento

Nell'immagine ho preferito omettere il resto del robot per due motivi essenziali : il primo è che questo è comunque una bozza di quello che dovrà venire fuori, il secondo è che l'unico modo per sapere esattamente i valori delle variabili di giunto è con l'applicazione della cinematica, per ora quello che si sa è solo come si vuole l'end effector nel movimento.

2.2 Approccio iniziale

La prima cosa che si deve fare è sicuramente quella di posizionarsi nel punto iniziale P0, seguendo le direttive desiderate, per fare ciò sfruttiamo la **cinematica inversa**, attenendoci inoltre a quelli che sono i parametri di DH[3].Questo approccio consiste nel ricavare le variabili di giunto, conoscendo lunghezza dei bracci e posizione desiderata, per farlo avremo bisogno della matrice di **rototraslazione** del robot, così da poter in seguito ricavare un algoritmo matlab che ci faciliti i calcolie generalizzi le operazioni.

[3]parametri di denavit hartenberg, servono per andare ad assegnare i sistemi di riferimento

2.2.1 Matrice di rototraslazione

In precedenza si era già parlato di quella che è la posa del robot, ovvero posizione + orientamento, le informazioni sulla posizione e sull'orientamento sono racchiuse nella matrice di rototraslazione, così composta :

$$\boldsymbol{A}_i^{i-1}(\vartheta_i) = \begin{bmatrix} c_i & -s_i & 0 & a_i c_i \\ s_i & c_i & 0 & a_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad i = 1, 2, 3.$$

Per avere la matrice finale possiamo procedere in due modi : quello diretto, per cui si fanno tutte le considerazioni necessarie e si trova la matrice totale, oppure quello algoritmo, che risulta essere maggiormente indicato soprattutto quando la struttura va a complicarsi, e consiste nella rappresentazione della matrice di rototraslazione rispetto ad ogni braccio e alla fine le varie matrici verranno moltiplicate. Personalmente ho preferito seguire la seconda strada e questa è la matrice risultante :

$$\boldsymbol{T}_3^0(\boldsymbol{q}) = \boldsymbol{A}_1^0 \boldsymbol{A}_2^1 \boldsymbol{A}_3^2 = \begin{bmatrix} c_{123} & -s_{123} & 0 & a_1 c_1 + a_2 c_{12} + a_3 c_{123} \\ s_{123} & c_{123} & 0 & a_1 s_1 + a_2 s_{12} + a_3 s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Questa matrice è di notevole importanza per i nostri scopi, difatti ci permetterà di applicare la **cinematica inversa**, ovvero, ricavare le variabili di giunto a partire dalle informazioni relative ai bracci e alla posizione finale. Tramite infatti una serie di calcoli arriviamo alla conclusione per cui :

$$cos(q2) = \frac{px^2 + py^2 - l2^2 - l1^2}{2 \cdot l1 \cdot l2}$$

$$sin(q2) = \sqrt{1 - (cos(q2))^2}$$

Per q2 basterà utilizzare l'operazione di atan2().

e inoltre dobbiamo considerare anche un'altra cosa, che riguarda q3, ovvero l'ultima variabile di giunto; quello che vogliamo nel primo step è che l'end effector sia perpendicolare al lato del quadrato(che per ora è ancora un segmento "immaginario"), dunque la somma degli angoli dovrà essere pari a 0 :

$$q1 + q2 + q3 = 0 \Rightarrow q3 = -q1 - q2$$

2.2.2 Posizionamento iniziale

Ora si hanno a diSsposizione tutti gli elementi per la costruzione di un algoritmo, e dunque ci spostiamo su matlab.

```
function [angoli] = cinematica_inversa(l1,l2,l3,px, py, verso)
```

```
%funzione che implementa la cinematica inversa dove
%l1, l2, l3 sono le lunghezze dei tre bracci e
%px e py le coordinate del punto in cui si vuole portare il robot
%restituisce un vetore di angoli, corrispondenti alle variabili di
%giunto q1, q2, q3


%dal momento che il vettore degli angoli ci restituisce q1,q2 e q3
%   importante specificare che per come    sto definito il problema

%la prima cosa che si deve andare a fare    capire se il sistema
%ammette soluzioni, dunque se il punto che abbiamo come destinazione
%   all'interno dello spazio su cui andiamo a lavorare
if sqrt(px^2+py^2) > (l1+l2+l3)
    disp("il punto (py,px)    fuori dal range di lavoro")
    return
end

if l2<l1 && (sqrt(px^2+py^2) < (l1-l2))
    disp("il punto (py,px)    fuori dal range di lavoro")
    return
end


c2=(px^2+py^2-l2^2-l1^2)/(2*l1*l2);

s2=sqrt(1-c2^2); % in verit  qui si potrebbe usare anche =-sqrt(1-c2
   ^2)
%questo dipende se si preferisce una soluzione a gomito alto o a
   gomito
%basso

c1s1=(1/(l1^2+l2^2+2*l1*l2*c2))*[l1+l1*c2 , l2*s2; -l2*s2, l1+l2*c2]*[
   px;py];

c1= c1s1(1:1);
s1=c1s1(2:2);
q1=atan2(s1,c1);
q2=atan2(s2,c2);

%per sapere se ci stiamo spostando perpendicolarmente
```

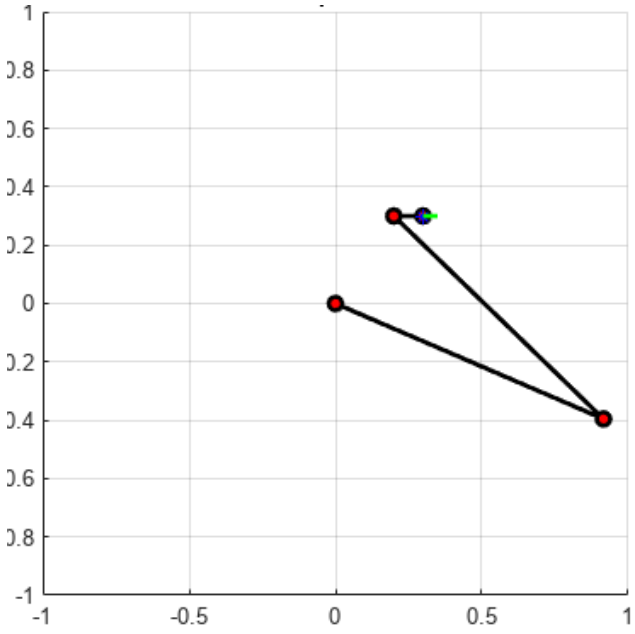
```
    %al lato verticale o a quello orizzontale
    if verso == 'lato'
        q3 = -q1 - q2;
    else if verso == 'alto'
        q3 = -90 - q1 - q2;
    end
    end
    angoli=[q1,q2,q3];

end
```

A questo punto ci siamo posizionati nel vertice in basso a sinistra del quadrato e la configurazione iniziale dei giunti, con *gomito basso* è :

ans = 1×3

	1	2	3
1	-0.4067	2.7791	-2.3723

Grafichiamo il tutto tramite una funzione di animazione definita in matlab con il tool apposito :



2.2.3 Movimento e rotazione

Quello che vogliamo ora, è muoverci lungo il segmento, mantenendo l'end effector verso l'interno, fino al punto (0.3, 0.7), per farlo ho realizzato una funzione che parametrizza il segmento con un λ che va da 0 a 1, e per ogni lambda che percorre restituisce la configurazione degli angoli, successivamente dal momento che vogliamo conoscere la velocità gli passeremo la matrice **Jacobiana** del nostro robot, e con sfruttando la relazione :

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J(q)\dot{q}$$

Prima di passare avanti è utile aprire una parentesi su quello che è la matrice jacobiana, questa è la derivata della matrice di rototraslazione, e la si può ottenere in modo **analitico** oppure in modo **geometrico**, a seconda del metodo che viene scelto si otterranno le omonime matrici, che risulteranno essere differenti, ma comunque collegate da una relazione che permetterà di passare da una all'altra. Personalmente ho utilizzato per questo progetto uno *jacobiano analitico* :

$$\begin{bmatrix} l1 \cdot s1 - l2 \cdot s12 - l3 \cdot s123 & -l2 \cdot s12 - l3 \cdot s123 & -l3 \cdot s123 \\ l1 \cdot c1 + l2 \cdot c12 + l3 \cdot c123 & l2 \cdot c12 + l3 \cdot c123 & l3 \cdot c123 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Questa è la funzione che ho utilizzato :

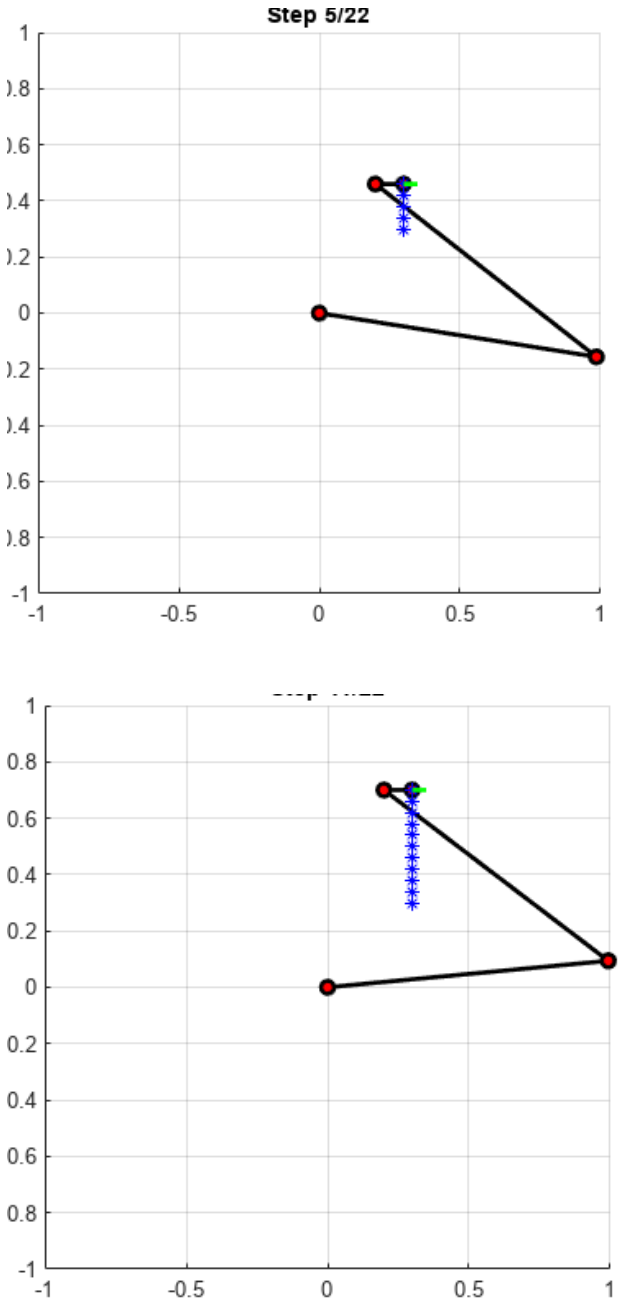
```
function [angoli, vel, tempo] = movimenti(P1, P2, l1, l2, l3, verso)
    %istanzio i tempi
    t_inizio = 0;
    stepTime = 0.1;
    t_fine = 1.5;
    tempo = t_inizio:stepTime:t_fine;
    durata = t_fine - t_inizio;
    %normalizzo il tempo
    tau = (tempo - t_inizio) / durata;
    %implemento la legge oraria
    lambda = 3*tau.^2 - 2*tau.^3;

    angoli = zeros(length(lambda), 3);
    vel = zeros(length(lambda), 3);

    %mi calcolo con la funzione di cinematica inversa le configurazioni
       dei
    %giunti
    for i = 1:length(lambda)
        %parametrizzazione del segmento
        p = P1 + lambda(i)*(P2 - P1);
        angoli(i,:) = cinematica_inversa(l1, l2, l3, p(1), p(2), verso);
    end
```

```
    %istanzio la derivata che mi servir  poi per la velocit
    dq_dt = zeros(size(angoli));

    for i = 2:length(angoli)-1
        dq_dt(i,:) = (angoli(i+1,:) - angoli(i-1,:)) / (2*stepTime);
    end

    dq_dt(1,:) = 0;
    dq_dt(end,:) = 0;

    for i = 1:length(lambda)
        %per una questione di leggibilit
        q1 = angoli(i,1); q2 = angoli(i,2); q3 = angoli(i,3);
        %preparo la jacobiana
        J = [ l1*sin(q1) - l2*sin(q1+q2) - l3*sin(q1+q2+q3), -l2*sin(q1+q2
           ) - l3*sin(q1+q2+q3), -l3*sin(q1+q2+q3);
              l1*cos(q1) + l2*cos(q1+q2) + l3*cos(q1+q2+q3),  l2*cos(q1+q2
                 ) + l3*cos(q1+q2+q3),  l3*cos(q1+q2+q3);
              0, 0, 0;
              0, 0, 0;
              0, 0, 0;
              1, 1, 1 ];
        v_cartesiane = J * dq_dt(i,:)';
        %ottengo le velocit
        vel(i,:) = [v_cartesiane(1:2)' v_cartesiane(6)];
    end
end
```

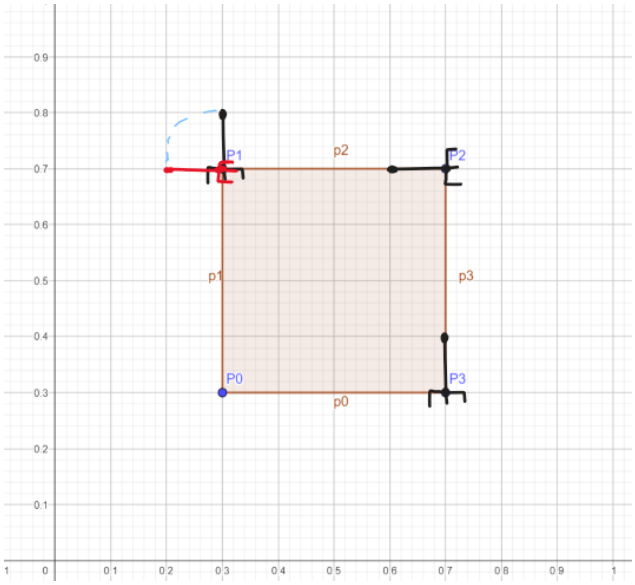
Adesso andiamo a richiamare la funzione, inserendo le proprietà dei bracci del robot, quindi la loro lunghezza e i punti da cui farlo partire e arrivare :

```
P1 = [0.2, 0.3];
P2 = [0.2, 0.7];
l1 = 1;
l2 = 1;
l3 = 0.1;

[angoli, vel,t1] = movimenti(P1, P2, l1, l2, l3, 'lato');
```

Sfruttiamo anche la funzione di animazione, nella relazione non posso inserire il video dell'animazione(sarà comunque disponibile all'interno dello script), dunque inserisco le immagini intermedie che ricavo :



È arrivato il momento di applicare la rotazione, visualizziamo l'obbiettivo e il punto di partenza, in rosso è indicato il punto in cui siamo arrivati applicando l'ultimo algoritmo, il 'pettinino' nero invece indica la posa che vogliamo fargli assumere



Il vincolo da tenere in conto è quello per cui il centro dell'end effector debba rimanere fermo, dunque saranno i giunti q1, q2 e q3 che dovranno muoversi, e la traiettoria più semplice che si possa percorrere per arrivare a quella posa è un quarto di circonferenza. Dunque quello che è stato fatto è stato, come prima cosa individuare la parametrizzazione della circonferenza, individuando il centro e il raggio, il centro non è altro che il punto in cui vogliamo che il centro del 'pettinino' stia fermo, dunque (0.3, 0.7), il raggio è invece equivalente alla lunghezza del terzo braccio (0.1m); la parametrizzazione di una generale circonferenza è :

$$\begin{cases} x = x_c + r \cdot cos(2\pi t) \\ y = y_c + r \cdot sin(2\pi t) \end{cases}$$

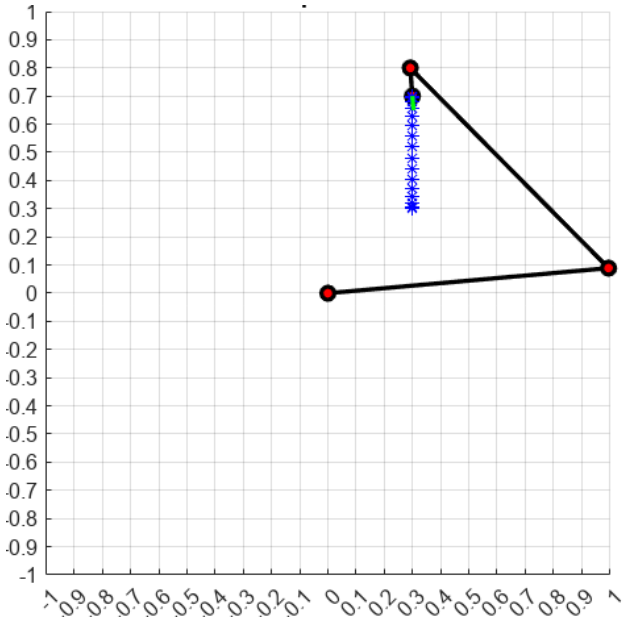
Implementiamo tutto quello che è stato detto sotto forma di algoritmo in matlab :

```
function [angoli, vel, tempo] = movimenti_rotoazionali(xe, ye, l1, l2, l3,
    orientamento)
    %istanzio il tempo
    t_inizio = 0;
    stepTime = 0.1;
    t_fine = 1.5;
    tempo = t_inizio : stepTime : t_fine;
    durata = t_fine - t_inizio;
    tau = (tempo - t_inizio) / durata;
    %legge oraria
    lambda = 3*tau.^2 - 2*tau.^3;

    %scelgo come farlo girare
    if orientamento == 0
        theta = pi - (pi/2) * lambda;
```

```
    elseif orientamento == 1
        theta = pi/2 - ((3*pi)/2) * lambda;
    end

    %legge oraria
    px = xe + l3 * cos(theta);
    py = ye + l3 * sin(theta);

    %preparo i vettori
    angoli = zeros(length(lambda), 3);
    vel = zeros(length(lambda), 3);

    for i = 1:length(lambda)
        x3 = px(i); y3 = py(i);
        dx = xe - x3; dy = ye - y3;
        q3 = atan2(dy, dx);
        [q1, q2] = ik_2link(x3, y3, l1, l2);
        angoli(i,:) = [q1, q2, q3 - q1 - q2];
    end

    dq_dt = zeros(size(angoli));
    for i = 2:length(angoli)-1
        dq_dt(i,:) = (angoli(i+1,:) - angoli(i-1,:)) / (2*stepTime);
    end
    dq_dt(1,:) = 0;
    dq_dt(end,:) = 0;

    for i = 1:length(lambda)
        q1 = angoli(i,1); q2 = angoli(i,2); q3 = angoli(i,3);
        J = [ l1*sin(q1) + l2*sin(q1+q2) + l3*sin(q1+q2+q3), ...
              l2*sin(q1+q2) + l3*sin(q1+q2+q3), l3*sin(q1+q2+q3);
             -l1*cos(q1) - l2*cos(q1+q2) - l3*cos(q1+q2+q3), ...
             -l2*cos(q1+q2) - l3*cos(q1+q2+q3), -l3*cos(q1+q2+q3);
              0, 0, 1 ];
        vel(i,:) = J * dq_dt(i,:)';
    end
end


% IK per braccio planare a 2 link
function [q1, q2] = ik_2link(x, y, l1, l2)
```

```
    r = sqrt(x^2 + y^2);
    cos_q2 = (r^2 - l1^2 - l2^2) / (2 * l1 * l2);
    sin_q2 = sqrt(1 - cos_q2^2);  % gomito gi
    q2 = atan2(sin_q2, cos_q2);

    k1 = l1 + l2 * cos(q2);
    k2 = l2 * sin(q2);
    q1 = atan2(y, x) - atan2(k2, k1);
end

%richiamo la funzione
[angoli2,vel2,t2] = movimenti_rotoazionali(0.3, 0.7, l1,l2,l3, 0);
```

Ecco quello che accade all'interno dell'animazione :



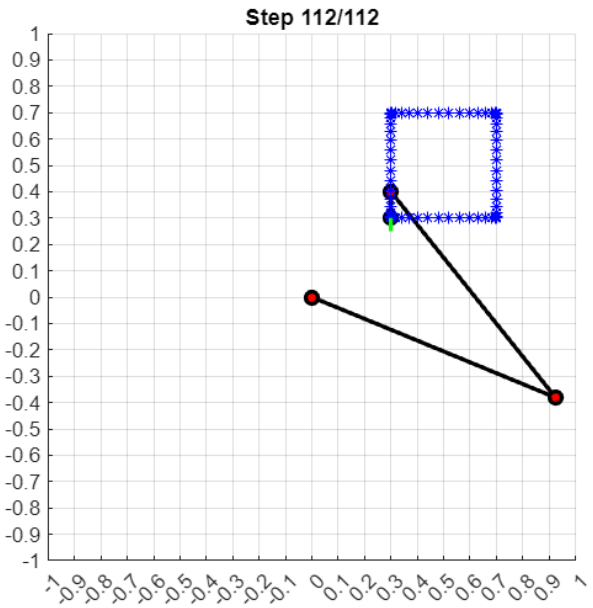
Ora che abbiamo visto come le due funzioni create operano, di seguito inserisco i passaggi per avere il tracciamento della figura quadrata(ovviamente tralascio i primi due tratti già fatti).

```
P3 = [0.3, 0.8];
P4 = [0.7, 0.8];

[angoli3,vel3,t3] = movimenti(P3,P4,l1,l2,l3, 'alto');
[angoli4,vel4,t4] = movimenti_rotoazionali(0.7,0.7, l1, l2,l3,1);
[angoli5,vel5,t5] = movimenti([0.6,0.7],[0.6,0.3],l1,l2,l3, 'lato');
[angoli6,vel6,t6] = movimenti_rotoazionali(0.7,0.3,l1,l2,l3,0);
[angoli7,vel7,t7] = movimenti([0.7,0.4],[0.3,0.4],l1,l2,l3, 'alto');
```

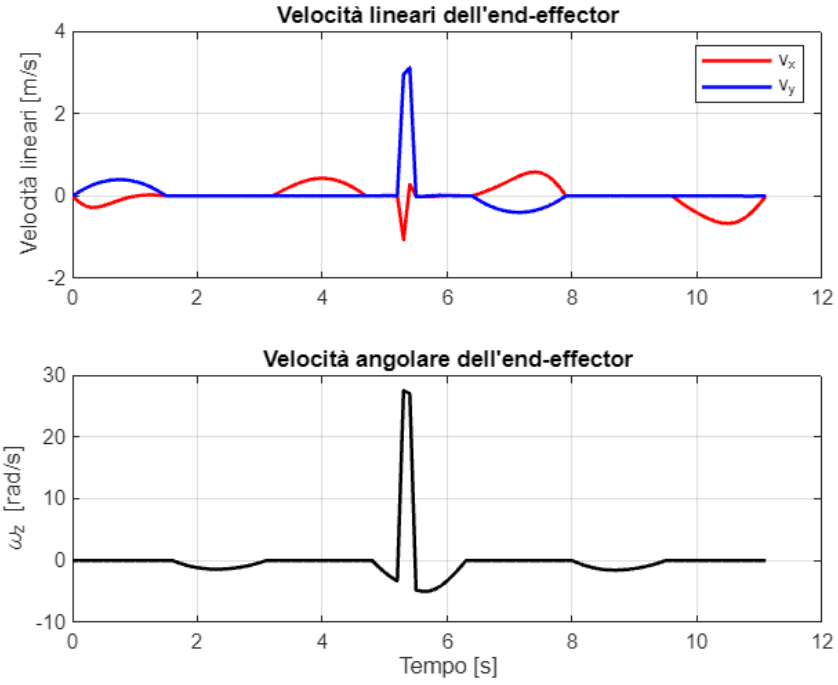
```
%vettore dei tempi per plottare in maniera opportuna le velocit
t2 = t2 + t1(end) + 0.1;
t3 = t3 + t2(end) + 0.1;
t4 = t4 + t3(end) + 0.1;
t5 = t5 + t4(end) + 0.1;
t6 = t6 + t5(end) + 0.1;
t7 = t7 + t6(end) + 0.1;
tempo_totale = [t1, t2, t3, t4, t5, t6, t7]

%raccolgo tutti gli angoli e tutte le velocit
[angoli_totali] = [angoli; angoli2; angoli3;angoli4;angoli5;angoli6;
    angoli7];
vel = vel(:,1:3);
vel2 = vel2(:,1:3);
vel3 = vel3(:,1:3);
vel4 = vel4(:,1:3);
vel5 = vel5(:,1:3);
vel6 = vel6(:,1:3);
vel7 = vel7(:,1:3);
vel_totali = [vel;vel2;vel3;vel4;vel5;vel6;vel7];
```

2.3 Grafici

A questo punto possiamo andare sia a fare l'animazione, che ci porterà a questa configurazione :



e sia le velocità :

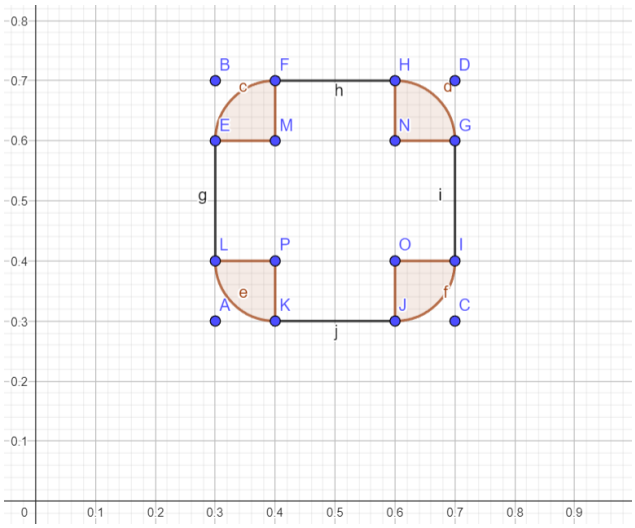


Possiamo notare dei particolari, nella velocità angolare quando si attraversano i tratti rettilinei del quadrato questa risulta essere zero, il picco invece non è un errore e non si tratta neanche di rumore, nasce in quanto nella seconda rotazione che viene fatta, il cambiamento di angolo è volutamente brusca(proprio per far notare il distacco), in quanto non prende la strada minima per girare.

3 Percorso Quadrato con angoli smussati

3.1 Richiesta

Per quanto riguarda la seconda richiesta, dunque il **quadrato smussato**, dobbiamo fare dei ragionamenti, per i lati rettilinei del quadrato, vale la stessa funzione movimenti, che è stata precedente utilizzata, ovviamente però bisogna capire quando andare a tracciare il raccordo, per evitare equivoci vediamo un grafico.



3.2 Modifiche rispetto al precedente

Per i tratti circolari, non vogliamo la stessa cosa che succede con il quadrato, perché lì il centro dell'end effector sta fermo e tutto il resto si orienta, questa volta l'end effectior dovrà proprio andare a seguire e tracciare l'arco di circonferenza. Dal precedente script quella che viene modificata è la funzione movimentiRotazionali.

```
function [angoli, vel, tempo] = movimenti_roto(centro_x, centro_y, raggio,
    vecchio_or, nuovo_or, l1, l2, l3)

    t_inizio = 0;
    stepTime = 0.1;
    t_fine = 1.5;
    tempo = t_inizio : stepTime : t_fine;
    durata = t_fine - t_inizio;

    tau = (tempo - t_inizio) / durata;
    lambda = 3 * tau.^2 - 2 * tau.^3;

    angoli = zeros(length(lambda), 3);
    vel = zeros(length(lambda), 3);

    for i = 1:length(lambda)
        orientamento = vecchio_or - (vecchio_or - nuovo_or) * lambda(i);
        px = centro_x - (raggio + l3) * cos(orientamento);
        py = centro_y - (raggio + l3) * sin(orientamento);
        angoli(i,:) = cinematica_inversa(l1, l2, l3, px, py, orientamento)
            ;
    end

    dq_dt = zeros(size(angoli));
    for i = 2:length(angoli)-1
        dq_dt(i,:) = (angoli(i+1,:) - angoli(i-1,:)) / (2 * stepTime);
    end
    %forzo le derivate
    dq_dt(1,:) = 0;
    dq_dt(end,:) = 0;

    for i = 1:length(lambda)
        q1 = angoli(i,1);
        q2 = angoli(i,2);
        q3 = angoli(i,3);
```

```
        J = [
             l1*sin(q1) + l2*sin(q1+q2) + l3*sin(q1+q2+q3),   l2*sin(q1+q2
                ) + l3*sin(q1+q2+q3),   l3*sin(q1+q2+q3);
            -l1*cos(q1) - l2*cos(q1+q2) - l3*cos(q1+q2+q3),  -l2*cos(q1+q2
                ) - l3*cos(q1+q2+q3),  -l3*cos(q1+q2+q3);
             0, 0, 1
        ];

        vel(i,:) = J * dq_dt(i,:)';
    end
end
```

In questa funzione rispetto alla precedente sono stati aggiunti come input il *raggio* e poi il *vecchio orientamento e il nuovo orientamento*, e viene utilizzata la funzione di cinematica inversa sfruttata anche dalla funzione movimenti. Grazie a queste modifiche senza problemi otterremo l'effetto desiderato, implementiamo e vediamo gli output.

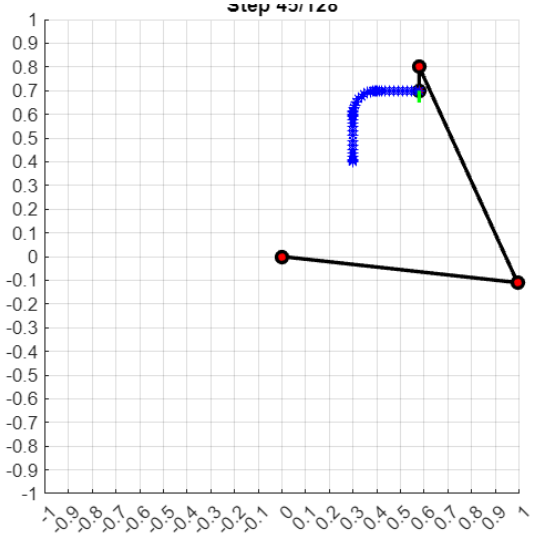
3.3 Impostazione dei dati e richiamo delle funzioni

```
%mettiamo i nostri dati
l1 = 1;
l2 = 1;
l3 = 0.1;
%definisco i punti
P0 = [0.2,0.4];
P1 = [0.2,0.6];

[angoli,vel,tempo] = movimenti(P0,P1,l1,l2,l3,0);

c1_x = 0.4;
c1_y = 0.6;

[angoli_rot1,vel_rot1,tempo_rot1] = movimenti_roto(c1_x,c1_y,l3,0,-pi/2,l1
    ,l2,l3);

P2 = [0.4,0.8];
P3 = [0.6,0.8];

[angoli2,vel2,tempo2] = movimenti(P2, P3, l1,l2,l3,-pi/2);

c2_x = 0.6;
c2_y = 0.6;
```

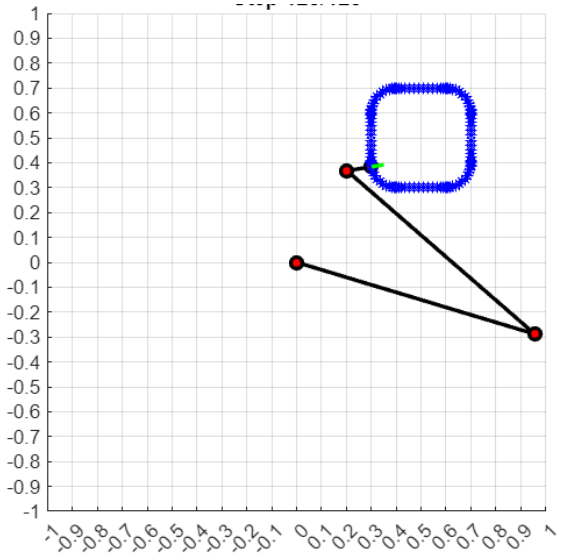
```
[angoli_rot2,vel_rot2,tempo_rot2] = movimenti_roto(c2_x,c2_y,l3,-pi/2,-pi,
    l1,l2,l3);

P4 = [0.8,0.6];
P5 = [0.8,0.4];
[angoli3,vel3,tempo3] = movimenti(P4, P5, l1,l2,l3,pi);
c3_x = 0.6;
c3_y = 0.4;

[angoli_rot3, vel_rot3,tempo_rot3] = movimenti_roto(c3_x, c3_y, l3, -pi,
    -(3*pi)/2,l1,l2,l3 );

P6 = [0.6,0.2];
P7 = [0.4,0.2];

[angoli4, vel4,tempo4] = movimenti(P6,P7,l1,l2,l3,pi/2);

c4_x = 0.4;
c4_y = 0.4;

[angoli_rot4, vel_rot4, tempo_rot4] = movimenti_roto(c4_x, c4_y, l3, -(3*
    pi)/2, -2*pi,l1,l2,l3 );

%definisco anche qui il tempo

tempo_rot1 = tempo_rot2 + tempo(end) + 0.1;
tempo2 = tempo2 + tempo_rot1(end) + 0.1;
tempo_rot2 = tempo_rot2 + tempo2(end) + 0.1;
tempo3 = tempo3 + tempo_rot2(end) + 0.1;
tempo_rot3 = tempo_rot3 + tempo3(end) + 0.1;
tempo4 = tempo4 + tempo_rot3(end) + 0.1;
tempo_rot4 = tempo_rot4 + tempo4(end) + 0.1;

tempo_tot = [tempo, tempo_rot1, tempo2, tempo_rot2, tempo3, tempo_rot3,
    tempo4, tempo_rot4];
angoli_tot = [angoli;angoli_rot1;angoli2;angoli_rot2;angoli3;angoli_rot3;
    angoli4;angoli_rot4];

vel_tot = [vel;vel_rot1;vel2;vel_rot2;vel3;vel_rot3;vel4;vel_rot4];
```

3.4 Grafici

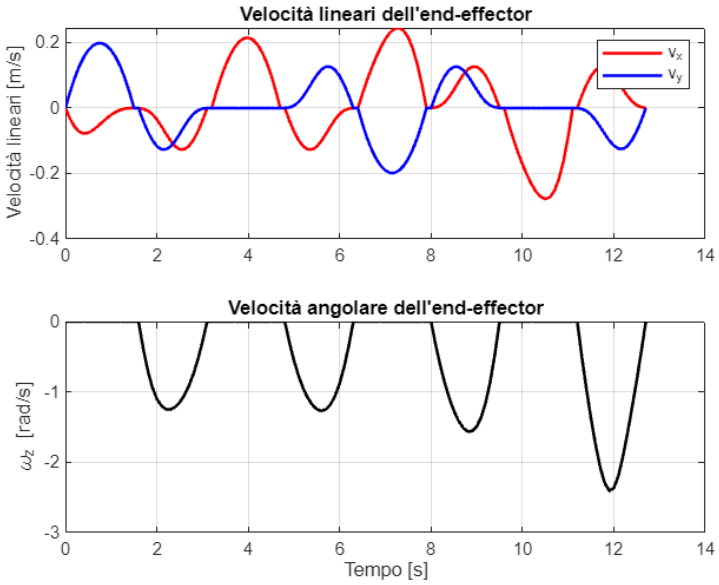
Con l'animazione quello che otteniamo è :



Alla fine avremo :



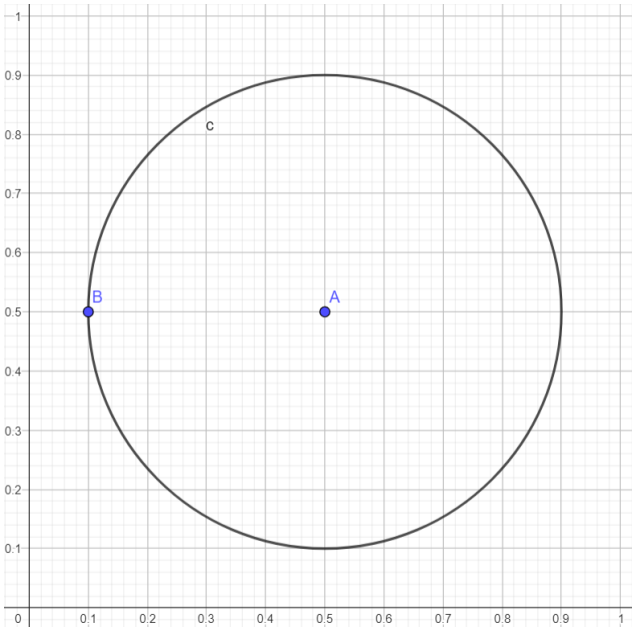
L'ultima cosa che rimane da fare è plottare le velocità come fatto in precedenza grazie al vettore tempo_totale e vel_totale :



4 Percorso Circonferenza

4.1 Richiesta

Ultima richiesta è quella del tracciamento di una circonferenza, dal momento che non sono state date particolari direttive riguardo le coordinate del centro e la lunghezza del raggio, ho arbitrariamente scelto un centro (0.5,0.5) e raggio 0.4. Viene inoltre richiesto che l'end effector sia per tutta la traiettoria *tangente* alla circonferenza.



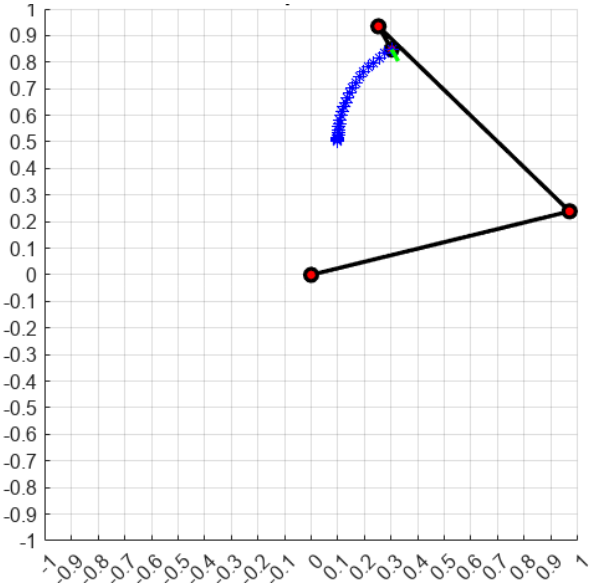
4.2 Approccio e Implementazione

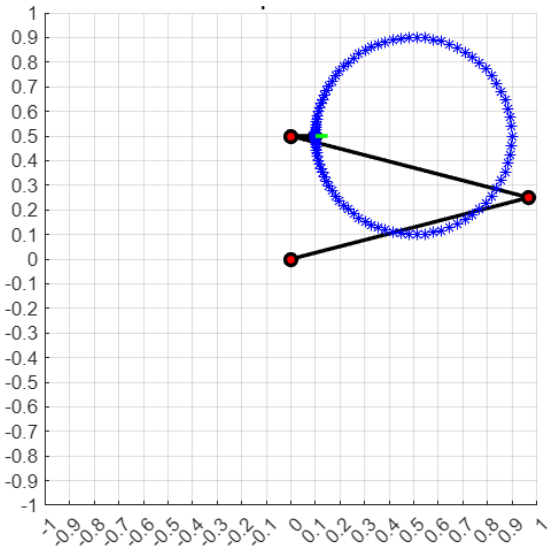
L'approccio al tracciamento della circonferenza viene più semplice, in quanto nelle precedenti richieste abbiamo già lavorato sul movimento rotazionale dove l'end effector risulta essere tangente. Quindi basterà andare a prendere quella funzione e cambiare solamente i dati da dargli in input. Difatti per la circonferenza per come abbiamo parametrizzato, vecchio_or $= 2\pi$, nuovo_or $= 0$ quindi fa un giro completo.

```
l2 = 1;
l3 = 0.1;
raggio = 0.4;
centro_x = 0.5;
centro_y = 0.5;

[angoli_circ, vel_circ,tempo] = movimenti_roto(centro_x,centro_y,raggio,2*
    pi,0,l1,l2,l3);
animazione_robot_planare(angoli_circ, l1,l2,l3)
grafico_vel(vel_circ,tempo)
```

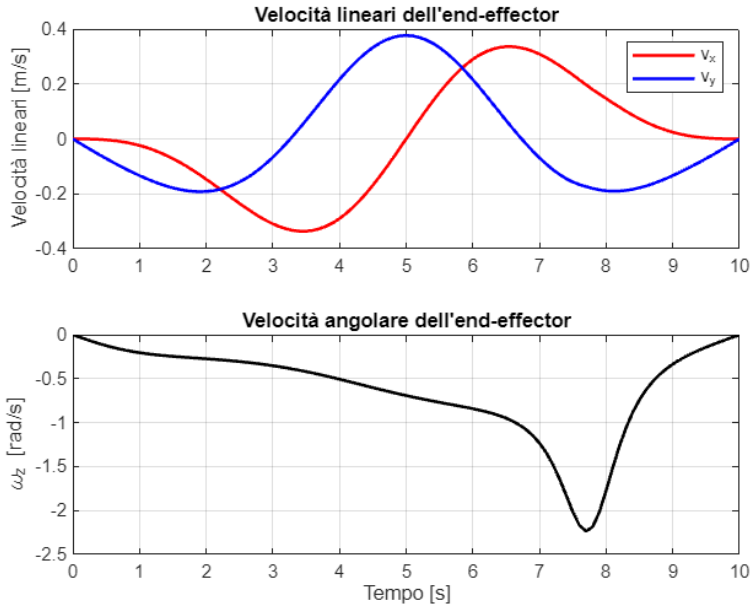
4.3 Grafici

Animazione :

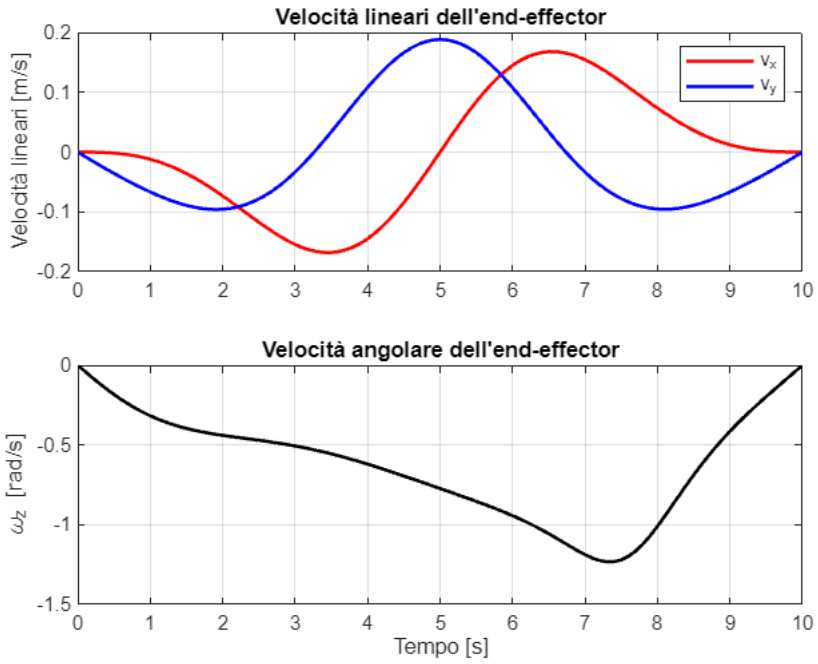




Velocità:



Possiamo osservare che anche qui la sottoelongazione si verifica in corrispondenza di un cambio brusco, o meglio un passaggio in cui gli angoli si muovono maggiormente(dall'animazione lasciata nello script si capisce meglio questo passaggio). Ho inoltre fatto alcune prove e ovviamente questo picco di sottoelongazione è meno presente quando il raggio è minore, questo è un esempio con la circonferenza che ha il centro nelle stesse coordinate di prima, ma un raggio pari a 0.2 :



5 Conclusione

Durante il seguente progetto è stato possibile osservare come avviene la manipolazione di un robot planare a tre bracci, avente tre variabili di giunto e con un polso di lunghezza non nulla; si sono affrontati differenti problemi di difficoltà diversa. L'approccio per lo svolgimento dell'elaborato è stato fin da subito quello di capire il problema, e scomporlo in più piccoli problemi da andare a soddisfare, difatti la prima cosa fatta è stata creare la funzione di *cinematica inversa* che è stata centrale in ogni altra funzione scritta, proprio perchè permette di ottenere la configurazione degli angoli a partire dalla posizione desiderata. Dopo di che si sono create delle funzioni di movimento che soddisfassero le varie esigenze, per le varie casistiche. A proposito di ciò, sicuramente la cosa che poteva essere migliorata in questo elaborato, è sicuramente la modularità delle funzioni e quindi la possibilità di utilizzarle senza a dare a modificarle così tanto. Per finire attraverso le funzioni di animazione e grafico c'è stata la possibilità di osservare in maniera empirica, facendo quindi prove e visualizzando come a diversi input corrispondessero output totalmente differenti, i vari comportamennti, che mi hanno permesso di trarre conclusioni anche su quelle che sono le varie dinamiche proposte.