Progetto Robotica Industriale

1. Discussione dello script *mainTriangolo.m*

Considerando i dati presenti nel PUMA560.pdf

```
clear, clc, close all

syms ("theta",[1,3])
a = [0; 0.9; 0.9];
alpha = [pi/2; 0; 0];
d = [0; 0; 0];
THETA = [theta1; theta2; theta3];

DH = [a,alpha,d,THETA]

Rb0 = [ 1 0 0 0.5;
        0 1 0 0.5;
        0 0 1 1;
        0 0 0 1 ];

P1 = [0.8; 0.8; 0.5];
P2 = [1.2; 0.8; 0.5];
P3 = [1.0; 1.2; 0.5];

P = [P1,P2,P3,P1];

Te = 40;
numRoute = 3;

typePath = 'Triangolo';
```

si istanziano i seguenti parametri:

- la tabella di Denavit-Hartenberg
- la matrice di rototraslazione R_0^b che lega il sistema di riferimento SRb e SR0
- la sequenza dei punti di interesse
- il tempo di percorrenza della traiettoria
- il numero di archi orientati della sequenza
  P1 -> P2 -> P3 -> P1
- la tipologia di percorso da seguire

Successivamente si passa alla fase di definizione temporale

```
%% 1 Definizione del tempo
numRange = numRoute+1;
Trange = linspace(0,Te,numRange);

numStep = input("1. numStep per discretizzare l'intervallo di tempo: ");
trange = zeros(numRoute,numStep);
for i=1:numRange-1
    trange(i,:) = linspace(Trange(i),Trange(i+1),numStep);
end
```

Dove si è deciso di costruire un vettore e una matrice

$$Trange_{1xnumRange} = [0, \dots, Te]$$

in cui $Trange(i)$ corrisponde all'istante di tempo in cui tocco il punto della sequenza $P(i)$.

Mentre

$$trange_{numRoutexnumStep} = \begin{bmatrix} Trange(1) & \cdots & Trange(2) \\ Trange(2) & \dots & Trange(3) \\ Trange(3) & \cdots & Trange(4) \end{bmatrix} = \begin{bmatrix} P1 & \rightarrow & P2 \\ P2 & \rightarrow & P3 \\ P3 & \rightarrow & P1 \end{bmatrix}$$

In cui ogni riga della matrice rappresenta l'intervallo di tempo discretizzato per percorrere un arco della sequenza.

Completata la fase di definizione del tempo, si passa alla fase di definizione della traiettoria

```
%% 2 Definizione della traiettoria
degree = input("2. ordine del polinomio lambda: ");
lambdaFun = lambdaFunction(degree);

position = zeros(3,numRoute*numStep); % pos = [ [x;y;z], ... ]
velocity = zeros(3,numRoute*numStep); % vel = [ [vx;vy;vz], ... ]

syms s;
k=0;
for i=1:numRoute
    for j=1:numStep
        t = trange(i,j);
        T2 = Trange(i+1);
        T1 = Trange(i);

        P1 = P(:,i);
        P2 = P(:,i+1);

        sigma = (t-T1)/(T2-T1);
        lambda = subs(lambdaFun,s,sigma);

        lambdaDot = subs(lambdaFun,s,sigma);

        k=k+1;
        position(:,k) = P1+lambda*(P2-P1);
        velocity (:,k) = ((P2-P1)/(T2-T1))*lambdaDot;
    end
end
```

La legge oraria della traiettoria sarà descritta da:

$$P_{i+1} = P_i + \lambda(\sigma(t)) \cdot (P_{i+1} - P_i)$$

$$v_{i+1} = \frac{(P_{i+1} - P_i)}{(Trange(i+1) - Trange(i))} \cdot \lambda'(\sigma(t))$$

$$\forall t \,\epsilon\, trange(i,:) = [Trange(i), Trange(i+1)]$$

Forma della variabile adimensionale $\sigma \,\epsilon\, [0,1]$

$$\sigma = \frac{t - Trange(i)}{Trange(i+1) - Trange(i)}$$

In particolare, la legge oraria dipenderà dall'andamento del parametro $\lambda \,\epsilon\, [0,1]$, che sarà una funzione dipendente da σ di classe polinomiale restituita in output da *lambdaFunction.m*, il cui parametro è l'ordine (*degree*) del polinomio che si vuole ottenere in output**.**

```
function ret = lambdaFunction(degree)
%% Costruzione polinomio
    switch degree
        case 3
            syms s a0 a1 a2 a3; % poly 4-order

            lambda = a0*s^3 + a1*s^2 + a2*s + a3;
            lambdaDot = 3*a0*s^2 + 2*a1*s + a2;
        case 5
            syms s a0 a1 a2 a3 a4 a5; % poly 6-order

            lambda = a0*s^5 + a1*s^4 + a2*s^3 + a3*s^2 + a4*s + a5;
            lambdaDot = 5*a0*s^4 + 4*a1*s^3 + 3*a2*s^2 + 2*a3*s + a4;
            lambdaDDot = 20*a0*s^3 + 12*a1*s^2 + 6*a2*s + 2*a3;
        case 7
            syms s a0 a1 a2 a3 a4 a5 a6 a7; % poly 8-order

            lambda = a0*s^7 + a1*s^6 + a2*s^5 + a3*s^4 + a4*s^3 + a5*s^2 + a6*s + a7;
            lambdaDot = 7*a0*s^6 + 6*a1*s^5 + 5*a2*s^4 + 4*a3*s^3 + 3*a4*s^2 + 2*a5*s + a6;
            lambdaDDot = 42*a0*s^5 + 30*a1*s^4 + 20*a2*s^3 + 12*a3*s^2 + 6*a4*s + 2*a5;
            lambdaDDDot = 210*a0*s^4 + 120*a1*s^3 + 60*a2*s^2 + 24*a3*s + 6*a4;
        otherwise
            exception = MException('ThisComponent:notFound','Degree %s not found',degree);
            throw(exception);
    end
```

L'ordine del polinomio può essere di 3^o, 4^o o 7^o grado

```matlab
%% Vincoli di continuità
    eq1 = subs(lambda,s,0) == 0;
    eq2 = subs(lambda,s,1) == 1;

    % dot
    eq3 = subs(lambdaDot,s,0) == 0;
    eq4 = subs(lambdaDot,s,1) == 0;

    if degree > 3  % dot dot
        eq5 = subs(lambdaDDot,s,0) == 0;
        eq6 = subs(lambdaDDot,s,1) == 0;
    end

    if degree > 5 % dot dot dot
        eq7 = subs(lambdaDDDot,s,0) == 0;
        eq8 = subs(lambdaDDDot,s,1) == 0;
    end

%% Risoluzione del sistema
    if degree == 3
        [a0, a1, a2, a3] = vpasolve([eq1, eq2, eq3, eq4]);
    end

    if degree == 5
        [a0, a1, a2, a3, a4, a5] = vpasolve([eq1, eq2, eq3, eq4, eq5, eq6]);
    end

    if degree == 7
        [a0, a1, a2, a3, a4, a5, a6, a7] = vpasolve([eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8]);
    end

    ret = subs(lambda);
end
```

Forma del polinomio $\lambda \epsilon [0,1]$ con $k = degree$

$$\lambda(\sigma) = a_0 \cdot \sigma^{\kappa} + \cdots + a_{k-1} \cdot \sigma + a_k$$

Ed in base all'ordine $(k+1)$ del polinomio, si impongono $k+1$ vincoli di continuità per risolvere il sistema le cui incognite sono $a_0, a_1, \dots, a_k$

La fase successiva consiste nel determinare la matrice di rototraslazione

```matlab
%% Ricavo matrice di rototralazione dalla tabella DH
[Rb3,q] = matrixRT(DH,Rb0);
```

Mediante la function *matrixRT.m* che riceve come parametri la tabella di D-H e R_0^b.

```matlab
function [T,q] = matrixRT(DH,Rb0)
    nLink = size(DH,1);
    syms ("q",[1,nLink]);
    T=Rb0;
    for i=1:nLink
        ai = DH(i,1);
        alphai = DH(i,2);
        di = DH(i,3);
        thetai = DH(i,4);

        % Giunti rotoidali
        if symType(thetai) == "variable"
            % SRi -> SRi'
            Rip = [ [rotZ(q(i)), traslZ(di)];
                    0 0 0 1 ];
            % SRi' -> SRi+1
            Ri = [ [rotX(alphai), traslX(ai)];
                   0 0 0 1 ];

        % Giunti prismatici
        elseif symType(di) == "variable"
            % SRi -> SRi'
            Rip = [ [rotZ(thetai), traslZ(q(i))];
                    0 0 0 1 ];
            % SRi' -> SRi+1
            Ri = [ [rotX(alphai), traslX(ai)];
                   0 0 0 1 ];
        end
        R = Rip*Ri;
        T = T*R;
    end
    T = simplify(T);
end
```

Per ogni link dell'antropomorfo costruisco la matrice di rototraslazione che lega SR_i a SR_{i+1}.

- $SR_i \to SR_{i'} \equiv \begin{matrix} rotazione\ di\ \theta_i\ intorno\ a\ z_i \\ traslazione\ di\ d_i\ lungo\ z_i \end{matrix} \Longrightarrow R_{i'}^{i} \equiv R_{ip}$

- $SR_{i'} \to SR_{i+1} \equiv \begin{matrix} rotazione\ di\ \alpha_i\ intorno\ a\ x_{i'} \\ traslazione\ di\ a_i\ lungo\ x_{i'} \end{matrix} \Longrightarrow R_{i+1}^{i'} \equiv R_i$

Ed in base alla tipologia del giunto si ha che:

$$q_i = \begin{matrix} \vartheta_i\ se\ rotoidale \\ d_i\ se\ prismatico \end{matrix}$$

Ottenendo così: $R_{i+1}^{i}(q_i) = R_{i'}^{i} \cdot R_{i+1}^{i'} \equiv R$

Infine, si ha la seguente composizione di trasformazioni:

$$T_3^b(q) = R_0^b \cdot R_1^0(q_1) \cdot R_2^1(q_2) \cdot R_3^2(q_3) \equiv R_3^b$$

Di seguito le function di supporto per effettuare rotazione e traslazione su un asse

matrixRt.m

```
% rotazione attorno z
function Rz = rotZ(angle)
    Rz = [ cos(angle) -sin(angle) 0;
           sin(angle) cos(angle) 0;
           0 0 1 ];
end
```

```
% rotazione attorno x
function Rx = rotX(angle)
    Rx = [ 1 0 0;
           0 cos(angle) -sin(angle);
           0 sin(angle) cos(angle) ];
end
```

```
% traslazione lungo z
function tz = traslZ(d)
    tz = [ 0;
           0;
           d ];
end
```

```
% traslazione lungo x
function tx = traslX(a)
    tx = [ a;
           0;
           0 ];
end
```

In questa parte dello script vi è la costruzione dello Jacobiano Geometrico o Analitico

```
which = input("3. Quale Jacobiano? ");
switch which
    case 'Geometrico'
        J = jacobianGeometric(DH,q);
    case 'Analitico'
        J = jacobianAnalytic(q,Rb3);
    otherwise
        exception = MException('ThisComponent:notFound','Jacobian %s not found',which);
        throw(exception);
end
```

La cinematica differenziale è stata implementata tramite le seguenti function che si occupano di calcolare lo Jacobiano della posizione.

Lo studio del legame tra la velocità dell'end-effector nello spazio di lavoro e la velocità nello spazio dei giunti è esplicito nella matrice Jacobiana così fatta:

$$J = \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \dot{P}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} J_p\big(Q(t)\big) \cdot \dot{Q}(t) \\ J_o\big(Q(t)\big) \cdot \dot{Q}(t) \end{bmatrix} = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$$

Tuttavia, usando la cinematica differenziale inversa per ricavare la velocità nello spazio dei giunti entrambi gli script restituiranno lo Jacobiano della posizione J_p.

- Costruzione dello Jacobiano Analitico

```
function [Ja,sing] = jacobianAnalytic(q,Rb3)
    numQ = length(q);
    Pos = Rb3(1:numQ,4);

    Ja = sym([]);
    for i = 1:numQ
        for j = 1:numQ
            Ja(i,j) = diff(Pos(i),q(j));
        end
    end

    eq = simplify(det(Ja)) == 0;
    sing = solve(eq,q);
end
```

Lo jacobiano analitico si calcola come un'operazione di derivazione dalle equazioni della cinematica diretta.

$$\begin{matrix} p = p(q) \\ \phi = \phi(q) \end{matrix} \rightarrow \begin{matrix} \dot{p} = \frac{\partial q}{\partial q} \cdot \dot{q} = J_p(q) \cdot \dot{q} \\ \dot{\phi} = \frac{\partial \phi}{\partial q} \dot{q} = J_\phi(q) \cdot \dot{q} \end{matrix} \rightarrow \begin{bmatrix} J_p(q) \\ J_\phi(q) \end{bmatrix} \cdot \dot{q}$$

Dove si ha che $J_a(q) = J_p(q)$

- Costruzione dello Jacobiano Geometrico

```
function Jg = jacobianGeometric(DH,q)
    a = double(DH(:,1));
    alpha = DH(:,2);
    d = DH(:,3);
    theta = DH(:,4);

    a2c1c2 = a(2)*cos(q(1))*cos(q(2));
    a2s1c2 = a(2)*sin(q(1))*cos(q(2));
    a2s2 = a(2)*cos(q(2));
    c1 = cos(q(1));
    s1 = sin(q(1));
    a2c2 = a(2)*cos(q(2));
    a3c23 = a(3)*cos(q(2)+q(3));
    a3s23 = a(3)*sin(q(2)+q(3));

    % pi-1 con i=1,2,3
    p0 = [ 0;0;0 ];
    p1 = [ 0;0;0 ];
    p2 = [ a2c1c2;
           a2s1c2;
           a2s2 ];
    % p
    p =[ c1*(a2c2+a3c23);
         s1*(a2c2+a3c23);
         a2s2 + a3s23 ];

    % Asse di rotazione dei giunti
    z0 = [ 0;0;1 ];
    z1 = [ s1;-c1;0 ];
    z2 = z1;
```

Parametri legati al manipolatore Antropomorfo

```
    % Giunti rotoidali
    if symType(theta(1)) == "variable"
        Z0 = Z(z0);
        Z1 = Z(z1);
        Z2 = Z(z2);

        J = [ Z0*(p-p0) Z1*(p-p1) Z2*(p-p2);
              z0 z1 z2 ];
    % Giunti prismatici
    elseif symType(d(1)) == "variable"
        O = [0;0;0];

        J = [ z0 z1 z2;
              O O O ];
    end
    Jg = J(1:3,:);

    eq = simplify(det(Jg)) == 0;
    sing = solve(eq,q);
end
```

$$J = \begin{bmatrix} Jp_i \\ Jo_i \end{bmatrix} = \left\{ \begin{matrix} \begin{bmatrix} z_{i-1}\, x\, (p - p_i) \\ z_{i-1} \end{bmatrix} = \begin{bmatrix} Z_{i-1} \cdot (p - p_i) \\ z_{i-1} \end{bmatrix} \; per\ guinto\ rotoidale \\ \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} \; per\ giunto\ prismatico \end{matrix} \right\}$$

per $i = 1{,}2{,}3$

```
function Zim1 = Z(zim1) % z di i meno 1
    Zim1 = [ 0 -zim1(3) zim1(2);
             zim1(3) 0 -zim1(1);
             -zim1(2) zim1(1) 0 ];
end
```

Funzione per effettuare il prodotto vettoriale come matrice per vettore

Lo Jacobiano è un indicatore delle capacità del robot di muoversi per svolgere il lavoro.

In particolar modo se $rank(J)\ diminuisce \implies singolarità\ cinematiche$ si ha:

a) Perdita di mobilità
b) ∞ soluzioni al problema di cinematica inversa
c) Nell'intorno delle singolarità in cui $\exists q_s\ t.c. \det\big(J(q_s)\big) = 0$ si possono generare velocità elevate nello spazio dei giunti a fronte di velocità ridotte dell'end-effector

Fortunatamente con entrambe le traiettorie non vi sono singolarità, quindi lo Jacobiano risulta essere sempre invertibile.

Infine, vi è la fase di calcolo delle variabili di giunto del manipolatore antropomorfo

```matlab
%% Calcolo delle variabili di giunto
Q = zeros(length(q),numRoute*numStep);
QDot = zeros(length(q),numRoute*numStep);

config = input("4. Gomito Alto o Basso? ");
switch config
    case 'Alto'
        cnfg = 2;
    case 'Basso'
        cnfg = 1;
    otherwise
        exception = MException('ThisComponent:notFound','Config %s not found',config);
        throw(exception);
end

Qstar = [0;0;pi/2];

for i=1:length(position)
    Qstar = cinematicaInversa(position(:,i),DH,Qstar);
    Qstar = double(Qstar(:,cnfg));

    Jq = subs(J,[q(1) q(2) q(3)],Qstar');
    Jq = double(Jq);

    Qdot = pinv(Jq)*velocity(:,i);

    Q(:,i) = Qstar;
    QDot(:,i) = Qdot;
end
```

In input la scelta della configurazione del robot, antropomorfo a gomito alto o a gomito basso

Con la cinematica inversa, dallo spazio di lavoro si passa allo spazio delle variabili di giunto partendo dalla configurazione iniziale

$$q_0 = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{2} \end{bmatrix}$$

Dopo aver sostituito allo Jacobiano le variabili di giunto determinate, si passa al calcolo della derivata delle variabili di giunto

$$\dot{q}(t) = \begin{bmatrix} \dot{q}_1(t) \\ \dot{q}_2(t) \\ \dot{q}_3(t) \end{bmatrix} = J^+(q(t)) \cdot v(t)$$

Per il calcolo della cinematica inversa del manipolatore antropomorfo, si fa riferimento al seguente script:

```matlab
function Q = cinematicaInversa(p,DH,q)
    a = double(DH(:,1));

    px = p(1);
    py = p(2);
    pz = p(3);

    Q1 = [atan2(py,px) atan2(py,px)+pi];

    c3 = (px^2 + py^2 + pz^2 - a(2)^2 - a(3)^2)/(2*a(2)*a(3));
    s3 = sqrt( 1-cos(q(3))^2 ) ;

    Q3 = [atan2(s3,c3) atan2(-s3,c3)];

    s2 = ( ( a(2)+a(3)*cos(q(3)) )*pz - a(3)*sin(q(3))*sqrt(px^2+py^2) )/( px^2 + py^2 + pz^2 );
    c2 = ( ( a(2)+a(3)*cos(q(3)) )*sqrt(px^2+py^2) + a(3)*sin(q(3))*pz )/( px^2 + py^2 + pz^2 );

    Q2 = atan2(s2,c2);

    Q = [ [Q1(1); Q2; Q3(1)],...
          [Q1(1); Q2; Q3(2)],...
          [Q1(2); Q2; Q3(1)],...
          [Q1(2); Q2; Q3(2)] ];
end
```

Obiettivo:

$$p(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} \Rightarrow q(t) = \begin{bmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \end{bmatrix}$$

Nella parte finale dello script vi sono i seguenti plot:

```
%% Plot
fprintf('\n- Plot della traiettoria \n')
plotPosition(numRoute*numStep,P,position)

pause

fprintf('- Plot con Robotics Toolbox \n')
plotAntropomorfo(P,Q,d,a,alpha,position)

pause

fprintf('- Plot andamento delle variabili di giunto \n')
plotQ(Q,QDot,trange,typePath)

pause

fprintf('- Cinematica diretta \n')
plotMyAntropomorfo(Rb0,Rb3,DH,q,Q)
```

È utile focalizzare l'attenzione sull'ultimo script cioè:

```
function plotMyAntropomorfo(Rb0,Rb3,DH,q,Q)
    figure("Name","Cinematica Diretta"); hold on; grid on; axis([-1 2 -1 2 -3 2]);
    xlabel("X"); ylabel("Y"); zlabel("Z");

    Rb2 = matrixRT(DH(1:2,:),Rb0);
    R0b = inv(Rb0);
    for i = 1:size(Q,2)
        T3 = cinematicaDiretta(Rb3,q,Q(:,i));

        Pb3 = [T3(1:3,4); 1];
        P03 = R0b*Pb3; % P03 = inv(Rb0)*Pb3 = R0b*Pb3
        X(i) = P03(1); Y(i) = P03(2); Z(i) = P03(3);

        % Primo braccio, spalla
        Pr1 = [-0.5 -0.5 -1];

        % Secondo braccio
        T2 = cinematicaDiretta(Rb2,q(1:2),Q(1:2,i));

        Pb2 = [T2(1:3,4); 1];
        Pr2 = R0b*Pb2;
        braccio2 = [Pr1; (Pr2(1:3))'];
        b2 = plot3(braccio2(:,1),braccio2(:,2),braccio2(:,3),"k","linewidth",4);

        % Terzo braccio
        braccio3 = [(Pr2(1:3))'; X(i) Y(i) Z(i)];
        b3 = plot3(braccio3(:,1),braccio3(:,2),braccio3(:,3),"k","linewidth",4);

        plot3(X(i),Y(i),Z(i),"ob");
        pause(0.01);
        delete(b2);
        delete(b3);
    end
end
```

Considerando $R_b^0 = (R_0^b)^{-1}$

Si ha che con $R_3^b = \begin{bmatrix} rot_3^b & t_3^b \\ 0 & 1 \end{bmatrix}$, allora per $t_3^b = P_3^b$

si ottiene $P_3^0 = R_b^0 \cdot P_3^b$

Stesso discorso per $R_2^b = \begin{bmatrix} rot_2^b & t_2^b \\ 0 & 1 \end{bmatrix}$, allora per $t_2^b = P_2^b$

si ottiene $P_2^0 = R_b^0 \cdot P_2^b$

Si fa uso della cinematica diretta, per il passaggio dallo spazio dei giunti allo spazio di lavoro del robot

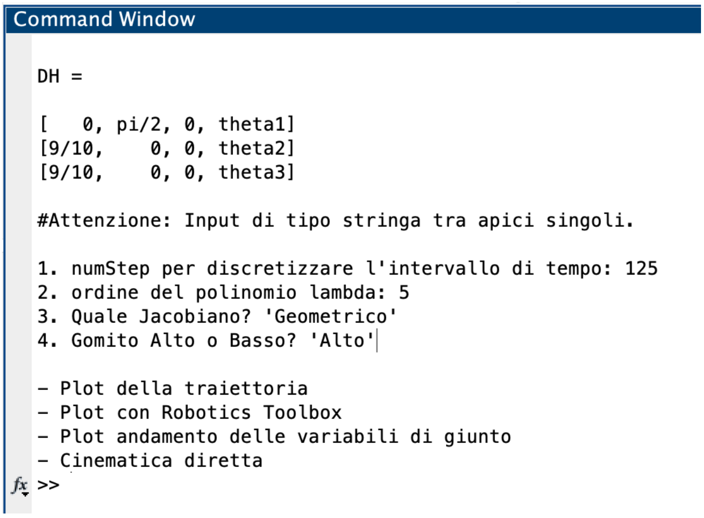
```
function T = cinematicaDiretta(R,q,Q)
    T = double(subs(R,q,Q'));
end
```

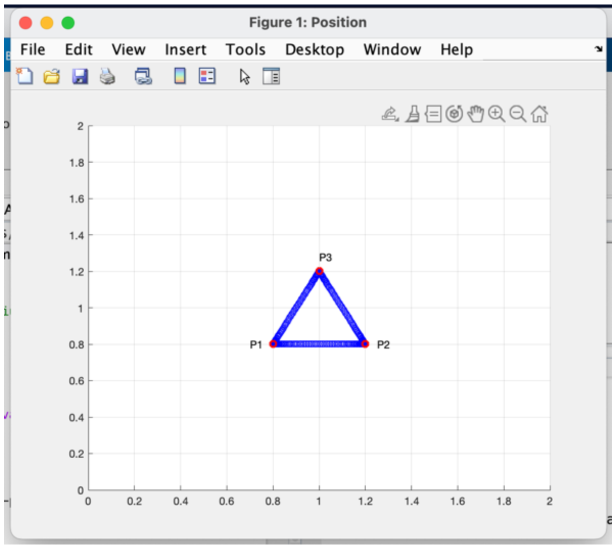
$$q(t) = \begin{bmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \end{bmatrix} \Longrightarrow p(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$$

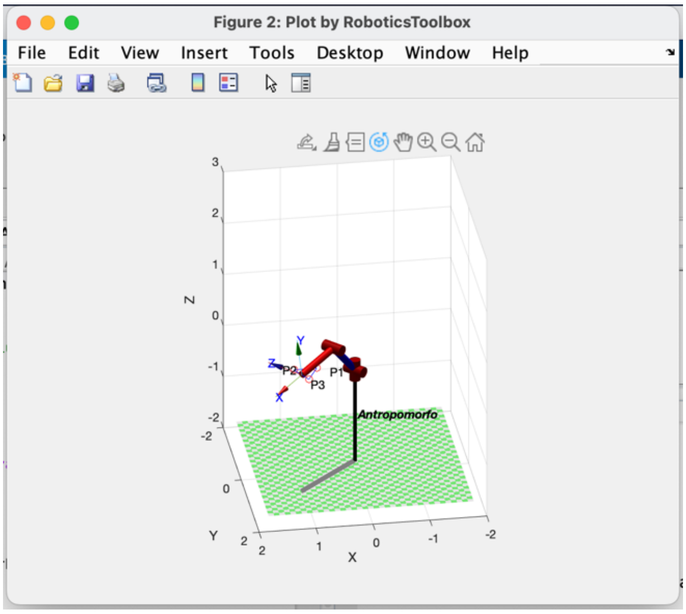
Andando a sostituire nella matrice di rototraslazione le variabili di giunto determinate, per ottenere:

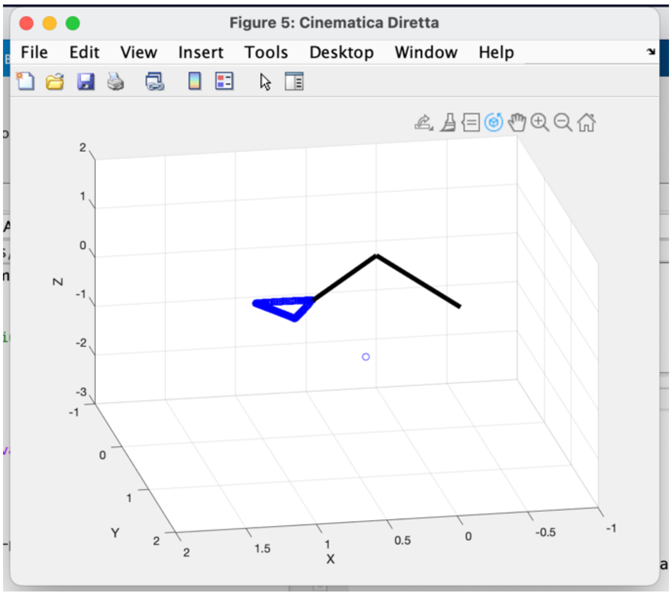
$$R_3^b \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} \; e \; R_2^b \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$$

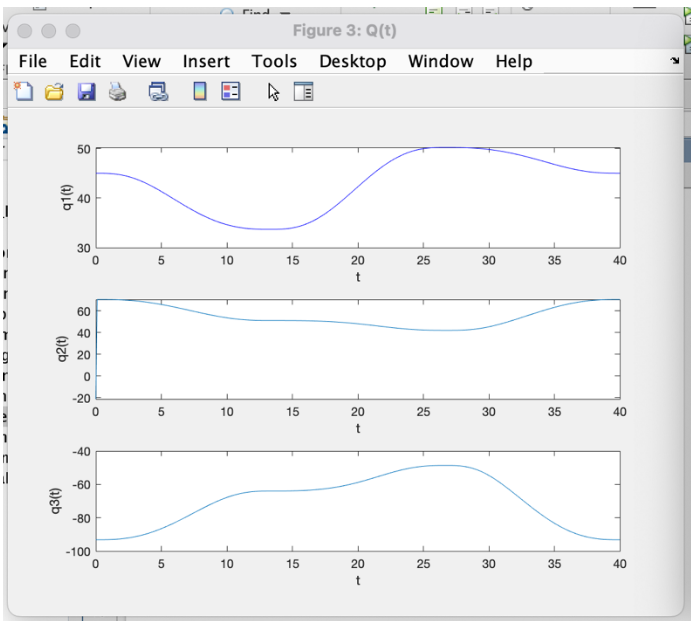
Di seguito un esempio di configurazione dal *command window* e dei plot legati alla traiettoria ‘traingolo’:

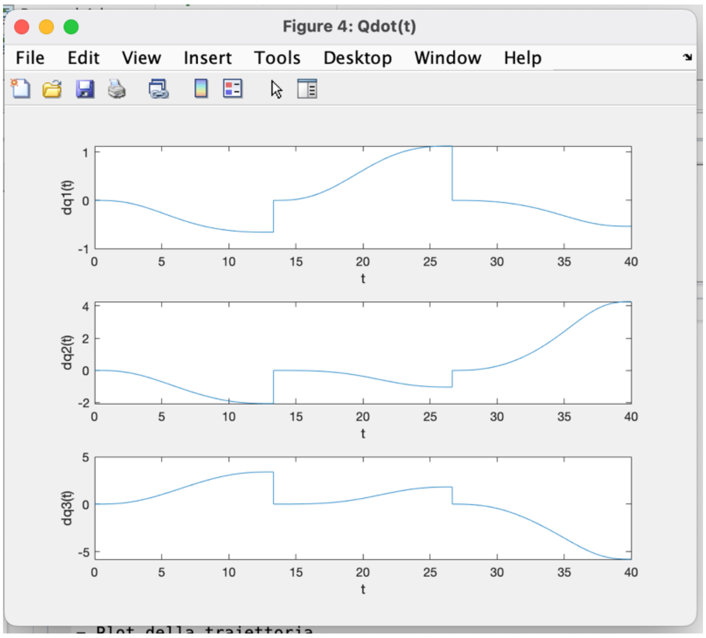












2. Discussione dello script *mainCirconferenza.m*

Sostanzialmente, rispetto allo script trattato in precedenza, cambiano le seguenti fasi:

```
%% 1. Definizione del tempo
numRange = numRoute+1;
Trange = linspace(0,Te,numRange);

numStep = input("\n1. numStep per discretizzare l'intervallo di tempo: ");
trange = linspace(0,Te,numRoute*numStep);
```

Dove si è deciso di costruire due vettori

$$Trange_{1xnumRange} = [0, \dots, Te]$$

in cui $Trange(i)$ corrisponde all'istante di tempo in cui tocco il punto della sequenza $P(i)$.

Mentre

$$trange_{1xnumRoute*numStep} = [0, \dots, t, \dots, Te]$$

In cui ogni elemento del vettore scandisce gli istanti di percorrenza della traiettoria.

```
%% 2. Definizione della traiettoria
degree = input("2. ordine del polinomio lambda: ");
lambdaFun = lambdaFunction(degree);

position = zeros(3,numRoute*numStep); % pos = [ [x;y;z], ... ]
velocity = zeros(3,numRoute*numStep); % vel = [ [vx;vy;vz], ... ]

[xc,yc,zc,r] = circonferenza(P);
Pc = [xc;yc;zc];

phi1 = double(pi+atan((P(1,1)-yc)/(P(2,1)-xc))); % atan(Py-Pcy,Px-Pcx)
phi2 = double(phi1+2*pi);

syms s;
T2=Te;
T1=0;
for i=1:numRoute*numStep
    t = trange(i);
    sigma = (t-T1)/(T2-T1);
    lambda = subs(lambdaFun,s,sigma);
    lambdaDot = subs(lambdaFun,s,sigma);

    Phi = phi1+lambda*(phi2-phi1);
    PhiDot = ((phi2-phi1)/(T2-T1))*lambdaDot;

    position(:,i) = Pc+r*[cos(Phi);sin(Phi);0];
    velocity (:,i) = r*[-sin(Phi);cos(Phi);0] * PhiDot;
end
```

La legge oraria della traiettoria sarà descritta da:

$$P_i = P_c + r \cdot \begin{bmatrix} \cos(\phi) \\ \sin(\phi) \\ 0 \end{bmatrix} \text{ e } v_i = r \cdot \begin{bmatrix} -\sin(\phi) \\ \cos(\phi) \\ 0 \end{bmatrix} \cdot \dot{\phi}$$

Dove ϕ è così costruita:

$$\phi = \phi_1 + \lambda(\sigma(t)) \cdot (\phi_2 - \phi_1)$$

$$\dot{\phi} = \frac{\phi_2 - \phi_1}{T_2 - T_1} \cdot \dot{\lambda}(\sigma(t))$$

Per ricavare le coordinate del centro $P_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$ e il raggio r della circonferenza passante per $P_1, P_2\ e\ P_3$

```
%% Circonferenza
function [xc,yc,zc,r] = circonferenza(P)
    syms x y a b c;
    eq = x^2 + y^2 + a*x + b*y + c;

    eq1 = subs(eq,[x y],[P(1,1) P(2,1)]) == 0;
    eq2 = subs(eq,[x y],[P(1,2) P(2,2)]) == 0;
    eq3 = subs(eq,[x y],[P(1,3) P(2,3)]) == 0;

    [a, b, c] = vpasolve([eq1, eq2, eq3]);
    xc = -a/2;
    yc = -b/2;
    zc = P(3,1); % sullo stesso piano
    r = sqrt(0.25*a^2 + 0.25*b^2 - c);
end
```

Si è risolto il sistema:

$$\begin{cases} P_{1,x}^2 + P_{1,y}^2 + a \cdot P_{1,x} + b \cdot P_{1,y} + c = 0 \\ P_{2,x}^2 + P_{2,y}^2 + a \cdot P_{2,x} + b \cdot P_{2,y} + c = 0 \\ P_{2,x}^2 + P_{2,y}^2 + a \cdot P_{2,x} + b \cdot P_{2,y} + c = 0 \end{cases}$$

Le cui incognite solo i parametri $a, b\ e\ c$

$$P_c = \begin{bmatrix} -\frac{a}{2} \\ -\frac{b}{2} \\ P_{1,z} \end{bmatrix} \text{ e } r = \sqrt{\frac{1}{4} \cdot a^2 + \frac{1}{4} \cdot b^2 - c}$$

Nota bene: i tre punti sono tutti sullo stesso piano

Di seguito un esempio di configurazione dal *command window* e dei plot legati alla traiettoria ‘circonferenza’: