CÓDIGOS

Diseño de control y simulación de un brazo robot para asistencia en laboratorio
Curso 2020-2021

Pablo León Barriga
Pablo Manuel Guzmán Manzanares
Fátima María Fernández Vázquez

Índice

1. ANEXO 2

2. MATLAB Cinemática 2

3. MATLAB Inercias 3

4. MATLAB L R3GDL round 4

5. MATLAB func transf JACO2 17

6. MATLAB controlPD Par Calculado 19

7. controlPID 20

8. MATLAB graf gazebo 22

9. PYTHON mov cont trabajo 24

10. PYTHON generador de controlador 27

1. ANEXO

2. MATLAB Cinemática

```
% Matriz de Denavit-Hartenberg --> Notacion Standard
% clear all
format compact
syms teta d a alfa real;
Aij=MDH(teta,d,a,alfa);
%Aij=trotz(teta)*transl(0,0,d)*transl(a,0,0)*trotx(alfa)
pi1=sym('pi');

%  para obtención del Modelo Cinemático Directo estandar
syms q1 q2 q3 q4 q5 q6 D1 D2 D3 D4 D5 D6 e2 d4b d5b d6b shi aa theta phi
    real;
pi1=sym('pi');

%% Parámetros de Denavit - Hartenberg
A01 = MDH(-q1,      D1,      0,      pi/2);
A12 = MDH(q2+pi/2,  0,       D2,     pi);
A23 = MDH(q3-pi/2,  -e2,     0,      pi/2);
A34 = MDH(q4,       -d4b,    0,      pi/3);
A45 = MDH(q5+pi,    -d5b,    0,      pi/3);
A56 = MDH(q6-pi/2,  -d6b,    0,      pi);

% lo he llamado phi al angulo para que salga mas bonito
T = A01*A12*A23*A34*A45*A56;
px1=T(1,4);
py1=T(2,4);
pz1=T(3,4);

%% Variables auxiliares - DATASHEET PAG 8 - 11
D1v = 0.2755;    D2v = 0.4100;    D3v = 0.2073;    D4v = 0.0741;
D5v = 0.0741;    D6v = 0.1600;    e2v = 0.0098;

aa = pi/6;
ca = cos(aa);        sa = sin(aa);
c2a = cos(2*aa);     s2a = sin(2*aa);

d4bv = D3 + sa/s2a * D4;
d5bv = sa/s2a * D4 + sa/s2a * D5;
d6bv = sa/s2a * D5 + D6;

%% Sustitución
T_sus = subs(T,[D1 D2 D3 D4 D5 D6 e2 d4b d5b d6b],[D1v D2v D3v D4v D5v
   D6v e2v d4bv d5bv d6bv])
```

3. MATLAB Inercias

```
%% Parametros obtenidos del fichero xacro dentro del paquete de ROS:
% https://github.com/Kinovarobotics/kinova-ros/blob/master/
    kinova_description/urdf/kinova_inertial.xacro
clc;clearvars
% Base 0
cdm_link0 = [0 0 0.1255];
masa_link0 = 0.46784;
alt_link0 = 0.14;
radio_link0 = 0.04;

% Shoulder 1
cdm_link1 = [0 -0.002 -0.0605];
masa_link1 = 0.7477;
alt_link1 = 0.14;
radio_link1 = 0.04;

% Arm 2
cdm_link2 = [0 -0.2065 -0.01];
masa_link2 = 0.99;
alt_link2 = 0.4100;
radio_link2 = 0.04;

% Forearm 3
cdm_link3 = [0 0.081 -0.0086];
masa_link3 = 0.6763;
alt_link3 = 0.2073;
radio_link3 = 0.03;

% Wrist 4
cdm_link4 = [0 -0.037 -0.0642];
masa_link4 = 0.426367;
alt_link4 = 0.15;
radio_link4 = 0.04;

%3-finger hand - 5
cdm_link5 = [0 -0.037 -0.0642];
masa_link5 = 0.99;
alt_link5 = 0.16;
radio_link5 = 0.04;

% Matrices con todos los parámetros
cdm = [cdm_link0; cdm_link1; cdm_link2; cdm_link3; cdm_link4; cdm_link5
    ];
masa = [masa_link0; masa_link1; masa_link2; masa_link3; masa_link4;
    masa_link5];
alt = [alt_link0; alt_link1; alt_link2; alt_link3; alt_link4; alt_link5
    ];
```

```
radios = [radio_link0; radio_link1; radio_link2; radio_link3;
    radio_link4; radio_link5];
ejes = [0 0 1 1 0 0];

%% Operaciones, nuevamente sacadas del fichero xacro
Ixx = zeros(1, 6);
Iyy = zeros(1, 6);
Izz = zeros(1, 6);

% Depende del valor incluido en el .xacro para los ejes
for i = 1:6
    if ejes(i) == 0
        Ixx(i) = 0.083333 * masa(i) * (3*radios(i)^2 + alt(i)^2);
        Iyy(i) = 0.083333 * masa(i) * (3*radios(i)^2 + alt(i)^2);
        Izz(i) = 0.5      * masa(i) * radios(i)^2;
    else
        Ixx(i) = 0.083333 * masa(i) * (3*radios(i)^2 + alt(i)^2);
        Iyy(i) = 0.5      * masa(i) * radios(i)^2;
        Izz(i) = 0.083333 * masa(i) * (3*radios(i)^2 + alt(i)^2);
    end
end

% Juntar valores de inercia
I_total = [Ixx; Iyy; Izz];

% Matrices de inercia
matrices_inercia = zeros(18, 3);
for i = 1:6
    for j = 1:3
        % Se almacenan cada matriz de inercia una debajo de otra
        matrices_inercia(3 * (i-1) + j, j) = I_total(j, i);
    end
end
clearvars -except cdm radios alt masa I_total matrices_inercia
```

4. MATLAB L R3GDL round

```
% Ejemplo de la utilización del algoritmo de Lagrange para la dinámica
% de un robot de 3 DGL
%M.G. Ortega (2020)

parametro_inercias_JACO2;

% Elegir entre R (rotación) y P (prismática)
Tipo_Q1 = 'R';
Tipo_Q2 = 'R';
```

```
Tipo_Q3 = 'R';
Tipo_Q4 = 'R';
Tipo_Q5 = 'R';
Tipo_Q6 = 'R';

if ( (Tipo_Q1 ~= 'R') & (Tipo_Q1 ~='P')); error('Elegir R o P para
    Tipo_Q1'); end;
if ( (Tipo_Q2 ~= 'R') & (Tipo_Q2 ~='P')); error('Elegir R o P para
    Tipo_Q2'); end;
if ( (Tipo_Q3 ~= 'R') & (Tipo_Q3 ~='P')); error('Elegir R o P para
    Tipo_Q3'); end;
if ( (Tipo_Q4 ~= 'R') & (Tipo_Q4 ~='P')); error('Elegir R o P para
    Tipo_Q1'); end;
if ( (Tipo_Q5 ~= 'R') & (Tipo_Q5 ~='P')); error('Elegir R o P para
    Tipo_Q2'); end;
if ( (Tipo_Q6 ~= 'R') & (Tipo_Q6 ~='P')); error('Elegir R o P para
    Tipo_Q3'); end;



% Definición de variables simbólicas
syms T1 T2 T3 q1 qd1 qdd1 q2 qd2 qdd2 q3 qd3 qdd3 q4 qd4 qdd4 q5 qd5
    qdd5 q6 qd6 qdd6 g real

%DATOS CINEMÁTICOS DEL BRAZO DEL ROBOT
% Dimensiones (m)
D1 = 0.2755;     D2 = 0.4100;     D3 = 0.2073;     D4 = 0.0741;
D5 = 0.0741;     D6 = 0.1600;     e2 = 0.0098;

aa = pi/6;

ca = cos(aa);          sa = sin(aa);
c2a = cos(2*aa);       s2a = sin(2*aa);

d4b = D3 + sa/s2a * D4;
d5b = sa/s2a * D4 + sa/s2a * D5;
d6b = sa/s2a * D5 + D6;

% Parámetros de Denavit-Hartenberg (utilizado en primera regla de Newton
    -Euler)
% Eslabón 1:
theta1 = -q1;            d1 = D1;          a1 = 0;          alpha1 = pi/2;
% Eslabón 2:
theta2 = q2 + pi/2;      d2 = 0;           a2 = D2;         alpha2 = pi;
% Eslabón 3:
theta3 = q3 - pi/2;      d3 = -e2;         a3 = 0;          alpha3 = pi/2;
% Eslabón 4
theta4 = q4;             d4 = -d4b;        a4 = 0;          alpha4 = pi/3;
% Eslabón 5
```

```
theta5 = q5 + pi;        d5 = -d5b;        a5 = 0;        alpha5 = pi/3;
% Eslabón 6
theta6 = q6 - pi/2;      d6 = -d6b;        a6 = 0;        alpha6 = pi;

%DATOS DINÁMICOS DEL BRAZO DEL ROBOT
% Ejecutar el fichero de inercias

% Eslabón 1
m1 = masa(1); % kg
s11 = cdm(1,:)'; %m
I11 = matrices_inercia(1:3,:); % kg.m2

% Eslabón 2
m2 = masa(2); % kg
s22 = cdm(2,:)'; %m
I22 = matrices_inercia(4:6,:); % kg.m2

% Eslabón 3
m3 = masa(3); % kg
s33 = cdm(3,:)'; %m
I33 = matrices_inercia(7:9,:); % kg.m2

% Eslabón 4
m4 = masa(4); % kg
s44 = cdm(4,:)'; %m
I44 = matrices_inercia(10:12,:); % kg.m2

% Eslabón 5
m5 = masa(5); % kg
s55 = cdm(5,:)'; %m
I55 = matrices_inercia(13:15,:); % kg.m2

% Eslabón 6
m6 = masa(6); % kg
s66 = cdm(6,:)'; %m
I66 = matrices_inercia(16:18,:); % kg.m2


%DATOS DE LOS MOTORES
% Inercias
Jm1 = min(diag(I11)); Jm2 = min(diag(I22)); Jm3 = min(diag(I33)); % kg.
    m2
Jm4 = min(diag(I44)); Jm5 = min(diag(I55)); Jm6 = min(diag(I66)); % kg.
    m2

% Coeficientes de fricción viscosa
Bm1 = 3.6e-5; Bm2 = 3.6e-5; Bm3 = 3.6e-5; %N.m / (rad/s)
Bm4 = 3.6e-5; Bm5 = 3.6e-5; Bm6 = 3.6e-5; %N.m / (rad/s)
% Factores de reducción
```

```
R1 = 1; R2 = 10; R3 = 12;
R4 = 11; R5 = 15; R6 = 16;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ALGORITMO DE CÁLCULO DE LA CINEMÁTICA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% wij : velocidad angular absoluta de eje j expresada en i
% wdij : aceleración angular absoluta de eje j expresada en i
% vij : velocidad lineal absoluta del origen del marco j expresada en i
% vdij : aceleración lineal absoluta del origen del marco j expresada en
    i
% aii : aceleración del centro de gravedad del eslabón i, expresado en i

% fij : fuerza ejercida sobre la articulación j-1 (unión barra j-1 con j
   ),
% expresada en i-1
%
% nij : par ejercido sobre la articulación j-1 (unión barra j-1 con j),
% expresada en i-1

% pii : vector (libre) que une el origen de coordenadas de i-1 con el de
    i,
% expresadas en i : [ai, di*sin(alphai), di*cos(alphai)] (a,d,aplha: par
   ámetros de DH)
%
% sii : coordenadas del centro de masas del eslabón i, expresada en el
   sistema
% i

% Iii : matriz de inercia del eslabón i expresado en un sistema paralelo
    al
% i y con el origen en el centro de masas del eslabón
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Condiciones iniciales de la base
w00=[0 0 0]';
v00 = [0 0 0]';

% Eje Z local
Z=[0 0 1]';

% Eslabón 1
p11 = [a1, d1*round(sin(alpha1),9), d1*round(cos(alpha1),9)]';
% Eslabón 2
p22 = [a2, d2*round(sin(alpha2),9), d2*round(cos(alpha2),9)]';
% Eslabón 3
p33 = [a3, d3*round(sin(alpha3),9), d3*round(cos(alpha3),9)]';
% Eslabón 4
```

```
p44 = [a4, d4*round(sin(alpha4),9), d4*round(cos(alpha4),9)]';
% Eslabón 5
p55 = [a5, d5*round(sin(alpha2),9), d5*round(cos(alpha5),9)]';
% Eslabón 6
p66 = [a6, d6*round(sin(alpha6),9), d6*round(cos(alpha6),9)]';


% Obtención de las matrices de rotación (i)R(i-1) y de sus inversas
R01=simplify([cos(theta1) -round(cos(alpha1),9)*sin(theta1) round(sin(
    alpha1),9)*sin(theta1);
    sin(theta1)  round(cos(alpha1),9)*cos(theta1)  -round(sin(alpha1),9)
        *cos(theta1);
    0            round(sin(alpha1),9)              round(cos(alpha1),9)
                 ]);
R10= R01';

R12=simplify([cos(theta2) -round(cos(alpha2),9)*sin(theta2) round(sin(
    alpha2),9)*sin(theta2);
    sin(theta2)  round(cos(alpha2),9)*cos(theta2)  -round(sin(alpha2),9)
        *cos(theta2);
    0            round(sin(alpha2),9)              round(cos(alpha2),9)
                 ]);
R21= R12';

R23=simplify([cos(theta3) -round(cos(alpha3),9)*sin(theta3) round(sin(
    alpha3),9)*sin(theta3);
    sin(theta3)  round(cos(alpha3),9)*cos(theta3)  -round(sin(alpha3),9)
        *cos(theta3);
    0            round(sin(alpha3),9)              round(cos(alpha3),9)
                 ]);
R32= R23';

R34=simplify([cos(theta4) -round(cos(alpha4),9)*sin(theta4) round(sin(
    alpha4),9)*sin(theta4);
    sin(theta4)  round(cos(alpha4),9)*cos(theta4)  -round(sin(alpha4),9)
        *cos(theta4);
    0            round(sin(alpha4),9)              round(cos(alpha4),9)
                 ]);
R43= R34';

R45=simplify([cos(theta5) -round(cos(alpha5),9)*sin(theta5) round(sin(
    alpha5),9)*sin(theta5);
    sin(theta5)  round(cos(alpha5),9)*cos(theta5)  -round(sin(alpha5),9)
        *cos(theta5);
    0            round(sin(alpha5),9)              round(cos(alpha5),9)
                 ]);
R54= R45';
```

```
R56=simplify([cos(theta6) -round(cos(alpha6),9)*sin(theta6) round(sin(
    alpha6),9)*sin(theta6);
    sin(theta6)  round(cos(alpha6),9)*cos(theta6)  -round(sin(alpha6),9)
        *cos(theta6);
    0            round(sin(alpha6),9)              round(cos(alpha6),9)
                 ]);
R65= R56';
%CÁLCULO DE LA LAGRANGIANA
%L = K - U

%ENERGÍA CINÉTICA
%K = sum_i=1_N Ki
%  donde
%         Ki = 1/2 mi * vcii'*vcii + 1/2* wii' * Ici*wii

%Cálculo de las velocidades angulares
% Articulación 1
if (Tipo_Q1=='R');
    w11= R10*(w00+Z*qd1);   %Si es de rotación
else
    w11 = R10*w00;          %Si es de translación
end
% Articulación 2
if (Tipo_Q2=='R');
    w22= R21*(w11+Z*qd2);   %Si es de rotación
else
    w22 = R21*w11;          %Si es de translación
end
% Articulación 3
if (Tipo_Q3=='R');
    w33= R32*(w22+Z*qd3);   %Si es de rotación
else
    w33 = R32*w22;          %Si es de translación
end
% Articulación 4
if (Tipo_Q4=='R');
    w44= R43*(w33+Z*qd4);   %Si es de rotación
else
    w44 = R43*w33;          %Si es de translación
end
% Articulación 5
if (Tipo_Q5=='R');
    w55= R54*(w44+Z*qd5);   %Si es de rotación
else
    w55 = R54*w44;          %Si es de translación
end
% Articulación 6
if (Tipo_Q6=='R');
    w66= R65*(w55+Z*qd6);   %Si es de rotación
```

```
else
    w66 = R65*w55;       % Si es de translación
end


%Cálculo de las velocidades lineales del origen de los marcos
% Articulación 1
if (Tipo_Q1=='R');
    v11 = R10*v00+cross(w11,p11); % Si es de rotación
else
    v11 = R10*(v00 + qd1*Z) +cross(w11,p11) ; % Si es de translación
end
% Articulación 2
if (Tipo_Q2=='R');
    v22 = R21*v11+cross(w22,p22); % Si es de rotación
else
    v22 = R21*(v11+ qd2*Z)+cross(w22,p22); % Si es de translación
end
% Articulación 3
if (Tipo_Q3=='R');
    v33 = R32*v22+cross(w33,p33); % Si es de rotación
else
    v33 = R32*(v22+qd3*Z)+cross(w33,p33); % Si es de translación
end
% Articulación 4
if (Tipo_Q4=='R');
    v44 = R43*v33+cross(w44,p44); % Si es de rotación
else
    v44 = R43*(v33+qd4*Z)+cross(w44,p44); % Si es de translación
end
% Articulación 5
if (Tipo_Q5=='R');
    v55 = R54*v44+cross(w55,p55); % Si es de rotación
else
    v55 = R54*(v44+qd5*Z)+cross(w55,p55); % Si es de translación
end
% Articulación 6
if (Tipo_Q6=='R');
    v66 = R65*v55+cross(w66,p66); % Si es de rotación
else
    v66 = R65*(v55+qd6*Z)+cross(w66,p66); % Si es de translación
end

%Cálculo de las velocidades lineales de los centros de masas a partir
   de la velocidad lineal del origen del marco.
% Articulación 1
vc11 = v11+cross(w11,s11); %Suponiendo que el centro de gravedad se
   mueva a la misma velocidad que el eslabón
% Articulación 2
```

```
vc22 = v22+cross(w22,s22); %Suponiendo que el centro de gravedad se
    mueva a la misma velocidad que el eslabón
%Articulación 3
vc33 = v33+cross(w33,s33); %Suponiendo que el centro de gravedad se
    mueva a la misma velocidad que el eslabón
%Articulación 4
vc44 = v44+cross(w44,s44); %Suponiendo que el centro de gravedad se
    mueva a la misma velocidad que el eslabón
%Articulación 5
vc55 = v55+cross(w55,s55); %Suponiendo que el centro de gravedad se
    mueva a la misma velocidad que el eslabón
%Articulación 6
vc66 = v66+cross(w66,s66); %Suponiendo que el centro de gravedad se
    mueva a la misma velocidad que el eslabón


%Energía cinética
%K = sum_i=1_N Ki
% donde
%         Ki = 0.5*mi*vcii'*vcii+0.5*wii'*Iii*wii;
K1=0.5*m1*vc11'*vc11+ 0.5* w11'*I11*w11;
K2=0.5*m2*vc22'*vc22+ 0.5* w22'*I22*w22;
K3=0.5*m3*vc33'*vc33+ 0.5* w33'*I33*w33;
K4=0.5*m4*vc44'*vc44+ 0.5* w44'*I44*w44;
K5=0.5*m5*vc55'*vc55+ 0.5* w55'*I55*w55;
K6=0.5*m6*vc66'*vc66+ 0.5* w66'*I66*w66;
K=K1+K2+K3+K4+K5+K6;
%Enecrgía potencial
%U = sum_i=1_N Ui
% donde
%         Ui = mi*g*Zi (suponiendo g en dirección a eje Z fijo)
T01=simplify([cos(theta1) -round(cos(alpha1),9)*sin(theta1) round(sin(
    alpha1),9)*sin(theta1) a1*cos(theta1);
    sin(theta1)  round(cos(alpha1),9)*cos(theta1)  -round(sin(alpha1),9)
        *cos(theta1) a1*sin(theta1);
    0            round(sin(alpha1),9)              round(cos(alpha1),9)
                    d1            ;
    0                  0                    0                        1
                ]);

T12=simplify([cos(theta2) -round(cos(alpha2),9)*sin(theta2) round(sin(
    alpha2),9)*sin(theta2) a2*cos(theta2);
    sin(theta2)  round(cos(alpha2),9)*cos(theta2)  -round(sin(alpha2),9)
        *cos(theta2) a2*sin(theta2);
    0            round(sin(alpha2),9)              round(cos(alpha2),9)
                    d2            ;
    0                  0                    0                        1
                ]);
```

```
T23=simplify([cos(theta3) -round(cos(alpha3),9)*sin(theta3) round(sin(
    alpha3),9)*sin(theta3) a3*cos(theta3);
     sin(theta3)  round(cos(alpha3),9)*cos(theta3)  -round(sin(alpha3),9)
        *cos(theta3) a3*sin(theta3);
     0            round(sin(alpha3),9)              round(cos(alpha3),9)
                     d3             ;
     0                  0                    0                        1
                   ]);
T34=simplify([cos(theta4) -round(cos(alpha4),9)*sin(theta4) round(sin(
    alpha4),9)*sin(theta4) a4*cos(theta4);
     sin(theta4)  round(cos(alpha4),9)*cos(theta4)  -round(sin(alpha4),9)
        *cos(theta4) a4*sin(theta4);
     0            round(sin(alpha4),9)              round(cos(alpha4),9)
                     d4             ;
     0                  0                    0                        1
                   ]);
T45=simplify([cos(theta5) -round(cos(alpha5),9)*sin(theta5) round(sin(
    alpha5),9)*sin(theta5) a5*cos(theta5);
     sin(theta5)  round(cos(alpha5),9)*cos(theta5)  -round(sin(alpha5),9)
        *cos(theta5) a5*sin(theta5);
     0            round(sin(alpha5),9)              round(cos(alpha5),9)
                     d5             ;
     0                  0                    0                        1
                   ]);
T56=simplify([cos(theta6) -round(cos(alpha6),9)*sin(theta6) round(sin(
    alpha6),9)*sin(theta6) a6*cos(theta6);
     sin(theta6)  round(cos(alpha6),9)*cos(theta6)  -round(sin(alpha6),9)
        *cos(theta6) a6*sin(theta6);
     0            round(sin(alpha6),9)              round(cos(alpha6),9)
                     d6             ;
     0                  0                    0                        1
                   ]);



Pc01=T01*[eye(3) s11; [0 0 0] 1];
U1=m1*g*Pc01(3,4);
Pc02=T01*T12*[eye(3) s22; [0 0 0] 1];
U2=m2*g*Pc02(3,4);
Pc03=T01*T12*T23*[eye(3) s33; [0 0 0] 1];
U3=m3*g*Pc03(3,4);
Pc04=T01*T12*T23*T34*[eye(3) s44; [0 0 0] 1];
U4=m4*g*Pc04(3,4);
Pc05=T01*T12*T23*T34*T45*[eye(3) s55; [0 0 0] 1];
U5=m5*g*Pc05(3,4);
Pc06=T01*T12*T23*T34*T45*T56*[eye(3) s66; [0 0 0] 1];
U6=m6*g*Pc06(3,4);
U=U1+U2+U3+U4+U5+U6;
```

```
L=K-U;

%A partir de la Lagrangiana, calcular las ecuaciones
%Ecuación 1: T1 = d/dt(dL/qd1) - dL/q1
dLdq1 = diff(L,qd1);
dLdq1dt = diff(dLdq1,q1)*qd1+diff(dLdq1,q2)*qd2+diff(dLdq1,q3)*qd3+diff(
    dLdq1,qd1)*qdd1+diff(dLdq1,qd2)*qdd2+diff(dLdq1,qd3)*qdd3+diff(dLdq1
    ,q4)*qd4+diff(dLdq1,q5)*qd5+diff(dLdq1,q6)*qd6+diff(dLdq1,qd4)*qdd4+
    diff(dLdq1,qd5)*qdd5+diff(dLdq1,qd6)*qdd6;
dLdq1 = diff(L,q1);
T1 = dLdq1dt - dLdq1;

%Ecuación 2: T2 = d/dt(dL/qd2) - dL/q2
dLdq2 = diff(L,qd2);
dLdq2dt = diff(dLdq2,q1)*qd1+diff(dLdq2,q2)*qd2+diff(dLdq2,q3)*qd3+diff(
    dLdq2,qd1)*qdd1+diff(dLdq2,qd2)*qdd2+diff(dLdq2,qd3)*qdd3+diff(dLdq2
    ,q4)*qd4+diff(dLdq2,q5)*qd5+diff(dLdq2,q6)*qd6+diff(dLdq2,qd4)*qdd4+
    diff(dLdq2,qd5)*qdd5+diff(dLdq2,qd6)*qdd6;
dLdq2 = diff(L,q2);
T2 = dLdq2dt - dLdq2;

%Ecuación3: T3 = d/dt(dL/qd3) - dL/q3
dLdq3 = diff(L,qd3);
dLdq3dt = diff(dLdq3,q1)*qd1+diff(dLdq3,q2)*qd2+diff(dLdq3,q3)*qd3+diff(
    dLdq3,qd1)*qdd1+diff(dLdq3,qd2)*qdd2+diff(dLdq3,qd3)*qdd3+diff(dLdq3
    ,q4)*qd4+diff(dLdq3,q5)*qd5+diff(dLdq3,q6)*qd6+diff(dLdq3,qd4)*qdd4+
    diff(dLdq3,qd5)*qdd5+diff(dLdq3,qd6)*qdd6;
dLdq3 = diff(L,q3);
T3 = dLdq3dt - dLdq3;

%Ecuación4: T4 = d/dt(dL/qd4) - dL/q4
dLdq4 = diff(L,qd4);
dLdq4dt = diff(dLdq4,q1)*qd1+diff(dLdq4,q2)*qd2+diff(dLdq4,q3)*qd3+diff(
    dLdq4,qd1)*qdd1+diff(dLdq4,qd2)*qdd2+diff(dLdq4,qd3)*qdd3+diff(dLdq4
    ,q4)*qd4+diff(dLdq4,q5)*qd5+diff(dLdq4,q6)*qd6+diff(dLdq4,qd4)*qdd4+
    diff(dLdq4,qd5)*qdd5+diff(dLdq4,qd6)*qdd6;
dLdq4 = diff(L,q4);
T4 = dLdq4dt - dLdq4;

%Ecuación4: T5 = d/dt(dL/qd5) - dL/q5
dLdq5 = diff(L,qd5);
dLdq5dt = diff(dLdq5,q1)*qd1+diff(dLdq5,q2)*qd2+diff(dLdq5,q3)*qd3+diff(
    dLdq5,qd1)*qdd1+diff(dLdq5,qd2)*qdd2+diff(dLdq5,qd3)*qdd3+diff(dLdq5
    ,q4)*qd4+diff(dLdq5,q5)*qd5+diff(dLdq5,q6)*qd6+diff(dLdq5,qd4)*qdd4+
    diff(dLdq5,qd5)*qdd5+diff(dLdq5,qd6)*qdd6;
dLdq5 = diff(L,q5);
T5 = dLdq5dt - dLdq5;

%Ecuación4: T6 = d/dt(dL/qd6) - dL/q6
```

```
dLdq6 = diff(L,qd6);
dLdq6dt = diff(dLdq6,q1)*qd1+diff(dLdq6,q2)*qd2+diff(dLdq6,q3)*qd3+diff(
    dLdq6,qd1)*qdd1+diff(dLdq6,qd2)*qdd2+diff(dLdq6,qd3)*qdd3+diff(dLdq6
    ,q4)*qd4+diff(dLdq6,q5)*qd5+diff(dLdq6,q6)*qd6+diff(dLdq6,qd4)*qdd4+
    diff(dLdq6,qd5)*qdd5+diff(dLdq6,qd6)*qdd6;
dLdq6 = diff(L,q6);
T6 = dLdq6dt - dLdq6;
%%

%%%MANIPULACIÓN SIMBÓLICA DE LAS ECUACIONES %%%
%En ecuaciones matriciales (solo parte del brazo):
%
%T= M(q)qdd+V(q,qd)+G(q) = M(q)qdd+VG(q,qd)
%

%Primera ecuación
%--------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M11 = diff(T1,qdd1);
Taux = (T1 - M11*qdd1);
M12 = diff(Taux,qdd2);
Taux = (Taux-M12*qdd2);
M13= diff(Taux,qdd3);
Taux = (Taux-M13*qdd3);
M14= diff(Taux,qdd4);
Taux = (Taux-M14*qdd4);
M15= diff(Taux,qdd5);
Taux = (Taux-M15*qdd5);
M16= diff(Taux,qdd6);
Taux = (Taux-M16*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
G1=diff(Taux,g)*g;
Taux=(Taux-G1);
%Taux restante contiene términos Centrípetos/Coriolis
V1=Taux;

%Segunda ecuación
%--------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M21 = diff(T2,qdd1);
Taux = (T2 - M21*qdd1);
M22 = diff(Taux,qdd2);
Taux = (Taux-M22*qdd2);
M23 = diff(Taux,qdd3);
Taux = (Taux-M23*qdd3);
M24 = diff(Taux,qdd4);
Taux = (Taux-M24*qdd4);
M25 = diff(Taux,qdd5);
```

```
Taux = (Taux-M25*qdd5);
M26 = diff(Taux,qdd6);
Taux = (Taux-M26*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
G2=diff(Taux,g)*g;
Taux=(Taux-G2);
%Taux restante contiene términos Centrípetos/Coriolis
V2=Taux;

%Tercera ecuación
%-------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M31 = diff(T3,qdd1);
Taux = (T3 - M31*qdd1);
M32 = diff(Taux,qdd2);
Taux = (Taux-M32*qdd2);
M33 = diff(Taux,qdd3);
Taux = (Taux-M33*qdd3);
M34 = diff(Taux,qdd4);
Taux = (Taux-M34*qdd4);
M35 = diff(Taux,qdd5);
Taux = (Taux-M35*qdd5);
M36 = diff(Taux,qdd6);
Taux = (Taux-M36*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
G3=diff(Taux,g)*g;
Taux=(Taux-G3);
%Taux restante contiene términos Centrípetos/Coriolis
V3=Taux;

%Cuarta ecuación
%-------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M41 = diff(T4,qdd1);
Taux = (T4 - M41*qdd1);
M42 = diff(Taux,qdd2);
Taux = (Taux-M42*qdd2);
M43 = diff(Taux,qdd3);
Taux = (Taux-M43*qdd3);
M44 = diff(Taux,qdd4);
Taux = (Taux-M44*qdd4);
M45 = diff(Taux,qdd5);
Taux = (Taux-M45*qdd5);
M46 = diff(Taux,qdd6);
Taux = (Taux-M46*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
```

```
G4=diff(Taux,g)*g;
Taux=(Taux-G4);
%Taux restante contiene términos Centrípetos/Coriolis
V4=Taux;

%Quinta ecuación
%--------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M51 = diff(T5,qdd1);
Taux = (T5 - M51*qdd1);
M52 = diff(Taux,qdd2);
Taux = (Taux-M52*qdd2);
M53 = diff(Taux,qdd3);
Taux = (Taux-M53*qdd3);
M54 = diff(Taux,qdd4);
Taux = (Taux-M54*qdd4);
M55 = diff(Taux,qdd5);
Taux = (Taux-M55*qdd5);
M56 = diff(Taux,qdd6);
Taux = (Taux-M56*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
G5=diff(Taux,g)*g;
Taux=(Taux-G5);
%Taux restante contiene términos Centrípetos/Coriolis
V5=Taux;

%Sexta ecuación
%--------------------
%Cálculo de los términos de la matriz de inercia (afines a qdd)
M61 = diff(T6,qdd1);
Taux = (T6 - M61*qdd1);
M62 = diff(Taux,qdd2);
Taux = (Taux-M62*qdd2);
M63 = diff(Taux,qdd3);
Taux = (Taux-M63*qdd3);
M64 = diff(Taux,qdd4);
Taux = (Taux-M64*qdd4);
M65 = diff(Taux,qdd5);
Taux = (Taux-M65*qdd5);
M66 = diff(Taux,qdd6);
Taux = (Taux-M66*qdd6);
%Taux restante contiene términos Centrípetos/Coriolis y Gravitatorios
%Términos gravitatorios: dependen linealmente de "g"
G6=diff(Taux,g)*g;
Taux=(Taux-G6);
%Taux restante contiene términos Centrípetos/Coriolis
V6=Taux;
```

```
% Apilación en matrices y vectores
M = [M11 M12 M13 M14 M15 M16; M21 M22 M23 M24 M25 M26; M31 M32 M33 M34
    M35 M36; M41 M42 M43 M44 M45 M46;M51 M52 M53 M54 M55 M56;M61 M62 M63
     M64 M65 M66];
V = [V1; V2; V3; V4; V5; V6];
G = [G1; G2; G3; G4; G5; G6];


% Inclusión de los motores en la ecuación dinámica
%
%T= Ma(q)qdd+Va(q,qd)+Ga(q)
%
%Ma = M + R^2*Jm      Va=V + R^2*Bm*qd      Ga=G
%
R=diag([R1 R2 R3 R4 R5 R6]);
Jm=diag([Jm1 Jm2 Jm3 Jm4 Jm5 Jm6]);
Bm=diag([Bm1 Bm2 Bm3 Bm4 Bm5 Bm6]);
%Kt=diag([Kt1 Kt2 Kt3]); %No utilizado

Ma=M+R*R*Jm;
Va=V+R*R*Bm*[qd1 ; qd2 ; qd3; qd4; qd5; qd6];
Ga = G;

%La función vpa del Symbolic Toolbox evalua las expresiones de las
%fracciones de una función simbólica, redondeándolas con la precisión
    que podría pasarse como segundo
%argumento.
Ma_lag=vpa(Ma,5);
Va_lag=vpa(Va,5);
Ga_lag=vpa(G,5);

save('ma_va_ga_LAG.mat','Ma','Va','Ga','Ma_lag','Va_lag','Ga_lag');
```

5. MATLAB func transf JACO2

```
%Funciones de transferencia de cada eslabon
clc;clear;

%Se carga el fichero con las matrice Ma, Va, Ga, y se realiza las
%sustituciones necesarias para poder obtener las matrices aproximadas,
    asi
%como las matrices finales. Se calculan también las funciones de
%transferencia de cada eslabón del robot.

%LAS SUSTITUCIONES NECESARIAS SON PARA ESTE CASO EXCLUSIVAMENTE, POR
    TANTO
```

```
%REVISAR SIEMPRE LAS SUSTITUCIONES, CAMBIARLAS MANUALMENTE
clearvars;clc
load('ma_va_ga_LAG.mat');
syms q1 q2 q3 q4 q5 q6 qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4 qdd5
    qdd6

d1=simplify(subs(diag(Ma),[q1 q2 q3 q4 q5 q6],[0 0 0 0 0 0]));

% Eslabon 1
d21=simplify(subs(Va(1),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[1 0 0 0 0 0 0 0 0 0 0 0]));
G1=tf(1,[double(d1(1)) double(d21) 0])

% Eslabon 2
d22=simplify(subs(Va(2),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[0 1 0 0 0 0 0 0 0 0 0 0]));
G2=tf(1,[double(d1(2)) double(d22) 0])

% Eslabon 3
d23=simplify(subs(Va(3),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[0 0 1 0 0 0 0 0 0 0 0 0]));
G3=tf(1,[double(d1(3)) double(d23) 0])

% Eslabon 4
d24=simplify(subs(Va(4),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[0 0 0 1 0 0 0 0 0 0 0 0]));
G4=tf(1,[double(d1(4)) double(d24) 0])

% Eslabon 5
d25=simplify(subs(Va(5),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[0 0 0 0 1 0 0 0 0 0 0 0]));
G5=tf(1,[double(d1(5)) double(d25) 0])

% Eslabon 6
d26=simplify(subs(Va(6),[qd1 qd2 qd3 qd4 qd5 qd6 qdd1 qdd2 qdd3 qdd4
    qdd5 qdd6],[0 0 0 0 0 1 0 0 0 0 0 0]));
G6=tf(1,[double(d1(6)) double(d26) 0])

Ma_aprox = double(diag(d1))
Va_aprox = double([d21;d22;d23;d24;d25;d26])

% clearvars -except G1 G2 G3 G4 G5 G6 Ma_aprox Va_aprox
% save('JACO_2_funcionesTransferencia.mat');
```

6. MATLAB controlPD Par Calculado

```
% Archivo para calculos de diseño frecuencial de controladores. CALCULO
    DE
%PD MEDIANTE TECNICAS FRECUENCIALES (BODE).

%ES NECESARIO TENER LAS FUNCIONES DE TRANSFERENCIA DE CADA ESLABÓN, ASÍ
%COMO LAS ESPECIFICACIONES DEL CONTROL (wc, Mfdes).

%CAMBIAR wc A LA QUE SE NECESITE. LAS VARIABLES MFDESEADO, MFACTUAL, Y
%GANANCIA ACTUAL SE PIDEN POR TECLADO EN LA VENTANA DE COMANDOS DE
    MATLAB.

%CAMBIAR TAMBIEN EL ALCANCE DEL BODE DENTRO DE LOGSPACE(X,Y,N PUNTOS),
    CON
%X TAL QUE 10^X, Y TAL QUE 10^Y

clear all;close all;

G=tf(1,[1 0 0]) %Funcion de transferencia para calcular el

numC=1;
denC=1;

C=tf(numC,denC);      %Se genera ft

bode(C*G,logspace(0,3,100));grid;    %Bode Gba con C=1

prompt11 = 'Introduzca el margen de fase deseado: ';
Mfdes = input(prompt11);      %Se lee la ventana de comandos

wc = 104;                     %Frecuencia de corte deseada, constante en
    los 3 controladores

%Introducir a continuación el valor de Mf actual
prompt1 = 'Introduzca la fase actual: ';
fase = input(prompt1);        %Se lee la fase actual, vista en el bode
    anteriormente dibujado

Mfact = 180 + fase;
angdes = Mfdes - Mfact;
angdesrad = angdes * pi/180;

%Calculo de tau (en radianes)
tau = tan(angdesrad)/ wc

%Diseño del controlador PD
C = tf([tau 1],1)
```

```
bode(C*G,logspace(0,3,100));grid;    %Se dibuja de nuevo Gba, ahora con
    el controlador calculado arriba

%Una vez calculado tau, se pasa a ajustar la ganancia.
prompt2 = 'Introducir la ganancia para wc: ';
g = input(prompt2); %Se lee la ganancia
kp = 10 ^ (-g/20)    %Ajuste de ganancia

%Una vez se tiene tau y kp, se crea el controlador en funcion de la
%necesidad de tener un PI o un PID
taf = 1/(10*wc);
C = tf(kp*[tau 1],1)

bode(C*G,logspace(0,3,100));grid;    %Se dibuja Gba, comprobando que se
    cumple el diseño de Mfdes y wc
```

7. controlPID

```
%Archivo para calculos de diseño frecuencial de controladores. CALCULO
    DE
%PD MEDIANTE TECNICAS FRECUENCIALES (BODE). ES NECESARIO TENER LAS
%FUNCIONES DE TRANSFERENCIA DE CADA ESLABÓN, ASÍ COMO LAS
    ESPECIFICACIONES
%DEL CONTROL (wc, Mfdes). CAMBIAR wc A LA QUE SE NECESITE. LAS
    VARIABLES
%MFDESEADO, MFACTUAL, Y GANANCIA ACTUAL SE PIDEN POR TECLADO EN LA
    VENTANA
%DE COMANDOS DE MATLAB

clear all;clc;

PI = 0;                    %Variable para indicar si se necesita un PI
    (=1) o un PID (=0)
ganan=[];                  %Se almacenarán las ganancias

load('JACO_2_funcionesTransferencia.mat'); %Hay que cargar la funcion
    de transferencia

for i=1:3

    if(i==1)
        G=G1;
    elseif(i==2)
        G=G2;
    elseif(i==3)
        G=G3;
```

```
elseif(i==4)
    G=G4;
elseif(i==5)
    G=G5;
else
    G=G6;
end

numC=1;
denC=1;

C=tf(numC,denC);      %Se genera ft

bode(C*G,logspace(0,3,100));grid;    %Bode Gba con C=1

prompt11 = 'Introduzca el margen de fase deseado: ';
Mfdes = input(prompt11);      %Se lee la ventana de comandos

wc = 104;                     %Frecuencia de corte deseada, constante
    en los 3 controladores

%Introducir a continuación el valor de Mf actual
prompt1 = 'Introduzca la fase actual: ';
fase = input(prompt1);        %Se lee la fase actual, vista en el
   bode anteriormente dibujado

Mfact = 180 + fase;
angdes = Mfdes - Mfact + 5;
angdesrad = angdes * pi/180;

if (angdes<=-35)
    mensaje=sprintf("Se necesita un PI")
    PI = 1;
else
    mensaje=sprintf("Se necesita un PID")
    PI = 0;
end

if (angdes>90)
    mensaje=sprintf("El angulo deseado es superior a 90, disminuir
        Mfdes")
end

%Calculo de tau (en radianes)
if (PI == 0)
    tau = tan((angdesrad + pi/2)/2) / wc
else
    tau = tan(angdesrad + pi/2)/ wc
end
```

```
    %Diseño del controlador en funcion de la necesidad de PI o PID
    if (PI == 1)
        C = tf([tau 1],[tau 0]);
    else
        C = tf(conv([tau 1],[tau 1]),[tau 0]);
    end

    bode(C*G,logspace(0,3,100));grid;    %Se dibuja de nuevo Gba, ahora
        con el controlador calculado arriba

    %Una vez calculado tau, se pasa a ajustar la ganancia.
    prompt2 = 'Introducir la ganancia para wc: ';
    g = input(prompt2); %Se lee la ganancia
    kp = 10 ^ (-g/20)    %Ajuste de ganancia

    %Una vez se tiene tau y kp, se crea el controlador en funcion de la
    %necesidad de tener un PI o un PID
    taf = 1/(10*wc);
    if (PI == 1)
        C = tf(kp*[tau 1],conv([taf 1],[tau 0]))
    else
        C = tf(kp*conv([tau 1],[tau 1]),conv([tau 0],[taf 1]))
    end



    bode(C*G,logspace(0,3,100));grid;    %Se dibuja Gba, comprobando que
         se cumple el diseño de Mfdes y wc
    ganan=[ganan;kp];    %Almacenamiento de la k
end
kG1 = ganan(1);
kG2 = ganan(2);
kG3 = ganan(3);
kG4 = ganan(4);
kG5 = ganan(5);
kG6 = ganan(6);
%save('kG.mat','kG1','kG2','kG3','kG4','kG5','kG6');     %Guardamos el
    fichero mat para tenerlo
```

8. MATLAB graf gazebo

```
clearvars; clc; close all
%"aaa" es el nombre con el que se han almacenado dentro de los .mat
    cada
%uno de los datos correspondientes a cada controlador
```

```
%Se cargan los datos
%load('datosSimGazebo_controlPAR.mat');
load('datosSimGazebo_controlPID.mat');
%load('datosSimGazebo_controlEstandar.mat');

%Offset de los datos
inicio = 2;

%Pasar de nanosegundos (valor del tiempo en el .bag obtenido de ros) a
%segundos
nanosec_to_seg = 1e-9;

%Escalar tiempo a segundos
aaa(2:end,1) = aaa(2:end,1)*nanosec_to_seg;

%Obtener el tiempo en que se empieza a tomar datos
time_offset = aaa(inicio,1);

%Hacer que ese sea el 0
aaa(2:end,1) = aaa(2:end,1) - time_offset;

%Coger valores
time = aaa(2:end,1);
q1ref = aaa(2:end,2);
q2ref = aaa(2:end,3);
q3ref = aaa(2:end,4);
q4ref = aaa(2:end,5);
q5ref = aaa(2:end,6);
q6ref = aaa(2:end,7);
q1 = aaa(2:end,8);
q2 = aaa(2:end,9);
q3 = aaa(2:end,10);
q4 = aaa(2:end,11);
q5 = aaa(2:end,12);
q6 = aaa(2:end,13);

qref = [q1ref q2ref q3ref q4ref q5ref q6ref];
q = [q1 q2 q3 q4 q5 q6];

%Graficar
for i = 1:6
    figure(i);plot(time,qref(:,i),time,q(:,i));
    grid;
    legend('Referencia', 'Real');
%     titulo = sprintf("Control par computado - articulacion %d", i);
    titulo = sprintf("Control PID - articulacion %d", i);
%     titulo = sprintf("Control estándar - articulacion %d", i);
    xlabel('Tiempo (s)');
```

```
    ylabel('Posición angular (rad)');
    title(titulo);
end

set(gcf,'color','w');
```

9. PYTHON mov cont trabajo

```
#! /usr/bin/env python
import rospy
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
from sensor_msgs.msg import JointState
from std_srvs.srv import Empty
import argparse
import time
import numpy as np

def actual_position(msj):
  global posicion_actual
  posicion_actual=np.asarray(msj.position)[0:6];


def argumentParser(argument):
  global bot
  parser = argparse.ArgumentParser(description='Script de movimiento de
      KinovaRos para el robot de asistencia en laboratorio')
  parser.add_argument('kinova_robotType', metavar='kinova_robotType',
      type=str, default='j2n6a300',
                      help='j2n6s300, el control y posicion esta hecho
                          especificamente para este tipo de robot')
  argv = rospy.myargv()
  args_ = parser.parse_args(argv[1:])
  bot = args_.kinova_robotType

def moveJoint (PoseFinal):
  pub = rospy.Publisher('/' + bot + '/effort_joint_trajectory_controller
      /command', JointTrajectory, queue_size=1)
  sub = rospy.Subscriber('/' + bot + '/joint_states',JointState,
      actual_position)
  ComandArticulacion = JointTrajectory()
  PuntInterm = JointTrajectoryPoint()
  ComandArticulacion.header.stamp = rospy.Time.now() + rospy.Duration.
      from_sec(0.0);
  PuntInterm.time_from_start = rospy.Duration.from_sec(5.0)
  for i in range(0,6):
```

```
    ComandArticulacion.joint_names.append(bot +'_joint_'+str(i+1))
    PuntInterm.positions.append(PoseFinal[i])
    PuntInterm.velocities.append(0)
    PuntInterm.accelerations.append(0)
    PuntInterm.effort.append(0)
  ComandArticulacion.points.append(PuntInterm)
  rate = rospy.Rate(100)
  aux = 0
  while (aux < 50):
    pub.publish(ComandArticulacion)
    aux = aux + 1
    rate.sleep()

def moveFingers (PoseFinal):
  pub = rospy.Publisher('/' + bot + '/
      effort_finger_trajectory_controller/command', JointTrajectory,
      queue_size=1)
  sub = rospy.Subscriber('/' + bot + '/joint_states',JointState,
      actual_position)
  ComandArticulacion = JointTrajectory()
  PuntInterm = JointTrajectoryPoint()
  ComandArticulacion.header.stamp = rospy.Time.now() + rospy.Duration.
      from_sec(0.0);
  PuntInterm.time_from_start = rospy.Duration.from_sec(5.0)
  for i in range(0,3):
    ComandArticulacion.joint_names.append(bot +'_joint_finger_'+str(i+1)
        )
    PuntInterm.positions.append(PoseFinal[i])
    PuntInterm.velocities.append(0)
    PuntInterm.accelerations.append(0)
    PuntInterm.effort.append(0)
  ComandArticulacion.points.append(PuntInterm)
  rate = rospy.Rate(100)
  aux = 0
  while (aux < 500):
    pub.publish(ComandArticulacion)
    aux = aux + 1
    rate.sleep()

def esperarPosicion(posicionFinal,numeroPosicion):
    err=abs(posicionFinal-posicion_actual)
    compensacion=[0,0,0,0,0,0]
    print err
    print np.any(err[0:3] > 0.2)
    while np.any(err[0:3] > 0.2):
        err=abs(posicion_actual-posicionFinal)-compensacion
        for x in range(3):
            if err[x] >= 2*3.14:
                compensacion[x] = compensacion[x] + 2*3.14
```

```
            elif err[x]<0:
                compensacion[x]=0
        print "esperando posicion " + str(numeroPosicion)
        print err[0:3]

if __name__ == '__main__':
  try:
    rospy.init_node('move_robot_using_trajectory_msg')
    argumentParser(None)
    #allow gazebo to launch
    time.sleep(5)

    # Unpause the physics
    rospy.wait_for_service('/gazebo/unpause_physics')
    unpause_gazebo = rospy.ServiceProxy('/gazebo/unpause_physics', Empty
        )
    resp = unpause_gazebo()
    posHome=[2.5,4,3,4.7,3.14,0.0]
    posRecogida=[2.5,4.5,3.4,4.7,3.14,0.0]
    posAlmacenamiento=np.array
        (([-0.64,3.7,2.7,4.7,3.14,0.0],[-0.84,3.7,2.7,4.7,3.14,0.0],
    [-1.04,3.7,2.7,4.7,3.14,0.0],[-1.24,3.7,2.7,4.7,3.14,0.0],
    [-1.34,3.7,2.7,4.7,3.14,0.0],[-1.54,3.7,2.7,4.7,3.14,0.0]))
    abrir=[0,0,0]
    cerrar=[1,1,1]
    moveJoint (posHome);
    esperarPosicion(posHome,0)
    moveFingers(abrir)
    for k in range(1,6,1):
        #home robots
        posQuerida=posRecogida
        moveJoint (posQuerida);
        esperarPosicion(posQuerida,k)
        moveFingers(cerrar)
        print "position " + str(k)

        posQuerida=posAlmacenamiento[k-1]
        moveJoint (posQuerida);
        esperarPosicion(posQuerida,k)
        moveFingers(abrir)
        print "position " + str(k)

        posQuerida=posHome
        moveJoint (posQuerida);
        esperarPosicion(posQuerida,k)
        print "position " + str(k)

    else:
      moveJoint ([0.0,2.9,0.0,1.3,4.2,1.4,0.0])
```

```
    moveFingers ([1,1,1])
  except rospy.ROSInterruptException:
    print "el programa se ha detenido por razones externas"
```

10. PYTHON generador de controlador

```
import yaml

robot_name = 'j2n6s300'
test_text = int(input ("Introduzca el controlador deseado(0 para Fabrica
    ,1 para controlador calculado n1 (par computado), 2 para
    controlador calcualdo n2 (PID)): "))
if test_text==0:
        joint_p = [5000,5000,5000,500,200,500,500]
        joint_i = [0,0,0,0.,0,0,0]
        joint_d = [0,0,0,0,0,0,0]
        print "0 seleccionado"
elif test_text==1:
        joint_p = [2818,2818,2818,2818,2818,2818,2818]
        joint_i = [0,0,0,0,0,0,0]
        joint_d = [101.38,101.38,101.38,101.38,101.38,101.38,101.38]
        print "1 seleccionado"
elif test_text==2:
        joint_p = [146.6,2924.4,720.3,79.6,202.3,384.6,0]
        joint_i = [0.055,0.055,0.055,0.055,0.055,0.055,0.055]
        joint_d = [0.2198,0.2198,0.2198,0.2198,0.2198,0.2198,0.2198]
        print "2 seleccionado"
else:
        test_text = int(input ("No te he entendido, repitelo!(0, 1, 2): "
            "))

finger_p = [10,10,10]
finger_i = [0,0,0]
finger_d = [0,0,0]
dof = int(robot_name[3])
fingers = int(robot_name[5])
robot_joints = []
finger_joints = []


for i in range(1,dof+1):
   robot_joints.append(robot_name + '_joint_' + str(i))

for i in range(1,fingers+1):
   finger_joints.append(robot_name + '_joint_finger_' + str(i))
```

```
  ############################## Joint state controller
joint_state_controller = {'joint_state_controller':{'type':'
    joint_state_controller/JointStateController', 'publish_rate' : 500}}
robot_controllers = joint_state_controller

  ############################## trajectory controllers

  ######## effort
  #joints
joints =[]
gains = {}
constraints = {}
i = 0
for joint in robot_joints:
    joints.append(joint)
    gains.update({joint:
                          {'p': joint_p[i], 'i': joint_i[i], 'd':
                              joint_d[i], 'i_clamp': 10}
                      })
    constraints.update({joint:
                         {
                          'trajectory':0.05,
                          'goal': 0.02
                         }
                        })
    i = i + 1
joints = {'joints': joints}
gains = {'gains': gains}
constraints_dic = {  'goal_time': 1.0,
                         'stopped_velocity_tolerance': 0.02}

constraints_dic.update(constraints)
constraints_dic = {'constraints': constraints_dic}
robot_trajectory_position_controller = {'type':'effort_controllers/
    JointTrajectoryController'}
robot_trajectory_position_controller.update(joints)
robot_trajectory_position_controller.update(gains)
robot_trajectory_position_controller.update(constraints_dic)
robot_trajectory_position_controller_dic = {'
    effort_joint_trajectory_controller':
   robot_trajectory_position_controller}

robot_controllers.update(robot_trajectory_position_controller_dic)

  #fingers
joints =[]
gains = {}
constraints = {}
```

```
i=0
for joint in finger_joints:
    joints.append(joint)
    gains.update({joint:
                          {'p': finger_p[i], 'i': finger_i[i], 'd':
                              finger_d[i], 'i_clamp': 1}
                      })
    constraints.update({joint:
                         {
                          'trajectory':0.05,
                          'goal': 0.02
                         }
                        })
    i = i + 1
joints = {'joints': joints}
gains = {'gains': gains}
constraints_dic = {  'goal_time': 1.0,
                         'stopped_velocity_tolerance': 0.02}

constraints_dic.update(constraints)
constraints_dic = {'constraints': constraints_dic}
robot_trajectory_position_controller = {'type':'effort_controllers/
   JointTrajectoryController'}
robot_trajectory_position_controller.update(joints)
robot_trajectory_position_controller.update(gains)
robot_trajectory_position_controller.update(constraints_dic)
robot_trajectory_position_controller_dic = {'
   effort_finger_trajectory_controller':
  robot_trajectory_position_controller}

robot_controllers.update(robot_trajectory_position_controller_dic)

  ############################## Joint by joint controllers

  #####position
  #add joint position controller
i = 0
for joint in robot_joints:
    robot_joint_position_controller =  { 'joint_' + str(i+1) + '
       _position_controller':
               {
               'type':'effort_controllers/JointPositionController',
               'joint':joint,
               'pid': {'p': joint_p[i], 'i': joint_i[i], 'd': joint_d[i
                  ]}
               }
        }
    robot_controllers.update(robot_joint_position_controller)
    i = i + 1
```

```
  #add finger joint position controllers
i = 0
for joint in finger_joints:
    finger_joint_position_controller =  { 'finger_' + str(i+1) + '
        _position_controller':
                {
                'type':'effort_controllers/JointPositionController',
                'joint':joint,
                'pid': {'p': finger_p[i], 'i': finger_i[i], 'd':
                    finger_d[i]}
                }
                                                        }
    i = i + 1
    robot_controllers.update(finger_joint_position_controller)

config = {robot_name: robot_controllers}

with open(robot_name +'_control.yaml', 'w') as outfile:
      yaml.dump(config, outfile, default_flow_style=False)
```