SIMULADOR DE UN ROBOT SCARA DE 4 GRADOS DE LIBERTAD BASADO EN REALIDAD VIRTUAL

ANEXOS

DIANA CAROLINA LUNA DIAGO
DIEGO ARMANDO CHECA ROJAS

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
LÍNEA DE INVESTIGACIÓN EN ROBÓTICA INDUSTRIAL
POPAYÁN
2007

SIMULADOR DE UN ROBOT SCARA DE 4 GRADOS DE LIBERTAD BASADO EN REALIDAD VIRTUAL

ANEXOS

Diana Carolina Luna Diago
Diego Armando Checa Rojas

Monografía Para Optar al Titulo de
Ingeniero en Automática Industrial

Director
Víctor Hugo Mosquera Leyton
Especialista en Informática Industrial
Ingeniero en Electrónica y Telecomunicaciones

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRONICA INSTRUMENTACION Y CONTROL
LINEA DE INVESTIGACION EN ROBOTICA INDUSTRIAL
POPAYAN
2007

TABLA DE CONTENIDO

LISTA DE FIGURAS

ANEXO A

CÓDIGO DE SIMULACIÓN EN MATLAB

A.1. Archivo de inicio del controlador PID

```
clear
clc

%lineal;
%Circular;
Unalinea;
Temp=0.001;
QI=[1.3979;-1.8930;3.6367;0.4000];        % Trayectoria lineal
%QI=[1.3534;-1.5828;3.3710;0.4000];       % Trayectoria circular

% Variables de Matlab
kp1=210000;
kp2=80000;
kp3=40000;
kp4=70000;

kv1=1500;
kv2=200;
kv3=60;
kv4=500;

ki1=1000;
ki2=1000;
ki3=1000;
ki4=1000;
```

A.2. Archivo de inicio del controlador CTC

```
clc;
clear all;
clear
Tem=0.0001;
```

```
%Circular;
%Ql=[1.3534;-1.5828;3.3710;0.4000];        % Trayectoria circular

Unalinea;
Ql=[1.3979;-1.8930;3.6367;0.4000];         % Trayectoria lineal

%proporcional
kp1=39000;
kp2=39000;
kp3=39000;
kp4=39000;

%derivativa
kv1=50;
kv2=50;
kv3=50;
kv4=50;

%integral
ki1=0;
ki2=0;
ki3=0;
ki4=0;
```

A.3 Trayectoria lineal

```
%------------Movimiento lineal------------
%-----------------------------------------------

Tfinal=1.0;
Tem=0.001;

% Cálculo del número de muestras %

nbech=(Tfinal/Tem)+1;
if ((round(nbech)-nbech) == 0)
        instant=[0:Tem:Tfinal]';
else
        nbech=nbech+1;
        instant=[0:Tem:Tfinal+Tem]';
end
```

```
instant = instant(1:1000,:);
t=0;
for h=1:1:1000
  t=t+Tem;
  x1(h)=t;
  y1(h)=t;
end

xx = x1';
yy = y1';

cons1= 0.35 + 0.1*xx;
cons2= 0.35 + 0.1*yy;
cons3 = 0.4*ones(1000,1);
```

A.4. Trayectoria Circular

```
%------------Movimiento circular------------
%-------------------------------------------------

Tfinal=3.0;
Tem=0.001;

% Cálculo del número de muestras %

nbech=(Tfinal/Tem)+1;
if ((round(nbech)-nbech) == 0)
        instant=[0:Tem:Tfinal]';
else
        nbech=nbech+1;
        instant=[0:Tem:Tfinal+Tem]';
end
t=0;

for h=1:1:nbech
  t=t+Tem;
  x1(h)=0.02*sin(2.0943951*t);% - 1.266);%2*pi/5 = 1.2566
  y1(h)=0.02*cos(2.0943951*t);% - 1.266);
end

x1=x1';
```

```
y1=y1';

cons1= 0.4 + x1;
cons2= 0.4 + y1;
cons3=0.5*ones(3001,1);
```

A.5 Modelo geométrico inverso

```
function q = mgi(x1,x2,x3)

% Modelo geométrico inverso del SCARA de cuatro ejes:

d2 = 0.5;
d3 = 0.3;

C2 = (x1^2 + x2^2 - d2^2 - d3^2)/(2*d2*d3);
B1 = d2 + d3*C2;

q2 = atan2((-sqrt(1 - C2^2)),C2);

B2 = d3*sin(q2);

S1 = (B1*x2 - B2*x1)/(B1^2 + B2^2);
C1 = (B1*x1 + B2*x2)/(B1^2 + B2^2);

q1 = atan2(S1,C1);
q3 = atan2(0,-1) - q2 - q1;% sy, sx
q4 = x3;

q = [q1 q2 q3 q4];
```

A.6 Modelo dinámico inverso

```
function GAM =
scara_inverso(position1,position2,position3,position4,vitesse1,vitesse2,vitesse3,vit
esse4,w1,w2,w3,w4)

G3=9.81;
M4=1.8;
d2=0.5;
d3=0.3;%
```

```
ZZ1R=3.38;
ZZ2R=0.063;
ZZ3R=0.1;%
MX2R=2*0.121;
MX3R=0.2;%
MY3R=0.1;
MY2=0.001;
FV1=0.1;
FV2=0.012;
FV3=FV1;
FV4=FV2;
FS1=0.57;
FS2=0.125;
FS3=FS1;
FS4=FS2;
IA2=0;
CZ1=0;
FX2=0;
FY2=0;
FX3=0;
FY3=0;
FZ3=0;
CZ3=0;
CZ4=0;
FX4=0;
FY4=0;
FZ4=0;
IA3=0;
IA4=0.045;

%Errores en el modelo:
%aa=1.35;
%M4=M4*aa;
%ZZ1R=ZZ1R*aa;
%ZZ2R=ZZ2R*aa;
%ZZ3R=ZZ3R*aa;
%MX2R=MX2R*aa;
%MX3R=MX3R*aa;
%MY3R=MY3R*aa;
%MY2=MY2*aa;
%FV1=FV1*aa;
```

```
%FV2=FV2*aa;
%FV3=FV1;
%FV4=FV2;
%FS1=FS1*aa;
%FS2=FS2*aa;
%FS3=FS1;
%FS4=FS2;
%IA4=IA4*aa;

t2=position2;
t3=position3;
QP1=vitesse1;
QP2=vitesse2;
QP3=vitesse3;
QP4=vitesse4;
QDP1=w1;
QDP2=w2;
QDP3=w3;
QDP4=w4;

S2=sin(t2);
C2=cos(t2);
S3=sin(t3);
C3=cos(t3);
DV331=-QP1.^2;
No31=QDP1.*ZZ1R;
W32=QP1 + QP2;
WP32=QDP1 + QDP2;
DV332=-W32.^2;
VSP12=d2.*DV331;
VSP22=d2.*QDP1;
VP12=C2.*VSP12 + S2.*VSP22;
VP22=-(S2.*VSP12) + C2.*VSP22;
F12=DV332.*MX2R - MY2.*WP32;
F22=DV332.*MY2 + MX2R.*WP32;
No32=WP32.*ZZ2R;
W33=QP3 + W32;
WP33=QDP3 + WP32;
DV333=-W33.^2;
VSP13=d3.*DV332 + VP12;
VSP23=VP22 + d3.*WP32;
```

```
VP13=C3.*VSP13 + S3.*VSP23;
VP23=-(S3.*VSP13) + C3.*VSP23;
F13=DV333.*MX3R - MY3R.*WP33;
F23=DV333.*MY3R + MX3R.*WP33;
No33=WP33.*ZZ3R;
VP34=-G3 + QDP4;
F14=M4.*VP13;
F24=M4.*VP23;
F34=M4.*VP34;
E14=F14 + FX4;
E24=F24 + FY4;
E34=F34 + FZ4;
E13=E14 + F13 + FX3;
E23=E24 + F23 + FY3;
N33=CZ3 + CZ4 + No33 - MY3R.*VP13 + MX3R.*VP23;
FDI13=C3.*E13 - E23.*S3;
FDI23=C3.*E23 + E13.*S3;
E12=F12 + FDI13;
E22=F22 + FDI23;
N32=d3.*FDI23 + N33 + No32 - MY2.*VP12 + MX2R.*VP22;
FDI22=C2.*E22 + E12.*S2;
N31=d2.*FDI22 + N32 + No31;
GAM1=N31 + FV1.*QP1 + FS1.*sign(QP1);
GAM2=N32 + IA2.*QDP2 + FV2.*QP2 + FS2.*sign(QP2);
GAM3=N33 + IA3.*QDP3 + FV3.*QP3 + FS3.*sign(QP3);
GAM4=E34 + IA4.*QDP4 + FV4.*QP4 + FS4.*sign(QP4);

GAM(1) = GAM1;
GAM(2) = GAM2;
GAM(3) = GAM3;
GAM(4) = GAM4;
```

A.7. Modelo dinámico directo

```
function [sys, x0,str,ts] = f_scara(t,x,u,flag,QI)
%  function [sys,x0,str,ts] =
f_scara(t,x,u,flag,L1,L2,ZZ1,ZZ2,MX2,MY2,FV1,FV2,FS1,FS2,QI)
%  f_scara An example M-file S-function for defining a continuous system.
%  Example M-file S-function implementing non linear continuous equations of a 2
dof scara robot
%  Etats : x(1)=q1,x(2)=q2,x(3)=qp1,x(4)=qp2
```

```
%     qpp =f(q,qp,u)
%     y  = x
%
%   See csfunc.m for a continuous linear state space example
%   See sfuntmpl.m for a general S-function template.
%
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(QI);

  %%%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%%
  case 1,
%   sys=mdlDerivatives(t,x,u,L1,L2,ZZ1,ZZ2,MX2,MY2,FV1,FV2,FS1,FS2);
   sys=mdlDerivatives(t,x,u);

  %%%%%%%%%%
  % Outputs %
  %%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%
  case { 2, 4, 9 },
    sys = [];

  %%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% end f_scara
```

```
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(QI)

sizes = simsizes;
sizes.NumContStates  = 8;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 8;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

%
% initialize the initial conditions
%

%x0=[QI(1);0;QI(2);0];
x0=[QI(1);QI(2);QI(3);QI(4);0;0;0;0]; % x=[q1,q2,q3,q4,qp1,qp2,qp3,qp4]

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes
%
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================
%
```

```
%function sys=mdlDerivatives(t,x,u,L1,L2,ZZ1,ZZ2,MX2,MY2,FV1,FV2,FS1,FS2)
function sys=mdlDerivatives(t,x,u)

% Parámetros del SCARA para la simulación:

G3=9.81;
M4=1.8;
d2=0.5;
d3=0.3;%
r4=0.2;%
ZZ1R=3.38;
ZZ2R=0.063;
ZZ3R=0.1;%
MX2R=2*0.121;
MX3R=0.2;%
MY3R=0.1;%
MY2=0.001;
FV1=0.1;
FV2=0.012;
FV3=FV1;
FV4=FV2;
FS1=0.57;
FS2=0.125;
FS3=FS1;
FS4=FS2;
IA2=0;
CZ1=0;
FX2=0;
FY2=0;
FX3=0;
FY3=0;
FZ3=0;
CZ3=0;
CZ4=0;
CX3=0;
CX4=0;
CY3=0;
CY4=0;
FX4=0;
FY4=0;
FZ4=0;
```

```
IA3=0;
IA4=0.045;
GAM1=u(1);
GAM2=u(2);
GAM3=u(3);
GAM4=u(4);

S1=sin(x(1));
C1=cos(x(1));
S2=sin(x(2));
C2=cos(x(2));
S3=sin(x(3));
C3=cos(x(3));

QP1=x(5);
QP2=x(6);
QP3=x(7);
QP4=x(8);

W32=QP1 + QP2;
JPR132=d2.*S2;
JPR232=C2.*d2;
W33=QP3 + W32;
JPR133=d3.*S3;
JPR233=C3.*d3;
JW31=QP1.*ZZ1R;
JW32=W32.*ZZ2R;
SW12=-(MX2R.*W32.^2);
SW22=-(MY2.*W32.^2);
LW12=-(C2.*d2.*QP1.^2);
LW22=d2.*QP1.^2.*S2;
JW33=W33.*ZZ3R;
SW13=-(MX3R.*W33.^2);
SW23=-(MY3R.*W33.^2);
LW13=-(C3.*d3.*W32.^2);
LW23=d3.*S3.*W32.^2;
JD4=1./(IA4 + M4);
JU64=JD4.*M4;
GW4=-FZ4 + GAM4 - FV4.*QP4 - FS4.*sign(QP4);
GK664=M4 - JU64.*M4;
VS64=GW4.*JU64;
```

```
AP64=FZ4 + VS64;
GX154=-(M4.*r4);
GX244=M4.*r4;
TKT114=-(GX154.*r4);
TKT514=-(M4.*r4);
TKT224=GX244.*r4;
TKT424=M4.*r4;
VBE13=-CX3 - CX4 + FY4.*r4;
VBE23=-CY3 - CY4 - FX4.*r4;
VBE33=-CZ3 - CZ4;
VBE43=-FX3 - FX4 - SW13;
VBE53=-FY3 - FY4 - SW23;
VBE63=-AP64 - FZ3;
JD3=1./(IA3 + ZZ3R);
JU33=JD3.*ZZ3R;
JU43=-(JD3.*MY3R);
JU53=JD3.*MX3R;
GW3=GAM3 - FV3.*QP3 + VBE33 - FS3.*sign(QP3);
GK333=ZZ3R - JU33.*ZZ3R;
GK343=-MY3R + JU33.*MY3R;
GK353=MX3R - JU33.*MX3R;
GK433=-MY3R - JU43.*ZZ3R;
GK443=M4 + JU43.*MY3R;
GK453=-(JU43.*MX3R);
GK533=MX3R - JU53.*ZZ3R;
GK543=JU53.*MY3R;
GK553=M4 - JU53.*MX3R;
NG13=LW23.*TKT514;
NG23=LW13.*TKT424;
NG33=GK343.*LW13 + GK353.*LW23;
NG43=GK443.*LW13 + GK453.*LW23;
NG53=GK543.*LW13 + GK553.*LW23;
VS33=GW3.*JU33 + NG33;
VS43=GW3.*JU43 + NG43;
VS53=GW3.*JU53 + NG53;
AP13=NG13 - VBE13;
AP23=NG23 - VBE23;
AP33=-VBE33 + VS33;
AP43=-VBE43 + VS43;
AP53=-VBE53 + VS53;
GX113=C3.*TKT114;
```

*GX123=-(S3.*TKT224);*
*GX143=-(S3.*TKT424);*
*GX153=C3.*TKT514;*
*GX163=C3.*MY3R + MX3R.*S3;*
*GX213=-(d3.*MY3R) + S3.*TKT114;*
*GX223=d3.*MX3R + C3.*TKT224;*
*GX243=C3.*TKT424;*
*GX253=S3.*TKT514;*
*GX263=-(d3.*GK664) - C3.*MX3R + MY3R.*S3;*
*GX313=JPR233.*TKT514;*
*GX323=JPR133.*TKT424;*
*GX333=GK333 + GK433.*JPR133 + GK533.*JPR233;*
*GX343=GK343 + GK443.*JPR133 + GK543.*JPR233;*
*GX353=GK353 + GK453.*JPR133 + GK553.*JPR233;*
*GX413=-(S3.*TKT514);*
*GX423=C3.*TKT424;*
*GX433=C3.*GK433 - GK533.*S3;*
*GX443=C3.*GK443 - GK543.*S3;*
*GX453=C3.*GK453 - GK553.*S3;*
*GX513=C3.*TKT514;*
*GX523=S3.*TKT424;*
*GX533=C3.*GK533 + GK433.*S3;*
*GX543=C3.*GK543 + GK443.*S3;*
*GX553=C3.*GK553 + GK453.*S3;*
*TKT113=C3.*GX113 - GX123.*S3;*
*TKT213=C3.*GX213 - GX223.*S3;*
*TKT313=C3.*GX313 - GX323.*S3;*
*TKT413=C3.*GX413 - GX423.*S3;*
*TKT513=C3.*GX513 - GX523.*S3;*
*TKT613=C3.*MY3R + MX3R.*S3;*
*TKT223=C3.*GX223 - d3.*GX263 + GX213.*S3;*
*TKT323=C3.*GX323 + GX313.*S3;*
*TKT423=C3.*GX423 + GX413.*S3;*
*TKT523=C3.*GX523 + GX513.*S3;*
*TKT623=-(d3.*GK664) - C3.*MX3R + MY3R.*S3;*
*TKT333=GX333 + GX343.*JPR133 + GX353.*JPR233;*
*TKT433=GX433 + GX443.*JPR133 + GX453.*JPR233;*
*TKT533=GX533 + GX543.*JPR133 + GX553.*JPR233;*
*TKT443=C3.*GX443 - GX453.*S3;*
*TKT543=C3.*GX543 - GX553.*S3;*
*TKT553=C3.*GX553 + GX543.*S3;*

```
MJE612=MY2 + TKT613;
MJE622=-MX2R + TKT623;
MJE332=TKT333 + ZZ2R;
MJE432=-MY2 + TKT433;
MJE532=MX2R + TKT533;
VBE12=-(AP13.*C3) + AP23.*S3;
VBE22=-(AP23.*C3) - AP13.*S3 - d3.*VBE63;
VBE32=-AP33 - AP43.*JPR133 - AP53.*JPR233;
VBE42=-(AP43.*C3) + AP53.*S3 - SW12;
VBE52=-(AP53.*C3) - AP43.*S3 - SW22;
JD2=1./(IA2 + MJE332);
JU12=JD2.*TKT313;
JU22=JD2.*TKT323;
JU32=JD2.*MJE332;
JU42=JD2.*MJE432;
JU52=JD2.*MJE532;
GW2=GAM2 - FV2.*QP2 + VBE32 - FS2.*sign(QP2);
GK112=TKT113 - JU12.*TKT313;
GK122=TKT213 - JU12.*TKT323;
GK132=-(JU12.*MJE332) + TKT313;
GK142=-(JU12.*MJE432) + TKT413;
GK152=-(JU12.*MJE532) + TKT513;
GK212=TKT213 - JU22.*TKT313;
GK222=TKT223 - JU22.*TKT323;
GK232=-(JU22.*MJE332) + TKT323;
GK242=-(JU22.*MJE432) + TKT423;
GK252=-(JU22.*MJE532) + TKT523;
GK312=TKT313 - JU32.*TKT313;
GK322=TKT323 - JU32.*TKT323;
GK332=MJE332 - JU32.*MJE332;
GK342=MJE432 - JU32.*MJE432;
GK352=MJE532 - JU32.*MJE532;
GK412=-(JU42.*TKT313) + TKT413;
GK422=-(JU42.*TKT323) + TKT423;
GK432=-(JU42.*MJE332) + MJE432;
GK442=-(JU42.*MJE432) + TKT443;
GK452=-(JU42.*MJE532) + TKT543;
GK512=-(JU52.*TKT313) + TKT513;
GK522=-(JU52.*TKT323) + TKT523;
GK532=-(JU52.*MJE332) + MJE532;
GK542=-(JU52.*MJE432) + TKT543;
```

```
GK552=-(JU52.*MJE532) + TKT553;
NG12=GK142.*LW12 + GK152.*LW22;
NG22=GK242.*LW12 + GK252.*LW22;
NG32=GK342.*LW12 + GK352.*LW22;
NG42=GK442.*LW12 + GK452.*LW22;
NG52=GK542.*LW12 + GK552.*LW22;
VS12=GW2.*JU12 + NG12;
VS22=GW2.*JU22 + NG22;
VS32=GW2.*JU32 + NG32;
VS42=GW2.*JU42 + NG42;
VS52=GW2.*JU52 + NG52;
AP12=-VBE12 + VS12;
AP22=-VBE22 + VS22;
AP32=-VBE32 + VS32;
AP42=-VBE42 + VS42;
AP52=-VBE52 + VS52;
GX112=C2.*GK112 - GK212.*S2;
GX122=C2.*GK122 - GK222.*S2;
GX132=C2.*GK132 - GK232.*S2;
GX142=C2.*GK142 - GK242.*S2;
GX152=C2.*GK152 - GK252.*S2;
GX162=C2.*MJE612 - MJE622.*S2;
GX212=C2.*GK212 - d2.*MJE612 + GK112.*S2;
GX222=C2.*GK222 - d2.*MJE622 + GK122.*S2;
GX232=C2.*GK232 + GK132.*S2;
GX242=C2.*GK242 + GK142.*S2;
GX252=C2.*GK252 + GK152.*S2;
GX262=-(d2.*GK664) + C2.*MJE622 + MJE612.*S2;
GX312=GK312 + GK412.*JPR132 + GK512.*JPR232;
GX322=GK322 + GK422.*JPR132 + GK522.*JPR232;
GX332=GK332 + GK432.*JPR132 + GK532.*JPR232;
GX342=GK342 + GK442.*JPR132 + GK542.*JPR232;
GX352=GK352 + GK452.*JPR132 + GK552.*JPR232;
GX412=C2.*GK412 - GK512.*S2;
GX422=C2.*GK422 - GK522.*S2;
GX432=C2.*GK432 - GK532.*S2;
GX442=C2.*GK442 - GK542.*S2;
GX452=C2.*GK452 - GK552.*S2;
GX512=C2.*GK512 + GK412.*S2;
GX522=C2.*GK522 + GK422.*S2;
GX532=C2.*GK532 + GK432.*S2;
```

```
GX542=C2.*GK542 + GK442.*S2;
GX552=C2.*GK552 + GK452.*S2;
TKT112=C2.*GX112 - GX122.*S2;
TKT212=C2.*GX212 - GX222.*S2;
TKT312=C2.*GX312 - GX322.*S2;
TKT412=C2.*GX412 - GX422.*S2;
TKT512=C2.*GX512 - GX522.*S2;
TKT612=C2.*MJE612 - MJE622.*S2;
TKT222=C2.*GX222 - d2.*GX262 + GX212.*S2;
TKT322=C2.*GX322 + GX312.*S2;
TKT422=C2.*GX422 + GX412.*S2;
TKT522=C2.*GX522 + GX512.*S2;
TKT622=-(d2.*GK664) + C2.*MJE622 + MJE612.*S2;
TKT332=GX332 + GX342.*JPR132 + GX352.*JPR232;
TKT432=GX432 + GX442.*JPR132 + GX452.*JPR232;
TKT532=GX532 + GX542.*JPR132 + GX552.*JPR232;
TKT442=C2.*GX442 - GX452.*S2;
TKT542=C2.*GX542 - GX552.*S2;
TKT552=C2.*GX552 + GX542.*S2;
MJE331=TKT332 + ZZ1R;
VBE11=-(AP12.*C2) + AP22.*S2;
VBE21=-(AP22.*C2) - AP12.*S2 - d2.*VBE63;
VBE31=-AP32 - AP42.*JPR132 - AP52.*JPR232;
VBE41=-(AP42.*C2) + AP52.*S2;
VBE51=-(AP52.*C2) - AP42.*S2;
JD1=1./MJE331;
JU11=JD1.*TKT312;
JU21=JD1.*TKT322;
JU31=JD1.*MJE331;
JU41=JD1.*TKT432;
JU51=JD1.*TKT532;
GW1=GAM1 - FV1.*QP1 + VBE31 - FS1.*sign(QP1);
QDP1=GW1.*JD1;
VR42=LW12 + JPR132.*QDP1;
VR52=LW22 + JPR232.*QDP1;
GU2=JU32.*QDP1 + JU42.*VR42 + JU52.*VR52;
QDP2=-GU2 + GW2.*JD2;
WP32=QDP1 + QDP2;
VR43=LW13 + C3.*VR42 + S3.*VR52 + JPR133.*WP32;
VR53=LW23 - S3.*VR42 + C3.*VR52 + JPR233.*WP32;
GU3=JU43.*VR43 + JU53.*VR53 + JU33.*WP32;
```

```
QDP3=-GU3 + GW3.*JD3;
WP33=QDP3 + WP32;
GU4=-(G3.*JU64);
QDP4=-GU4 + GW4.*JD4;

sys(1) = x(5) ;
sys(2) = x(6) ;
sys(3) = x(7) ;
sys(4) = x(8) ;
sys(5) = QDP1;
sys(6) = QDP2;
sys(7) = QDP3;
sys(8) = QDP4;
% end mdlDerivatives
%
%=============================================================
% mdlOutputs
% Return the block outputs.
%=============================================================
%
function sys=mdlOutputs(t,x,u)

  sys(1) = x(1);
  sys(2) = x(2);
  sys(3) = x(3);
  sys(4) = x(4);
  sys(5) = x(5);
  sys(6) = x(6);
  sys(7) = x(7);
  sys(8) = x(8);

% end mdlOutputs
```

A.8. Medición del índice (promedio cuadrado del error)

```
double errorX;
double errorY;

errorX=0;
errorY=0;
```

```
for i=1:nbench
   % Sumatoria de los errores en cada eje (cons1 deseada, Xr obtenida)
   errorX= errorX + (cons1(i) - Xr(i))^2;
   errorY= errorY + (cons2(i) - Yr(i))^2;
end

errorX
errorY

errorX = errorX/nbench;        % El error sobre el número de muestras
errorY = errorY/nbench;

%Ahora se halla el error total
double ErrorT ;
ErrorT = (errorX + errorY)/2
```

ANEXO B

CÓDIGO DE LA APLICACIÓN EN EASY JAVA SIMULATIONS

B.1. Modelo dinámico directo

```
public void modelo () {

double GAM1=Torque[0];
double GAM2=Torque[1];
double GAM3=Torque[2];
double GAM4=Torque[3];

double S1=Math.sin(x0);
double C1=Math.cos(x0);
double S2=Math.sin(x1);
double C2=Math.cos(x1);
double S3=Math.sin(x2);
double C3=Math.cos(x2);

double QP1=x4;
double QP2=x5;
double QP3=x6;
double QP4=x7;

double W32=QP1 + QP2;
double JPR132=d2*S2;
double JPR232=C2*d2;
double W33=QP3 + W32;
double JPR133=d3*S3;
double JPR233=C3*d3;
double JW31=QP1*ZZ1R;
double JW32=W32*ZZ2R;
double SW12=-(MX2R*W32*W32);
double SW22=-(MY2*W32*W32);
double LW12=-(C2*d2*QP1*QP1);
double LW22=d2*QP1*QP1*S2;
double JW33=W33*ZZ3R;
double SW13=-(MX3R*W33*W33);
```

```
double SW23=-(MY3R*W33*W33);
double LW13=-(C3*d3*W32*W32);
double LW23=d3*S3*W32*W32;
double JD4=1./(IA4 + M4);
double JU64=JD4*M4;
double GW4=-FZ4 + GAM4 - FV4*QP4 - FS4*Math.signum(QP4);
double GK664=M4 - JU64*M4;
double VS64=GW4*JU64;
double AP64=FZ4 + VS64;
double GX154=-(M4*r4);
double GX244=M4*r4;
double TKT114=-(GX154*r4);
double TKT514=-(M4*r4);
double TKT224=GX244*r4;
double TKT424=M4*r4;
double VBE13=-CX3 - CX4 + FY4*r4;
double VBE23=-CY3 - CY4 - FX4*r4;
double VBE33=-CZ3 - CZ4;
double VBE43=-FX3 - FX4 - SW13;
double VBE53=-FY3 - FY4 - SW23;
double VBE63=-AP64 - FZ3;
double JD3=1./(IA3 + ZZ3R);
double JU33=JD3*ZZ3R;
double JU43=-(JD3*MY3R);
double JU53=JD3*MX3R;
double GW3=GAM3 - FV3*QP3 + VBE33 - FS3*Math.signum(QP3);
double GK333=ZZ3R - JU33*ZZ3R;
double GK343=-MY3R + JU33*MY3R;
double GK353=MX3R - JU33*MX3R;
double GK433=-MY3R - JU43*ZZ3R;
double GK443=M4 + JU43*MY3R;
double GK453=-(JU43*MX3R);
double GK533=MX3R - JU53*ZZ3R;
double GK543=JU53*MY3R;
double GK553=M4 - JU53*MX3R;
double NG13=LW23*TKT514;
double NG23=LW13*TKT424;
double NG33=GK343*LW13 + GK353*LW23;
double NG43=GK443*LW13 + GK453*LW23;
double NG53=GK543*LW13 + GK553*LW23;
double VS33=GW3*JU33 + NG33;
```

```
double VS43=GW3*JU43 + NG43;
double VS53=GW3*JU53 + NG53;
double AP13=NG13 - VBE13;
double AP23=NG23 - VBE23;
double AP33=-VBE33 + VS33;
double AP43=-VBE43 + VS43;
double AP53=-VBE53 + VS53;
double GX113=C3*TKT114;
double GX123=-(S3*TKT224);
double GX143=-(S3*TKT424);
double GX153=C3*TKT514;
double GX163=C3*MY3R + MX3R*S3;
double GX213=-(d3*MY3R) + S3*TKT114;
double GX223=d3*MX3R + C3*TKT224;
double GX243=C3*TKT424;
double GX253=S3*TKT514;
double GX263=-(d3*GK664) - C3*MX3R + MY3R*S3;
double GX313=JPR233*TKT514;
double GX323=JPR133*TKT424;
double GX333=GK333 + GK433*JPR133 + GK533*JPR233;
double GX343=GK343 + GK443*JPR133 + GK543*JPR233;
double GX353=GK353 + GK453*JPR133 + GK553*JPR233;
double GX413=-(S3*TKT514);
double GX423=C3*TKT424;
double GX433=C3*GK433 - GK533*S3;
double GX443=C3*GK443 - GK543*S3;
double GX453=C3*GK453 - GK553*S3;
double GX513=C3*TKT514;
double GX523=S3*TKT424;
double GX533=C3*GK533 + GK433*S3;
double GX543=C3*GK543 + GK443*S3;
double GX553=C3*GK553 + GK453*S3;
double TKT113=C3*GX113 - GX123*S3;
double TKT213=C3*GX213 - GX223*S3;
double TKT313=C3*GX313 - GX323*S3;
double TKT413=C3*GX413 - GX423*S3;
double TKT513=C3*GX513 - GX523*S3;
double TKT613=C3*MY3R + MX3R*S3;
double TKT223=C3*GX223 - d3*GX263 + GX213*S3;
double TKT323=C3*GX323 + GX313*S3;
double TKT423=C3*GX423 + GX413*S3;
```

```
double TKT523=C3*GX523 + GX513*S3;
double TKT623=-(d3*GK664) - C3*MX3R + MY3R*S3;
double TKT333=GX333 + GX343*JPR133 + GX353*JPR233;
double TKT433=GX433 + GX443*JPR133 + GX453*JPR233;
double TKT533=GX533 + GX543*JPR133 + GX553*JPR233;
double TKT443=C3*GX443 - GX453*S3;
double TKT543=C3*GX543 - GX553*S3;
double TKT553=C3*GX553 + GX543*S3;
double MJE612=MY2 + TKT613;
double MJE622=-MX2R + TKT623;
double MJE332=TKT333 + ZZ2R;
double MJE432=-MY2 + TKT433;
double MJE532=MX2R + TKT533;
double VBE12=-(AP13*C3) + AP23*S3;
double VBE22=-(AP23*C3) - AP13*S3 - d3*VBE63;
double VBE32=-AP33 - AP43*JPR133 - AP53*JPR233;
double VBE42=-(AP43*C3) + AP53*S3 - SW12;
double VBE52=-(AP53*C3) - AP43*S3 - SW22;
double JD2=1./(IA2 + MJE332);
double JU12=JD2*TKT313;
double JU22=JD2*TKT323;
double JU32=JD2*MJE332;
double JU42=JD2*MJE432;
double JU52=JD2*MJE532;
double GW2=GAM2 - FV2*QP2 + VBE32 - FS2*Math.signum(QP2);
double GK112=TKT113 - JU12*TKT313;
double GK122=TKT213 - JU12*TKT323;
double GK132=-(JU12*MJE332) + TKT313;
double GK142=-(JU12*MJE432) + TKT413;
double GK152=-(JU12*MJE532) + TKT513;
double GK212=TKT213 - JU22*TKT313;
double GK222=TKT223 - JU22*TKT323;
double GK232=-(JU22*MJE332) + TKT323;
double GK242=-(JU22*MJE432) + TKT423;
double GK252=-(JU22*MJE532) + TKT523;
double GK312=TKT313 - JU32*TKT313;
double GK322=TKT323 - JU32*TKT323;
double GK332=MJE332 - JU32*MJE332;
double GK342=MJE432 - JU32*MJE432;
double GK352=MJE532 - JU32*MJE532;
double GK412=-(JU42*TKT313) + TKT413;
```

```
double GK422=-(JU42*TKT323) + TKT423;
double GK432=-(JU42*MJE332) + MJE432;
double GK442=-(JU42*MJE432) + TKT443;
double GK452=-(JU42*MJE532) + TKT543;
double GK512=-(JU52*TKT313) + TKT513;
double GK522=-(JU52*TKT323) + TKT523;
double GK532=-(JU52*MJE332) + MJE532;
double GK542=-(JU52*MJE432) + TKT543;
double GK552=-(JU52*MJE532) + TKT553;
double NG12=GK142*LW12 + GK152*LW22;
double NG22=GK242*LW12 + GK252*LW22;
double NG32=GK342*LW12 + GK352*LW22;
double NG42=GK442*LW12 + GK452*LW22;
double NG52=GK542*LW12 + GK552*LW22;
double VS12=GW2*JU12 + NG12;
double VS22=GW2*JU22 + NG22;
double VS32=GW2*JU32 + NG32;
double VS42=GW2*JU42 + NG42;
double VS52=GW2*JU52 + NG52;
double AP12=-VBE12 + VS12;
double AP22=-VBE22 + VS22;
double AP32=-VBE32 + VS32;
double AP42=-VBE42 + VS42;
double AP52=-VBE52 + VS52;
double GX112=C2*GK112 - GK212*S2;
double GX122=C2*GK122 - GK222*S2;
double GX132=C2*GK132 - GK232*S2;
double GX142=C2*GK142 - GK242*S2;
double GX152=C2*GK152 - GK252*S2;
double GX162=C2*MJE612 - MJE622*S2;
double GX212=C2*GK212 - d2*MJE612 + GK112*S2;
double GX222=C2*GK222 - d2*MJE622 + GK122*S2;
double GX232=C2*GK232 + GK132*S2;
double GX242=C2*GK242 + GK142*S2;
double GX252=C2*GK252 + GK152*S2;
double GX262=-(d2*GK664) + C2*MJE622 + MJE612*S2;
double GX312=GK312 + GK412*JPR132 + GK512*JPR232;
double GX322=GK322 + GK422*JPR132 + GK522*JPR232;
double GX332=GK332 + GK432*JPR132 + GK532*JPR232;
double GX342=GK342 + GK442*JPR132 + GK542*JPR232;
double GX352=GK352 + GK452*JPR132 + GK552*JPR232;
```

```
double GX412=C2*GK412 - GK512*S2;
double GX422=C2*GK422 - GK522*S2;
double GX432=C2*GK432 - GK532*S2;
double GX442=C2*GK442 - GK542*S2;
double GX452=C2*GK452 - GK552*S2;
double GX512=C2*GK512 + GK412*S2;
double GX522=C2*GK522 + GK422*S2;
double GX532=C2*GK532 + GK432*S2;
double GX542=C2*GK542 + GK442*S2;
double GX552=C2*GK552 + GK452*S2;
double TKT112=C2*GX112 - GX122*S2;
double TKT212=C2*GX212 - GX222*S2;
double TKT312=C2*GX312 - GX322*S2;
double TKT412=C2*GX412 - GX422*S2;
double TKT512=C2*GX512 - GX522*S2;
double TKT612=C2*MJE612 - MJE622*S2;
double TKT222=C2*GX222 - d2*GX262 + GX212*S2;
double TKT322=C2*GX322 + GX312*S2;
double TKT422=C2*GX422 + GX412*S2;
double TKT522=C2*GX522 + GX512*S2;
double TKT622=-(d2*GK664) + C2*MJE622 + MJE612*S2;
double TKT332=GX332 + GX342*JPR132 + GX352*JPR232;
double TKT432=GX432 + GX442*JPR132 + GX452*JPR232;
double TKT532=GX532 + GX542*JPR132 + GX552*JPR232;
double TKT442=C2*GX442 - GX452*S2;
double TKT542=C2*GX542 - GX552*S2;
double TKT552=C2*GX552 + GX542*S2;
double MJE331=TKT332 + ZZ1R;
double VBE11=-(AP12*C2) + AP22*S2;
double VBE21=-(AP22*C2) - AP12*S2 - d2*VBE63;
double VBE31=-AP32 - AP42*JPR132 - AP52*JPR232;
double VBE41=-(AP42*C2) + AP52*S2;
double VBE51=-(AP52*C2) - AP42*S2;
double JD1=1./MJE331;
double JU11=JD1*TKT312;
double JU21=JD1*TKT322;
double JU31=JD1*MJE331;
double JU41=JD1*TKT432;
double JU51=JD1*TKT532;
double GW1=GAM1 - FV1*QP1 + VBE31 - FS1*Math.signum(QP1);
QDP1=GW1*JD1;
```

```
double VR42=LW12 + JPR132*QDP1;
double VR52=LW22 + JPR232*QDP1;
double GU2=JU32*QDP1 + JU42*VR42 + JU52*VR52;
QDP2=-GU2 + GW2*JD2;
double WP32=QDP1 + QDP2;
double VR43=LW13 + C3*VR42 + S3*VR52 + JPR133*WP32;
double VR53=LW23 - S3*VR42 + C3*VR52 + JPR233*WP32;
double GU3=JU43*VR43 + JU53*VR53 + JU33*WP32;
QDP3=-GU3 + GW3*JD3;
double WP33=QDP3 + WP32;
double GU4=-(G3*JU64);
QDP4=-GU4 + GW4*JD4;
}
```

B.2. Trayectoria lineal

```
public void trayectoria () {
// Posición en x
        xent=tamRobot*temp/10 + inix;
// Posición en y
        yent=tamRobot*temp/10 + iniy;
// Posición en z
        zent =iniz;//constante
}
```

B.3. Trayectoria circular

```
public void circular(){

// Una trayectoria circular con un radio de 20mm.
// Posición en x
        xent=tamRobot*0.02*Math.sin(2.0943951*temp)+inix;
// Posición en y
        yent=tamRobot*0.02*Math.cos(2.0943951*temp)+iniy;//6.2831853/3
// Posición en z
        zent=iniz;
}
```

B.4. Trayectoria triangular

```
public void triangulo(){
```

```
// Una trayectoria triangular, tipo isosceles

if (temp<=1){

// Posición en x
        xent=tamRobot*temp/10 +inix;
// Posición en y
        yent=tamRobot*temp/10+ iniy;
// Posición en z
        zent=iniz;
}

if (temp>1 && temp<=2){

// Posición en x
        xent=tamRobot*(temp-1)/10  + (inix+tamRobot*0.1);
// Posición en y
        yent=tamRobot*(-temp+1)/10+ (iniy+tamRobot*0.1);
// Posición en z
        zent=iniz;
}

if (temp>2){

// Posición en x
        xent=tamRobot*(-temp+2)/10 + (inix+tamRobot*0.2);
// Posición en y
        yent=iniy;
// Posición en z
        zent=iniz;
 } }
```

B.5. Trayectoria pick-and-place

```
public void pickup(){

// Una trayectoria que varía principalmente en el eje z

if (temp<=1){

// Posición en x
```

```
        xent=inix;
// Posición en y
        yent=iniy;
// Posición en z
        zent=-temp/10+ iniz;
}

if (temp>1 && temp<=2){

// Posición en x
        xent=inix;
// Posición en y
        yent=iniy;
// Posición en z
        zent=(temp-1)/10 + (iniz-0.1);
}

// Desplazamiento
if (temp>2 && temp<=3){

// Posición en x
        xent=inix - tamRobot*(temp-2)/10;
// Posición en y
        yent=iniy;
// Posición en z
        zent=iniz;
}

if (temp>3 && temp<=4)
{
// Posición en x
        xent=inix-tamRobot*0.1;
// Posición en y
        yent=iniy;
// Posición en z
        zent=(-temp+3)/10 +iniz;
}

if (temp>4 && temp<=5)
{
// Posición en x
```

```
        xent=inix-tamRobot*0.1;
// Posición en y
        yent=iniy;
// Posición en z
        zent=(temp-5)/10 +iniz;
}
}
```

B.6 Modelo geométrico inverso

```
public void MGI () {

// Variables necesarias para el cálculo de la función

double C2,B1,S1,C1,B2;

C2 = (xent*xent + yent*yent - d2*d2 - d3*d3) / (2*d2*d3);
B1 = d2 + d3*C2;

q[1] = Math.atan2((-Math.sqrt(1 - C2*C2)),C2);

B2 = d3*Math.sin(q[1]);

S1 = (B1*yent - B2*xent)/(B1*B1 + B2*B2);
C1 = (B1*xent + B2*yent)/(B1*B1 + B2*B2);

q[0] = Math.atan2(S1,C1);
q[2] = Math.atan2(0,-1) - q[1] - q[0];
q[3] = zent;
}
```

B.7. Controlador PID

```
public void PID () {

// Proporcional

double[] proporcional=new double[4];
double[] diferencia=new double[4];

proporcional[0] = q[0] - x0;  // Esta diferencia es la que se integra
```

```
proporcional[1] = q[1] - x1;
proporcional[2] = q[2] - x2;
proporcional[3] = q[3] - x3;

for(int a=0;a<=3;a++)
        {
        diferencia[a] = proporcional[a];
        proporcional[a]*= Kp[a];
        }
// Derivativa

derivativa(q); // Aquí se le pasa a Derivativa los q

double[] derivativo=new double[4];

derivativo[0] = qres[0]-x4;
derivativo[1] = qres[1]-x5;
derivativo[2] = qres[2]-x6;
derivativo[3] = qres[3]-x7;

for(int b=0;b<=3;b++)
        {
        derivativo[b]*= Kv[b];
        }

// Integral

double[] integral=new double[4];

integral(diferencia);

for(int k=0;k<=3;k++)
        {
        integral[k]=IntRes[k]*Ki[k];
        }

for(int m=0;m<=3;m++)
        {
        Torque[m]=proporcional[m]+integral[m]+derivativo[m];
        } }
```

B.8. Controlador CTC

```
public void CTC () {

// Se obtiene la diferencia entre la posición deseada y la posición actual
double[] p= new double[4];

p[0] = q[0] - x0;
p[1] = q[1] - x1;
p[2] = q[2] - x2;
p[3] = q[3] - x3;

// Se obtiene la velocidad deseada y el cálculo de la diferencia entre la deseada y
la obtenida
derivativa(q);

double[] v=new double[4];

v[0] = qres[0]-x4;
v[1] = qres[1]-x5;
v[2] = qres[2]-x6;
v[3] = qres[3]-x7;

// Multiplica a cada uno por los valores de las constantes
for(int a=0;a<4;a++)
{
        v[a]*=Kv[a];
        p[a]*=Kp[a]; //
}

double[] wt= new double[4];

for(int a=0;a<4;a++)
{
        wt[a] = p[a] + v[a];
}

// Se agrega el modelo inverso del scara
double t2=q[1];
double t3=q[2];
double QP1=qres[0];
```

```
double QP2=qres[1];
double QP3=qres[2];
double QP4=qres[3];
QDP1=wt[0];
QDP2=wt[1];
QDP3=wt[2];
QDP4=wt[3];
double S2=Math.sin(t2);
double C2=Math.cos(t2);
double S3=Math.sin(t3);
double C3=Math.cos(t3);
double DV331=-QP1*QP1;
double No31=QDP1*ZZ1R;
double W32=QP1 + QP2;
double WP32=QDP1 + QDP2;
double DV332=-W32*W32;
double VSP12=d2*DV331;
double VSP22=d2*QDP1;
double VP12=C2*VSP12 + S2*VSP22;
double VP22=-(S2*VSP12) + C2*VSP22;
double F12=DV332*MX2R - MY2*WP32;
double F22=DV332*MY2 + MX2R*WP32;
double No32=WP32*ZZ2R;
double W33=QP3 + W32;
double WP33=QDP3 + WP32;
double DV333=-W33*W33;
double VSP13=d3*DV332 + VP12;
double VSP23=VP22 + d3*WP32;
double VP13=C3*VSP13 + S3*VSP23;
double VP23=-(S3*VSP13) + C3*VSP23;
double F13=DV333*MX3R - MY3R*WP33;
double F23=DV333*MY3R + MX3R*WP33;
double No33=WP33*ZZ3R;
double VP34=-G3 + QDP4;
double F14=M4*VP13;
double F24=M4*VP23;
double F34=M4*VP34;
double E14=F14 + FX4;
double E24=F24 + FY4;
double E34=F34 + FZ4;
double E13=E14 + F13 + FX3;
```

```
double E23=E24 + F23 + FY3;
double N33=CZ3 + CZ4 + No33 - MY3R*VP13 + MX3R*VP23;
double FDI13=C3*E13 - E23*S3;
double FDI23=C3*E23 + E13*S3;
double E12=F12 + FDI13;
double E22=F22 + FDI23;
double N32=d3*FDI23 + N33 + No32 - MY2*VP12 + MX2R*VP22;
double FDI22=C2*E22 + E12*S2;
double N31=d2*FDI22 + N32 + No31;
double GAM1=N31 + FV1*QP1 + FS1*Math.signum(QP1);
double GAM2=N32 + IA2*QDP2 + FV2*QP2 + FS2*Math.signum(QP2);
double GAM3=N33 + IA3*QDP3 + FV3*QP3 + FS3*Math.signum(QP3);
double GAM4=E34 + IA4*QDP4 + FV4*QP4 + FS4*Math.signum(QP4);

Torque[0]= GAM1;
Torque[1]= GAM2;
Torque[2] = GAM3;
Torque[3] = GAM4;
}
```

B.9. Función derivativa

```
public void derivativa (double v[]) {

for(int a=0;a<4;a++)
        {
        qant1[a]=qact[a];
        qact[a]=v[a];
        }

        for(int b=0;b<4;b++)
                {
                qres[b]=(qact[b]-qant1[b])/delta;
                }
i++;
}
```

B.10. Función integradora

```
public void integral (double qact[]) {
double[] ci= new double[4];
```

```
// Se obtiene la altura del rectángulo y se multiplica por la base.
for(int j=0;j<=3;j++)
        {
        ci[j]=(IntAnt[j]+qact[j])/2;
        IntRes[j]+=ci[j]*delta;
        }
//Se actualiza el punto anterior
for(int j=0;j<=3;j++)
{
        IntAnt[j]=qact[jj];
} }
```

B.11. Salida (conversión de articulares a cartesianas, medición del error)

```
public void salida () {

// Se convierte de coordenadas articulares a coordenadas cartesianas (tomado del
programa mgd_scara)
xsal=(d3*Math.cos(x0+x1)+d2*Math.cos(x0));
ysal=(d3*Math.sin(x0+x1) + d2*Math.sin(x0));
zsal=x3;

//Cálculo del error en cada eje
ErrorX+= (xent - xsal)*(xent - xsal);
ErrorY+=(yent - ysal)*(yent - ysal);

Motor1x=0;
Motor1y=0;

Motor2x=d2*Math.cos(x0);
Motor2y=d2*Math.sin(x0);

Motor3x=d3*Math.cos(x0+x1)+d2*Math.cos(x0);
Motor3y=d3*Math.sin(x0+x1)+d2*Math.sin(x0);

Motor4x=d3*Math.cos(x0+x1)+d2*Math.cos(x0);
Motor4y=d3*Math.sin(x0+x1)+d2*Math.sin(x0);
}
```

B.12. Determinación del error

```
// Esta función es llamada por medio de un botón que se muestra después de la
simulación
public void error () {

// División de los errores por el número de muestras (i)
ErrorX/=i;
ErrorY/=i;

// Suma de los errores, se obtiene el promedio para encontrar el errorTotal
ErrorT= (ErrorX + ErrorY)/2;
}
```

B.13. Ángulos entre articulaciones

```
public void angulo () {
//Esta función calcula el ángulo que hay entre la articulación 1,2 y 2,3.

//La posición 1 es 0,0 la posición de la segunda articulación esta en Motor2x y
//Motor2y
//Además la hipotenusa también es constante por que es la extensión del brazo

Angulo12=Math.atan2(Motor2x,Motor2y);

double varx,vary;

varx=Motor2x-Motor3x;
vary=Motor2y-Motor3y;

Angulo23=Math.atan2(varx,vary);
}
```

B.14. Configuración para una nueva simulación

```
public void reiniciar () {
i=0;
temp=0;

if(circunferencia==true)
{
```

```
        xent=tamRobot*0.02*Math.sin(6.2831853/3*temp)+inix;
        yent=tamRobot*0.02*Math.cos(6.2831853/3*temp)+iniy;
}
// Para las otras trayectorias
else
{
        xent=inix;
        yent=iniy;
}

// Esto si es común
        zent=iniz;
        xsal=xent;
        ysal=yent;
        zsal=zent;

// Cálculo de las posiciones articulares dependiendo de los datos obtenidos
        MGI();

// Se actualizan los estados
        x0=q[0];
        x1=q[1];
        x2=q[2];
        x3=q[3];

//Selección del tiempo de simulación dependiendo de la trayectoria
if(linea==true)
        tiempo=1000;
else if(circunferencia==true)
        tiempo=3000;
else if(triangle==true)
        tiempo=4000;
else if(pick==true)
        tiempo=5000;

//Velocidades en cero
        x4=x5=x6=x7=0;
//Se da el valor al tamaño
        tamRobot=tamV;

QDP1=QDP2=QDP3=QDP4=0;
```

```
extex12=extey12=extex23=extey23=0;

ErrorX=ErrorY=0;

_view.resetTraces();
_initialize();//Tomar los valores de la vista
_play();
}
```

B.15. Sección inicialización

```
// Se toman los valores de la vista para las constantes del control

i=0;
temp=0;

if(MostrarPID==true)//Constantes para el control PID
{
        Kp[0]=Kp1;
        Kp[1]=Kp2;
        Kp[2]=Kp3;
        Kp[3]=Kp4;

        Kv[0]=Kv1;
        Kv[1]=Kv2;
        Kv[2]=Kv3;
        Kv[3]=Kv4;
}
else //Constantes para el control CTC
{
        Kp[0]=cKp1;
        Kp[1]=cKp2;
        Kp[2]=cKp3;
        Kp[3]=cKp4;

        Kv[0]=cKd1;
        Kv[1]=cKd2;
        Kv[2]=cKd3;
        Kv[3]=cKd4;
}
        Ki[0]=Ki1;
```

```
        Ki[1]=Ki2;
        Ki[2]=Ki3;
        Ki[3]=Ki4;

mensajes();

//Dependiendo de la posición cartesiana inicial se calcula la posición articular. La
única trayectoria que presenta diferentes puntos que los dados en la vista es la
trayectoria circular, ya que a esta se le da el centro y no el punto de partida, por
eso se calcula aparte

if(circunferencia==true)
{
        xent=tamRobot*0.02*Math.sin(6.2831853/3*temp)+inix;
        yent=tamRobot*0.02*Math.cos(6.2831853/3*temp)+iniy;
}
// Para las otras trayectorias
else
{
        xent=inix;
        yent=iniy;
}

// Esto si es común

        zent=iniz;
        xsal=xent;
        ysal=yent;
        zsal=zent;

// Ahora se calculan las posiciones articulares dependiendo de los datos obtenidos
        MGI();

// Se actualizan los estados
        x0=q[0];
        x1=q[1];
        x2=q[2];
        x3=q[3];

// Selección del tiempo de simulación dependiendo de la trayectoria
if(linea==true)
```

```
        tiempo=1000;
else if(circunferencia==true)
        tiempo=3000;
else if(triangle==true)
        tiempo=4000;
else if(pick==true)
        tiempo=5000;

// Velocidades en cero
        x4=x5=x6=x7=0;
        // valor al tamaño
        tamRobot=tamV;

QDP1=QDP2=QDP3=QDP4=0;
extex12=extey12=extex23=extey23=0;

ErrorX=ErrorY=0;

for(int a=0;a<=3;a++)
{
        q[a]=0;
        qp[a]=0;
        Torque[a]=0;
        IntRes[a]=0;
        qres[a]=0;
}

MGI();
if(MostrarPID==true)
        PID();
else
        CTC();
modelo();
salida();
angulo();

link1= ((ControlElement3D) _view.getElement("Link1")).getElement();
transformFixed = Matrix3DTransformation.rotationX(-1.57);

link12=((ControlElement3D)_view.getElement("Link12")).getElement();
link2=((ControlElement3D)_view.getElement("Link2")).getElement();
```

```
link23=((ControlElement3D)_view.getElement("Link23")).getElement();

//Pasos que tienen que pasar para que haya una actualización en el modelo virtual
_setStepsPerDisplay(15);
```

B.16. Sección ligaduras

```
// Primero se selecciona la trayectoria a seguir por el Scara
if (linea==true)
        trayectoria();
else if (circunferencia==true)
        circular();
else if(pick==true)
        pickup();
else if(triangle==true)
        triangulo();

// Ahora se llaman el resto de funciones
MGI();
if(MostrarPID==true)
        PID();
else
        CTC();
modelo();
salida();
angulo();

// Tiempo de simulación
if(i>=tiempo)_pause();

//////////////////////////Configuración de los elementos 3D///////////////

//Angulo para la primera articulación
Matrix3DTransformation transformTheta = Matrix3DTransformation.rotationZ(-
Angulo12);

Matrix3DTransformation transformLink1 =
  (Matrix3DTransformation) transformTheta.clone();
transformLink1.multiply((Matrix3DTransformation)transformFixed);
((Element)link1).setTransformation(transformLink1);
```

```
// Transformación para la extensión del enlace 1
double[] array=new double[]{0,0,d2/2};
((Element) link1).toSpaceFrame(array);
extex12=array[0];
extey12=array[1];

Matrix3DTransformation transformLink12 =
  (Matrix3DTransformation) transformTheta.clone();
transformLink12.multiply ((Matrix3DTransformation)transformFixed);
((Element) link12).setTransformation(transformLink12);
// This is required for arm.toSpaceFrame() below to work properly
((Element) link12).setXYZ(array);

////////////////////////////////////////////////////////////////////////////////////////

// Angulo para la segunda articulación
Matrix3DTransformation transformBeta= Matrix3DTransformation.rotationZ(-
Angulo23);

Matrix3DTransformation transformLink2=
(Matrix3DTransformation)transformBeta.clone();
transformLink2.multiply((Matrix3DTransformation)transformFixed);
((Element)link2).setTransformation(transformLink2);

// Transformación para la extensión de el enlace 2
array=new double[]{0,0,-d3/2};
((Element) link2).toSpaceFrame(array);
extex23=array[0];
extey23=array[1];

Matrix3DTransformation transformLink23 =
  (Matrix3DTransformation) transformBeta.clone();
transformLink23.multiply ((Matrix3DTransformation)transformFixed);
((Element) link23).setTransformation(transformLink23);
// This is required for arm.toSpaceFrame() below to work properly
((Element) link23).setXYZ(array);
```

B.17. Sección evolución

***Variable independiente**: temp;*
***Incremento**: delta;*

//Estados

- *dX0/dtemp = x4;*
- *dX1/dtemp=x5;*
- *dX2/dtemp=x6;*
- *dX3/dtemp=x7;*
- *dX4/dtemp=QDP1;*
- *dx5/dtemp=QDP2;*
- *dX6/dtemp=QDP3;*
- *dX7/dtemp=QDP4;*

***Método**: Euler-Richardson(punto medio)*

B.18. Lista de variables definidas como globales

```
public double QDP1  = 0.0; // Variables.Modelo:1
public double QDP2  = 0.0; // Variables.Modelo:2
public double QDP3  = 0.0; // Variables.Modelo:3
public double QDP4  = 0.0; // Variables.Modelo:4
public double Torque []; // Variables.Modelo:5
public double x0  = 0; // Variables.Modelo:6
public double x1  = 0; // Variables.Modelo:7
public double x2  = 0; // Variables.Modelo:8
public double x3  = 0; // Variables.Modelo:9
public double x4  = 0.0; // Variables.Modelo:10
public double x5  = 0.0; // Variables.Modelo:11
public double x6  = 0.0; // Variables.Modelo:12
public double x7  = 0.0; // Variables.Modelo:13
public double d2  = 0.5; // Variables.Robot virtual:1
public double d3  = 0.3; // Variables.Robot virtual:2
public double r4  = 0.2; // Variables.Robot virtual:3
public Object link1  = null; // Variables.Robot virtual:4
public Object link12  = null; // Variables.Robot virtual:5
public Object link2  = null; // Variables.Robot virtual:6
public Object link23  = null; // Variables.Robot virtual:7
public double extex12  = 0; // Variables.Robot virtual:8
public double extey12  = 0.0; // Variables.Robot virtual:9
public double extex23  = 0.0; // Variables.Robot virtual:10
public double extey23  = 0.0; // Variables.Robot virtual:11
public Object transformFixed  = null; // Variables.Robot virtual:12
public double h1  = 0.5; // Variables.Robot virtual:13
public double h2  = 0.15; // Variables.Robot virtual:14
public double h3  = 0.15; // Variables.Robot virtual:15
public double h4  = 0.2; // Variables.Robot virtual:16
public double Kp1  = 210000.0; // Variables.Robot virtual:17
public double Kp2  = 80000; // Variables.Robot virtual:18
```

```
public double Kp3  = 40000; // Variables.Robot virtual:19
public double Kp4  = 70000; // Variables.Robot virtual:20
public double Ki1  = 1000; // Variables.Robot virtual:21
public double Ki2  = 1000; // Variables.Robot virtual:22
public double Ki3  = 1000; // Variables.Robot virtual:23
public double Ki4  = 1000; // Variables.Robot virtual:24
public double Kv1  = 1500; // Variables.Robot virtual:25
public double Kv2  = 200; // Variables.Robot virtual:26
public double Kv3  = 60; // Variables.Robot virtual:27
public double Kv4  = 500; // Variables.Robot virtual:28
public double inix  = 0.35; // Variables.Robot virtual:29
public double iniy  = 0.35; // Variables.Robot virtual:30
public double iniz  = 0.3; // Variables.Robot virtual:31
public double tamV  = 1.0; // Variables.Robot virtual:32
public double tamRobot  = 1.0; // Variables.Robot virtual:33
public double cKp1  = 1500000.0; // Variables.Robot virtual:34
public double cKp2  = 390; // Variables.Robot virtual:35
public double cKp3  = 390; // Variables.Robot virtual:36
public double cKp4  = 390; // Variables.Robot virtual:37
public double cKd1  = 50; // Variables.Robot virtual:38
public double cKd2  = 50; // Variables.Robot virtual:39
public double cKd3  = 50; // Variables.Robot virtual:40
public double cKd4  = 50; // Variables.Robot virtual:41
public double areaTrabajo  = 0.0; // Variables.Robot virtual:42
public double xent  = 0.35; // Variables.Trayectoria:1
public double yent  = 0.35; // Variables.Trayectoria:2
public double zent  = 0.35; // Variables.Trayectoria:3
public double xsal  = 0.0; // Variables.Trayectoria:4
public double ysal  = 0.0; // Variables.Trayectoria:5
public double zsal  = 0.0; // Variables.Trayectoria:6
public double q []; // Variables.Trayectoria:7
public double qp []; // Variables.Trayectoria:8
public double temp  = 0.0; // Variables.Generales:1
public double delta  = 0.001; // Variables.Generales:2
public double i  = 0.0; // Variables.Generales:3
public boolean circunferencia  = false; // Variables.Generales:4
public boolean linea  = true; // Variables.Generales:5
public boolean triangle  = false; // Variables.Generales:6
public boolean pick  = false; // Variables.Generales:7
public double tiempo  = 0; // Variables.Generales:8
public boolean MostrarPID  = true; // Variables.Generales:9
public boolean MostrarCTC  = false; // Variables.Generales:10
public double ErrorX  = 0; // Variables.Generales:11
public double ErrorY  = 0; // Variables.Generales:12
public double ErrorT  = 0.0; // Variables.Generales:13
public double resultado  = 0.0; // Variables.Generales:14
public double Kp []; // Variables.PID:1
public double Ki []; // Variables.PID:2
```

```
public double Kv []; // Variables.PID:3
public double qant1 []; // Variables.Derivativa:1
public double qact []; // Variables.Derivativa:2
public double qres []; // Variables.Derivativa:3
public double IntAnt []; // Variables.Integral:1
public double IntRes []; // Variables.Integral:2
public double Motor1x  = 0.0; // Variables.Motores:1
public double Motor1y  = 0.0; // Variables.Motores:2
public double Motor2x  = 0.0; // Variables.Motores:3
public double Motor2y  = 0.0; // Variables.Motores:4
public double Motor3x  = 0.0; // Variables.Motores:5
public double Motor3y  = 0.0; // Variables.Motores:6
public double Motor4x  = 0.0; // Variables.Motores:7
public double Motor4y  = 0.0; // Variables.Motores:8
public double Angulo12  = 0.0; // Variables.Angulos:1
public double Angulo23  = 0.0; // Variables.Angulos:2
public double G3  = 9.81; // Variables.ModeloVE:1
public double M4  = 1.8; // Variables.ModeloVE:2
public double ZZ1R  = 3.38; // Variables.ModeloVE:3
public double ZZ2R  = 0.063; // Variables.ModeloVE:4
public double ZZ3R  = 0.1; // Variables.ModeloVE:5
public double MX2R  = 2*0.121; // Variables.ModeloVE:6
public double MX3R  = 0.2; // Variables.ModeloVE:7
public double MY3R  = 0.1; // Variables.ModeloVE:8
public double MY2  = 0.001; // Variables.ModeloVE:9
public double FV1  = 0.1; // Variables.ModeloVE:10
public double FV2  = 0.012; // Variables.ModeloVE:11
public double FV3  = FV1; // Variables.ModeloVE:12
public double FV4  = FV2; // Variables.ModeloVE:13
public double FS1  = 0.57; // Variables.ModeloVE:14
public double FS2  = 0.125; // Variables.ModeloVE:15
public double FS3  = FS1; // Variables.ModeloVE:16
public double FS4  = FS2; // Variables.ModeloVE:17
public double IA2  = 0.0; // Variables.ModeloVE:18
public double CZ1  = 0.0; // Variables.ModeloVE:19
public double FX2  = 0.0; // Variables.ModeloVE:20
public double FY2  = 0.0; // Variables.ModeloVE:21
public double FX3  = 0.0; // Variables.ModeloVE:22
public double FY3  = 0.0; // Variables.ModeloVE:23
public double FZ3  = 0.0; // Variables.ModeloVE:24
public double CZ3  = 0.0; // Variables.ModeloVE:25
public double CZ4  = 0.0; // Variables.ModeloVE:26
public double CX3  = 0.0; // Variables.ModeloVE:27
public double CX4  = 0.0; // Variables.ModeloVE:28
public double CY3  = 0.0; // Variables.ModeloVE:29
public double CY4  = 0.0; // Variables.ModeloVE:30
public double FX4  = 0.0; // Variables.ModeloVE:31
public double FY4  = 0.0; // Variables.ModeloVE:32
```

```
public double FZ4  = 0.0; // Variables.ModeloVE:33
public double IA3  = 0.0; // Variables.ModeloVE:34
public double IA4  = 0.045; // Variables.ModeloVE:35
```

ANEXO C

MANUAL DE LA SIMULACIÓN CREADA EN EJS

El manual se encuentra divido en dos secciones dependiendo de las necesidades del usuario. La primera sección se centra en las necesidades que tienen las personas que quieren usar la simulación tal como esta, solo desean cambiar los parametros de la como trayectoria, posicion inicial o tipo de controlador, además de sintonizar los controladores obviamente. Para este tipo de uso se creará el “Manual de usuario”.

La segunda sección es para los usuarios con necesidades más exigentes, los cuales desean cambiar el codigo de la simulación introduciendo nuevos tipos de control, nuevas trayectorias, etc. Para este tipo de uso se crerá el “Manual del programador”.

C.1. Manual de usuario

El diseño de la simulacion virtual del robot SCARA se creó para que fuera lo más fácil y simple que pueda ser. Cuando se ejecuta la aplicación, aparecen varias ventanas y dialogos que permiten realizar muchas acciones, todas ellas necesarias.

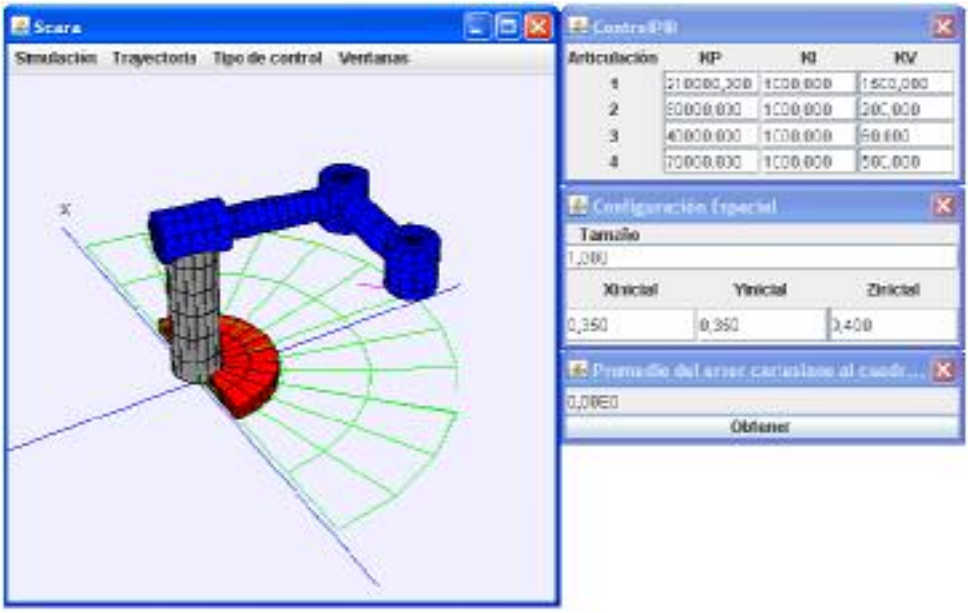

Figura 1. Vista inicial al ejecutar la aplicación.

Al inicio de la simulación 4 ventanas estan activas, ver Figura 1. Una para mostrar el robot virtual. Las otras tres se encuentran al lado derecho de la simulación. En orden descendente ellas son, la configuración de PID, configuración de la posición inicial y tamaño, y una ventana que permite obtener el promedio del error cartesiano cal cuadrado, el cual fue el parámetro de validación. Cada una de ellas se describirá acontinuación.

C.1.1. Ventana Principal

Además de contener el modelo virtual del SCARA, esta ventana esta provista del control de la simulación, selección de la trayectoria, selección del tipo de control y además se tiene una opción para poder ocultar o mostrar las otras ventanas de acuerdo a las necesidades de los usuarios. Como esta es la ventana principal de la aplicación, al cerrar esta se da por hecho de que se quiere terminar la ejecución de la simulación y se cerrará el programa. Esto no sucede con ninguna de las otras ventanas, ver Figura 2.

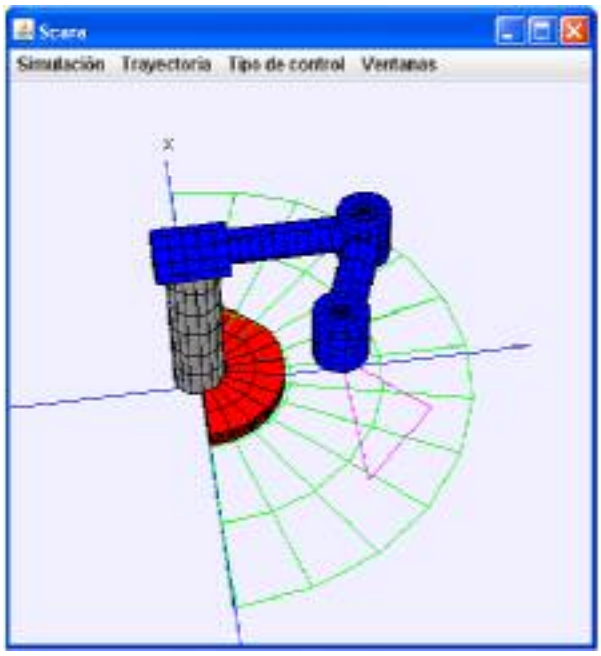

Figura 2. Ventana principal de la aplicación.

Es posible dar click izquierdo sobre el mundo virtual y luego arrastrar el mouse para cambiar de angúlo de visión sobre el modelo. Para que el usuario tenga una pequeña guía para ese tipo de movimiento se tiene un eje tridimensional indicando la dirección de cada uno de ellos.

Ahora se explicara cada uno de los items que estan en la barra de Menú.

C.1.2. Simulación

Este menú contiene las opciones que tienen el control de ejecución de la simulación, ver Figura 3.

a. **Play** permite continuar con la ejecución después de que se haya parado e sistema mediante pause.

b. Pause: Esta opción permite detener la ejecución de la aplicación en cualquier momento.

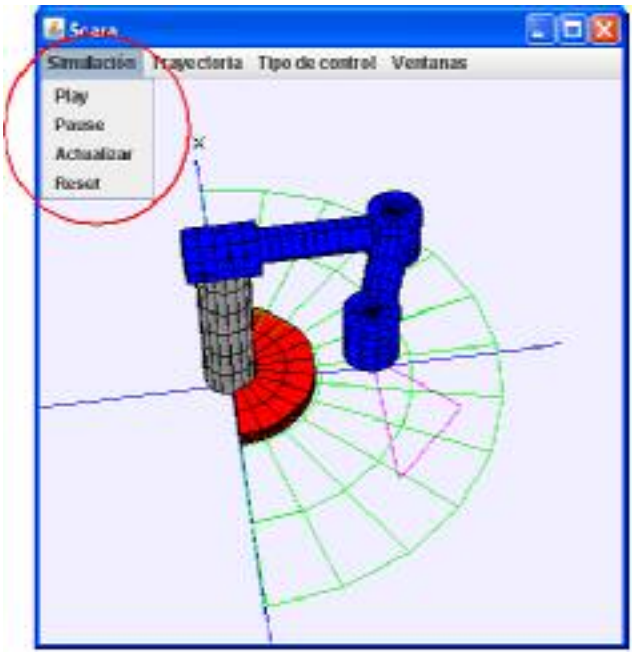

Figura 3. Menu simulación.

c. Actualizar permite reiniciar la aplicación con los datos que el usuario ha configurado, sea que cambio el tipo de trayectoria o los parámetros del controlador, todos estos cambios son tomados para congirurar y reiniciar la simulación. Esta opción se desabilita cada vez que la simulación esta en ejecución. Para activarla es ecesario detener la simulación o esperar a que termine sola.

d. Reset la función de esta opción es reiniciar la aplicación cargando los valores predeterminados en la simulación.

C.1.3. Trayectoria

Mediante este menu es posible escoger entre los cuatro tipos de trayectorias que se han predefinido en la aplicación, ver figura 4.

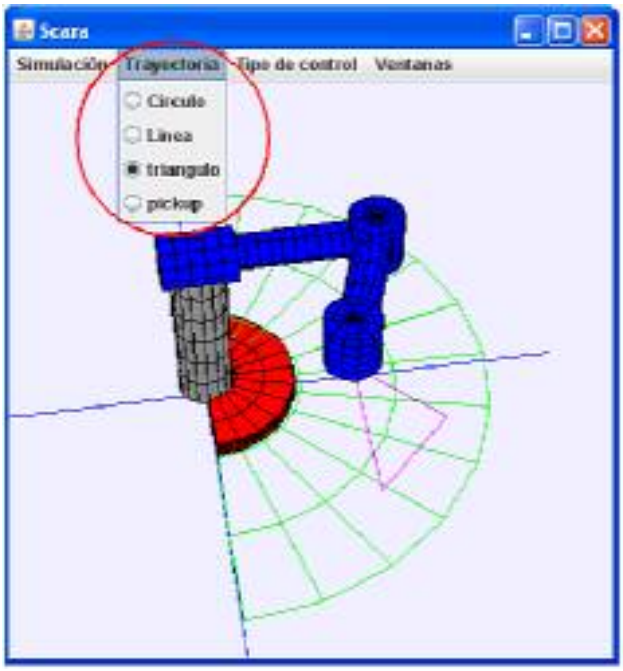

Figura 4. Menu Trayectoria.

Realmentemente los nombres son muy descriptivos y no merecen mayor explicación. La opción de pick-and-place es una trayectoria que simula el recorrido de levantar un objeto de la posicion inicial y llevarlo hasta otro sitio, especificado por la trayectoria y no por el usuario.

C.1.4. Tipo de control

Permite escoger el tipo de control con el cual se quiere trabajar, los dos controladores con los que se trabajaron fueron el PID y el CTC, ver Figura 5.

Cada vez que una persona escoja el tipo de control, la ventana de configuración de este aparecerá de igual manera que al control no seleccionado se le esconde su ventana de configuración. Esta característica se implemento para que no hubieran ventanas con información innecesaria en determinado momento.

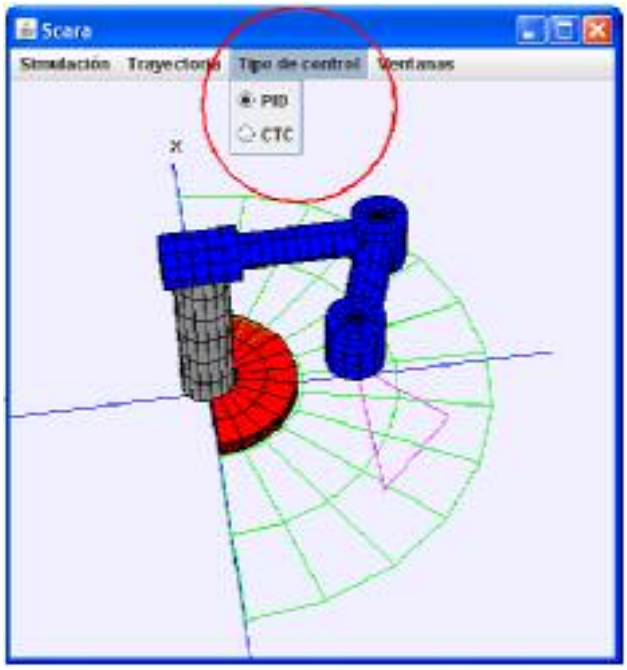

Figura 5. Selección del tipo de control.

C.1.5. Ventanas

Para facilitar la navegación por el espacio de trabajo del usuario se creo la forma de que pudieran ocultar o mostrar las ventanas que ofrecen información de la simulación, ver Figura 6.

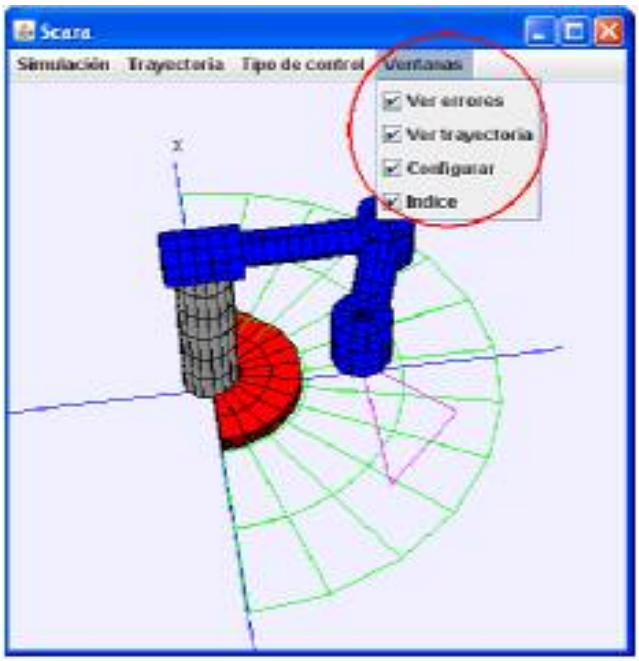

Figura 6. Selección de ventanas que serán visibles.

Ver errores y ver trayectoria permiten esconder ó mostrar las ventanas con las información de los errores articulares y cartesianos, y la trayectoria obtenida. Más adelante se mostrará la información que ofrece cda una de estas ventanas

C.1.6. Ventana configuración del PID

Por medio de este dialogo es posible configurar las constantes del controlador PID. Cada uno de los campos numericos esta relacionado internamente con las variables que sintonizan el controlador, ver Figura 7.

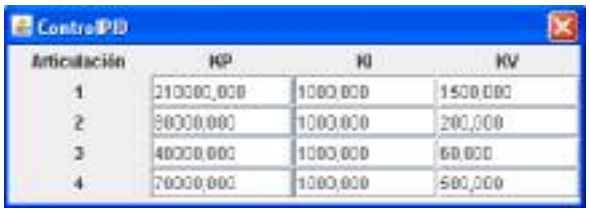

Figura 7. Dialogo de configuración del controlador PID.

La columna que se encuentra más a la izquierda muestra la articulación y la fila superior tiene los nombres de la variable. De esta forma es fácil cambiar algún valor del controlador. Por ejemplo si se quisiera cambiar la constante integral de la tercera articualción se tendría que cambiar el campo númerico resaltado en amarillo. Para cambiar un valor se debe dar click en el campo, editar la información, lo que hará que el campo se vuleva amarillo y presionar enter para aceptar el cambio. Si luego de editar la casilla no se acepta el cambio, no se tomará en cuenta al actualizar la simulación, ver Figura 8.

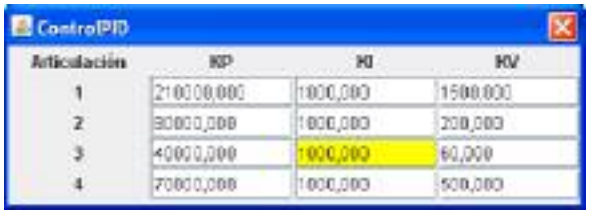

Figura 8. Campo númerico relacionado con el valor de la constante integral de la tercera articulación.

Par tener en cuenta, esta ventana aparecerá o desaparecerá cada vez que se cambie el tipo de controlador. Aún si el usuario cerro el dialogo por accidente puede volverlo a abrir desde el menu “tipo de control”.

C.1.7. Ventana configuración de CTC

Este dialogo es similar al que configura el control PID. Dispone de 8 campos (4 variables Kp y 4 variables Kv), ver Figura 9.

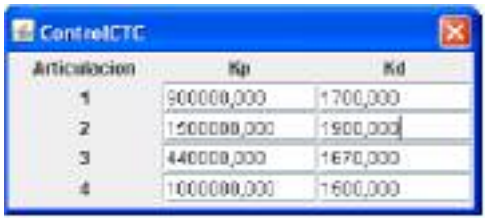

Figura 9. Cofiguración del CTC.

C.1.8. Ventana configuración espacial

Configurar la posición inicial y el tamaño de las trayectorias es posible usando este dialogo. La posición incial es expresada con las tres cordenadas cartesianas, se puede configurar cualquier punto siempre y cuando este dentro del espacio de trabajo del SCARA, ver Figura 10.

Figura 10. Configuración espacial.

Hay un campo que permite configurar el tamaño de la trayectoria. Por medio de este es posible hacer un cambio **proporcional** a las trayectorias. Por ejemplo, la trayectoria circular tiene normalmente 2 cm de radio, si se configura un tamaño igual a 2, el nuevo de

radio del trayecto es de 4cm. Esto funciona de la misma manera para cada una de las trayectorias.

C.1.9. Ventana índice de error

Al final de cada simulación es posible calcular el promedio del error cartesiano al cuadrado con solo dar click en el boton Obtener que tiene el dialogo. La forma como se calculó el indice ya se describío dentro del documento. El campo muestra el valor de forma científica, ver Figura 11.

Figura 11. Promedio del error cartesiano al cuadrado.

C.1.10. Ventana trayectoria

Muestra el desarrollo de la simulación en coordenadas cartesianas. En esta pantalla aparece la tryectoria deseada en verde y la obtenida en rojo. De esta manera es fácil determinar gráficamente los errores más grandes, ver Figura 12.

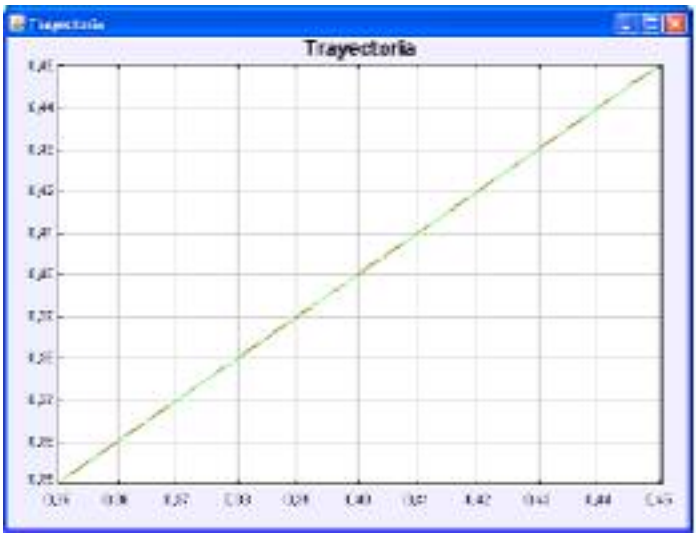

Figura 12. Trayectoria obtenida.

C.1.11. Ventana de error cartesiano

Además del promedio del error cartesiano al cuadrado, es posible obtener información del error cartesiano. Esta información surge de una simple resta entre los datos de entrada y salida, los resultados obtenidos son graficados, ver Figura 13.

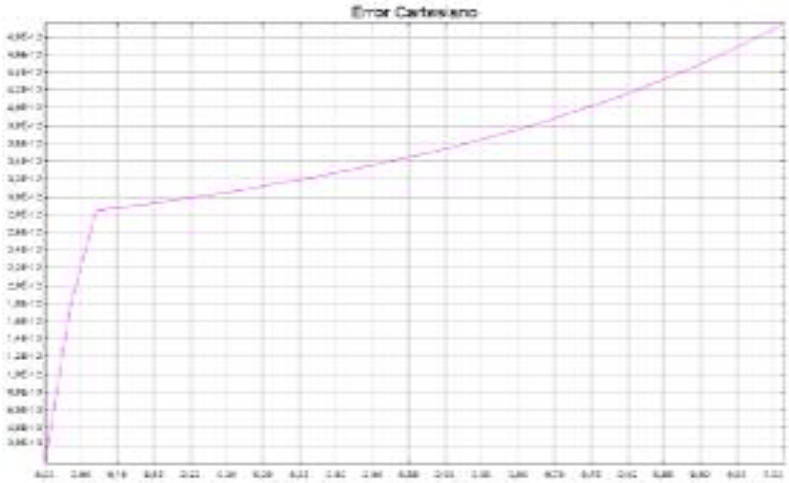

Figura 13. Error Cartesiano.

C.1.12. Ventana de error articular

El usuario podrá visualizar tambien mediante una ventana (ver Figura 14) el error articular del robot. Con esta información el diseñador puede tomar decisiones importantes sobre la sintonización del controlador y la precisión que tiene la aplicación en desarrollo.

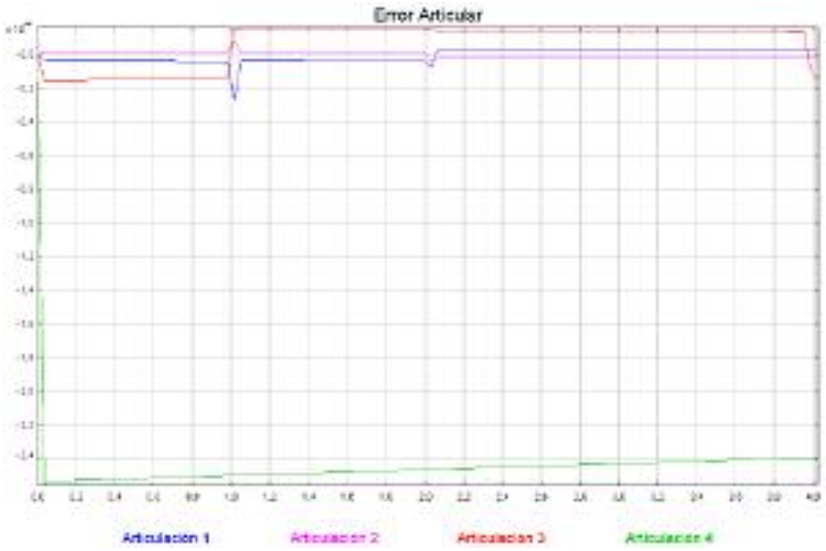

Figura 14. Error articular.

C.2. Manual del programador

Las necesidades de un programador, de una persona que desee ampliar las capacidades de la aplicación son las siguientes.

a. Añadir un nuevo controlador
b. Agregar una nueva trayectoria

En este manual no se explicará la forma de utilizar Easy Java, para ver en más detalle sobre este aspecto visitar la pagina principal de EJS.

C.2.1. Añadir un nuevo controlador

Básicamente para añadir un nuevo controlador se debe tener en cuenta que

a. El modelo del robot o la planta recibe la señal de torques, y tiene como salida la velocidad y la posicion en espacio articular.

b. Las posiciones obtenidas se guardan en las variables x0,x1,x2,x3 y las velocidades en las variables x4,x5,x6,x7.

c. La implementación actual solo permite realizar una solo derivada a una señal e igualmente una integral. Esto es debido a que cada una de estas funciones utiliza un vector en donde se guardan los resultados. Si se llamará más a una de ellas más de una vez por ciclo se perdería la información del anterior llamado.

Para construir un controlador se debe implementar su codigo dentro de la sección propio. De ser necesario variables globales se pueden añadir en la sección Variables, sea dentro de una pestaña ya creada o creando una nueva, depende del criterio del programador.

Uc es la señal de entrada la cual se obtiene del vector q[], este vector es creado por la la función MGI() después de recibir la posicion deseada en coordenadas cartesianas.

Utilizando la función derivar(q[]) se puede estimar la velocidad deseada.

Teniendo tanto la velocidad como la posición depende del programador el control y el código que desea implementar. No importa como lo plantee el programador la función servirá. El resultado del controlador debe ser gurdado en el vector Torque[] para que la siguiente función en ligaduras, el modelo, pueda recibir y procesar la señal de control.

Esto en cuanto al controlador. Sin embargo para que la función sea tomada por el programa debe añadirse en la inicialización, en ligaduras y en la vista.

1. En la sección inicialización es necesario añadir el nuevo controlador en la siguiente parte.

```
if(MostrarPID==true)
  PID();
else
  CTC();
```

MostrarPID es una variable global booleana que esta conectada al menu “tipo de control” de la ventana principal y por medio de el se hace invisible o visible la ventana de configuración del PID. De esta manera, el programador crea una variable boleana como MostrarDifuso (para un controlador difuso por ejemplo). Para poder añadirlo de la siguiente forma.

```
if(MostrarPID==true)
  PID();
else if(MostrarCTC==true)
  CTC();
Else if(MostrarDifuso==true)
  Difuso();
```

Algo para tener en cuenta es que al momento de crear las variables boleanas para el controlador se deje una por defecto en verdadero y las otras en false. Esto evita que se puedan tener dos controladores funcionen al mismo tiempo, aunque se pueden configurar desde la vista lo mejor es realizar este simple paso.

En el programa actual existen 12 variables para ajustar los controladores ellos son 3 vectores de 4 posiciones (de 0 a 3) Kp, Ki y Kv. El nuevo controlador puede usar algunas o todas estas variables para sintonizar el controlador. No existe razón alguna para no crear unas variables propias.

Para que el usuario pueda configurar el nuevo controlador se debe utilizar unas variables para la vista y otras para el funcionamiento interno. ¿Por qué? Porque de esta manera solo se toman los valores cada vez que se inicia una nueva simulación y no se permiten cambios mientras esta en una simulación.

```
Kp[0]=Kp1;
Kp[1]=Kp2;
Kp[2]=Kp3;
Kp[3]=Kp4;

Kv[0]=Kv1;
Kv[1]=Kv2;
Kv[2]=Kv3;
Kv[3]=Kv4;
```

En esta caso del controlador PID. En la ventana de configuración cada cmapo numerico esta realcionado con una variable. La constante Kp de la primera articulación, por ejemplo, esta relacionada en la vista con Kp1 pero internamente se encuentra relacionada con Kp[0]. De esta manera cada vez que se actualiza el programa Kp[0] toma el valor de la vista, el introducido por el usuario, y no mientras este corriendo una simulación.

En este caso todas las variables tanto las internas (derecha) como las de la vista (izquierda) son globales, declaradas en la sección variables.

2. En la sección ligaduras en indispensable realizar el mismo paso, exceptuando la configuración de las variables.

3. Lo más probable es que sea necesario crear una nueva interfaz para poder configurar el nuevo controlador. Lo más simple es crear un dialogo e ir insertando los elementos necesarios en ella (titulos, campos numéricos, botones, etc).

Luego de crear la ventana se deben crear un control en el menu tipo de control en la ventana principal. Asegurarse que la nueva opción es un radiobutton para que solo sea posible seleccionar un tipo de control en cada momento, ver Figura 5.

El nuevo radiobutton debe tener en las opciones (haciendo doble click sobre la opción en vista) como Variable MostrarDifuso. En la casilla valor debe ir el valor por defecto. En la simulación creada se encuentra seleccionado el PID como controlador por defecto, esto es verdadero, así que el nuevo control deberá tener el valor de falso para no tener problemas.

La ventana debe ser configurada de la siguiente forma para que pueda aparecer o desaparecer al seleccionar o no el control respectivamente, ver Figura 15.

Figura 15. Configuración de la ventana.

La grafica x. es la configuración de la posible ventana que se necesitará para configurar un controlador difúso. El circulo en rojo muestra la parte donde se debe agregar la variable boleana para que el dialogo pueda aparecer cuando sea seleccionado.

Con estos cambios es posbile ejecutar la aplicación y utilizar el nuevo controlador que se ha creado.

C.2.2. Añadir una nueva trayectoria.

Para añadir un nuevo trayecto es un poco más sencillo que agregar un controlador. Primero se deben tener en cuenta las restricciones en cuanto al espacio de trabajo que tiene un SCARA.

Lo más importante es crear la trayectoria en ecuaciones, las que dispone actualmente el modelo son ecuaciones en función de la variable temp, que es la que lleva el tiempo de la simulación. Por ejemplo la función que crea el movimiento en X de la trayectoria lineal es:

```
xent=tamRobot*temp/10 + inix;
```

Con esta ecuación se pueden observar tres cosas

1. Las coordenadas deseadas son guardadas en xent, yent y zent.
2. Con tamRobot es posible dejar que la trayectoria sea cambiada de tamaño proporcionalmente.
3. Inix, iniy e iniz contiene la posición inicial deseada del recorrido.

Las trayectorias se calculan paso a paso de la simulación en función de la variable temp como se dijo anteriormente. Es importante entonces agregar la función de la trayectoria dentro de la sección ligaduras.

```
// Primero se selecciona la trayectoria a seguir por el Scara
if (linea==true)
  trayectoria();
else if (circunferencia==true)
  circular();
else if(pick==true)
  pickup();
else if(triangle==true)
  triangulo();
```

De igual manera que con el controlador, se hace necesario crear una variable boleana que permita escoger la trayectoria desde la vista. Suponiendo que la trayectoria se una rectangulo, se creará una variable boleana rect, la cual se agregara en ligaduras para poder llamar la función en cada paso y así determinar cual es el nuevo punto de la trayectoria.

```
// Primero se selecciona la trayectoria a seguir por el Scara
if (linea==true)
        trayectoria();
else if (circunferencia==true)
        circular();
else if(pick==true)
        pickup();
else if(triangle==true)
        triangulo();
else if(rect==true)
        rectangulo();
```

Es importante que se establezca que tiempo de simulación es necesario para ejecutar la totalidad de la trayectoria. Para calcular este se debe tener en cuenta que el tiempo de muestreo por defecto es de 1 milisegundo.

Para el caso de la trayectoria lineal se tiene 1 segundo, ya que se quiere que haya 1000 puntos entre 0.35 y 0.45.

```
xent=tamRobot*temp/10 + inix;
```

La división entre 10 es para que cada paso de simulación aumente en 0.0001cm y no en 1mm por ciclo y no sobrepase los 0.45m.

Calculando así la cantidad de ciclos que se requieren para completar la trayectoria es posible obtener el tiempo de simulación necesario. Este tiempo debe ser configurado en la siguiente sección del código de inicialización.

```
//seleccion del tiempo de simulacion dependiendo de la trayectoria
if(linea==true)
        tiempo=1000;
else if(circunferencia==true)
        tiempo=3000;
else if(triangle==true)
        tiempo=4000;
else if(pick==true)
        tiempo=5000;
```

Suponiendo que el tiempo necesario para terminar el recorrido es de 2 segundos se convierte a milisegundos.

```
//seleccion del tiempo de simulacion dependiendo de la trayectoria
if(linea==true)
        tiempo=1000;
else if(circunferencia==true)
        tiempo=3000;
else if(triangle==true)
        tiempo=4000;
else if(pick==true)
        tiempo=5000;
else if(rect==true)
        tiempo=2000;
```

Con esta breve pero completa explicación es posible agregar una nueva trayectoria a la simulación.