UNIVERSIDAD TECNOLÓGICA NACIONAL – FRBA

INGENIERÍA ELECTRÓNICA

Asignatura: Robótica
Curso: R6051

Código: 95-0482
Año: 2011

Tesis Final:

Plataforma Stewart

Alumnos:

Cignoli, Rodolfo
Pietrasanta, Agustín

Profesor: Mas. Ing. GIANNETA, Hernán

JTP: Ing. GRANZELLA, Damián

Primera Parte: Análisis Cinemático

Segunda Parte: Análisis Dinámico

Tercera Parte: Compilador

Primera parte: Análisis cinemático.

Objetivos:

- Obtención del modelo cinemático inverso de una plataforma Stewart genérica.
- Implementación en Matlab.
- Implementación en la familia DSP Freescale 56800E, mediante el IDE de CodeWarrior.

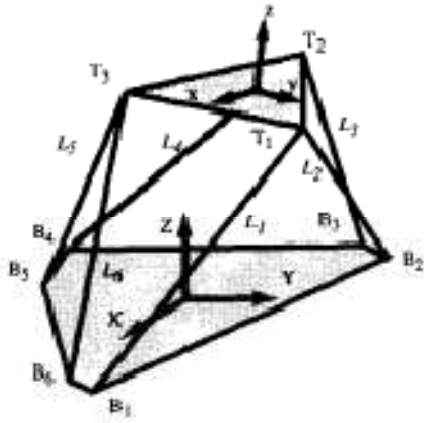
Introducción:

La plataforma Stewart tiene dos características fundamentales que la separan de los robots industriales. Posee seis actuadores lineales directamente conectados a una base móvil. Esta unión es simple y en forma cerrada, y con ella se logra un sistema mucho más rígido en proporción al tamaño y peso de cualquier manipulador de tipo cadena abierta. Por ser un sistema de cadena cerrada tiene muchas más restricciones de movimiento. Pero a la vez la fuerza de los seis actuadores se complementa obteniendo así una gran fuerza.

Cinemática Inversa y Modelado Físico:

En este trabajo, nuestro interés se centra en la configuración de tipo Stewart que utiliza seis actuadores lineales idénticos. La "Base" es un hexágono semi-regular. La parte superior "Top" es un triangulo equilátero. Los actuadores lineales son las "Piernas", y están conectadas a los vértices de la Base y Top con articulaciones de 2 a 3 grados de libertad. Con esto se logra un sistema completo de 6 grados de libertad, es decir, traslación en los ejes X,Y,Z y rotación alrededor de los mismos.

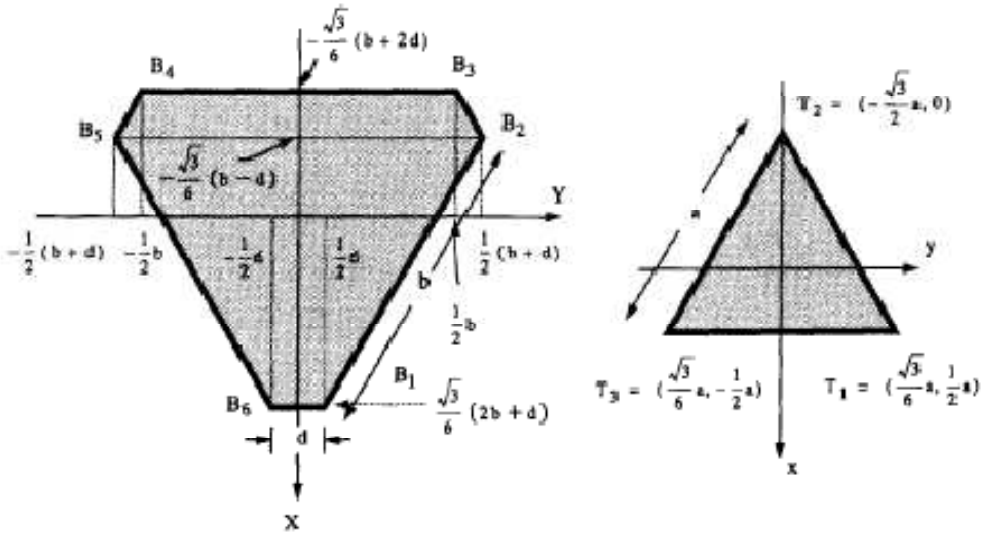
El marco de referencia (X,Y,Z) se encuentra en el centro de la Base con el eje Z apuntado verticalmente hacia arriba.



Sistema de referencia de la plataforma Stewart

Otro marco de referencia se sitúa en el centro de Top (x,y,z) en su centro de gravedad, con el eje z apuntando verticalmente.

A continuación se puede ver para la Base y para Top la relación de dimensiones.



Las longitudes de las piernas se denominarán **L_1, L_2, L_3, L_4, L_5 y L_6**, el origen del sistema Top se determina como **$[px,py,pz]^T$**, y las rotaciones respecto de cada eje **(alpha, beta, gamma),** donde alpha corresponde a rotación sobre el eje X, beta sobre el eje Y y gamma sobre el eje Z.

Entonces la posición y orientación de la plataforma Top queda definida por el vector **X = [px, py, pz, alpha, beta, gamma].**

Mediante el estudio de la cinemática inversa se encuentran las longitudes que deberán tener las piernas, para cumplir con la posición y orientación que solicite el vector X.

Las expresiones de las longitudes de las piernas se obtienen mediante el siguiente razonamiento, que hacemos para la primera, y con la misma lógica se extiende al resto.

$$L_1 = \sqrt{(X_{T1} - X_{B1})^2 + (Y_{T1} - Y_{B1})^2 + (Z_{T1} - Z_{B1})^2}$$

Donde $X_{T1} \dots Z_{Bn}$ se obtienen del análisis geométrico de la plataforma. Ver referencia [1] para su estudio completo.

$$L_1 = \sqrt{\left(X_{T1} - d/2\sqrt{3} - b/\sqrt{3}\right)^2 + \left(Y_{T1} - d/2\right)^2 + Z_{T1}^2}$$
$$L_2 = \sqrt{\left(X_{T1} - d/2\sqrt{3} + b/2\sqrt{3}\right)^2 + \left(Y_{T1} - d/2 - b/2\right)^2 + Z_{T1}^2}$$
$$L_3 = \sqrt{\left(X_{T2} + d/\sqrt{3} + b/2\sqrt{3}\right)^2 + \left(Y_{T2} - b/2\right)^2 + Z_{T2}^2}$$
$$L_4 = \sqrt{\left(X_{T2} + d/\sqrt{3} + b/2\sqrt{3}\right)^2 + \left(Y_{T2} + b/2\right)^2 + Z_{T2}^2}$$
$$L_5 = \sqrt{\left(X_{T3} - d/2\sqrt{3} + b/2\sqrt{3}\right)^2 + \left(Y_{T3} + b/2 + d/2\right)^2 + Z_{T3}^2}$$
$$L_6 = \sqrt{\left(X_{T3} - d/2\sqrt{3} - b/\sqrt{3}\right)^2 + \left(Y_{T3} + d/2\right)^2 + Z_{T3}^2}\,.$$

Implementación en Matlab:

Script principal GraficoStewart.m:

Define los parámetros dimensionales de la plataforma y permite que el usuario determine una trayectoria, configurando las traslaciones respecto a los ejes cartesianos de la base, y sus rotaciones correspondientes.

```
clc;
clear all;
syms alpha beta gamma ; %alpha: rot en x
                        %beta: rot en y
                        %gamma: rot en z

syms px py pz;  %Distancia al origen del TOP respecto de BASE

%Parametros BASE stewart
b=925;    %mm Lado largo de la base
d=150;   %mm Lado corto de la base
a=612.5;  %mm Lado TOP

px=0;
py=0;
pz=200;
alpha=0;
beta=0;
gamma=0;

for m = 1:10
%EJEMPLO DE TRAYECTORIA:
%Configuracion para traslacion
px=0;
py=0;
pz=pz+60;
%Configuracion para rotacion
alpha=0;
beta=beta+0.1;
gamma=gamma+0.1;
%-------------------------------
```

```
%Llamo a script Stewart.m, para el cálculo cinemático.
Stewart;

%Grafico de líneas de la base
t = 1:1:7;
%Puntos de la BASE
Xb = [Xb1 Xb2 Xb3 Xb4 Xb5 Xb6 Xb1];
Yb = [Yb1 Yb2 Yb3 Yb4 Yb5 Yb6 Yb1];
Zb = [Zb1 Zb2 Zb3 Zb4 Zb5 Zb6 Zb1];
plot3(Xb(t), Yb(t), Zb(t),'-')
hold on;
%Grafico de líneas de Top
t = 1:1:4;
Xt = [Xt1 Xt2 Xt3 Xt1];
Yt = [Yt1 Yt2 Yt3 Yt1];
Zt = [Zt1 Zt2 Zt3 Zt1];
plot3(Xt(t), Yt(t), Zt(t),'-')
axis([-600 600 -600 600 0 1200]);
grid on;
hold on;
end

hold off;

Xb1 = (sqrt(3)/6)*(2*b+d);
Yb1 = (1/2)*d;
Zb1 = 0;

Xb2 = -(sqrt(3)/6)*(b-d);
Yb2 = (1/2)*(b+d);
Zb2 = 0;

Xb3 = -(sqrt(3)/6)*(b+2*d);
Yb3 = (1/2)*b;
Zb3 = 0;

Xb4 = -(sqrt(3)/6)*(b+2*d);
Yb4 = -(1/2)*b;
Zb4 = 0;

Xb5 = -(sqrt(3)/6)*(b-d);
Yb5 = -(1/2)*(b+d);
Zb5 = 0;

Xb6 = (sqrt(3)/6)*(2*b+d);
Yb6 = -(1/2)*d;
Zb6 = 0;

%Con subindice en minuscula es respecto a TOP
%Con mAyuscula respecto a la base
xt1 = (sqrt(3)/6)*a;
yt1 = (1/2)*a;
zt1 = 0;
```

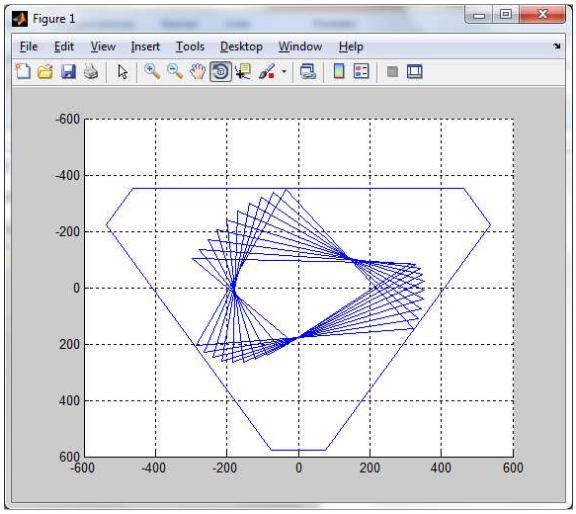
```
xt2 = -(sqrt(3)/3)*a;
yt2 = 0;
zt2 = 0;
```

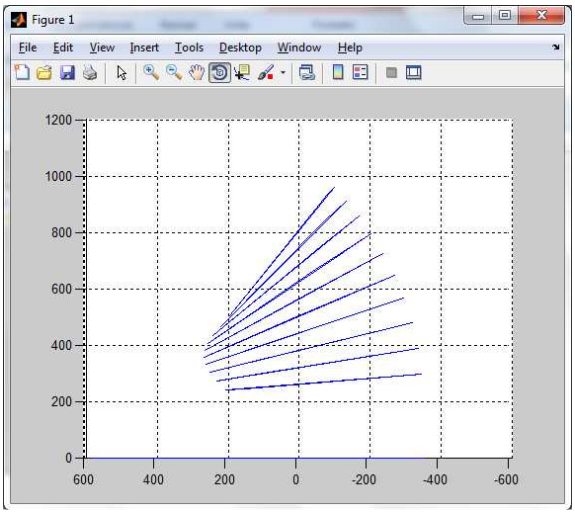
Script Stewart.m:

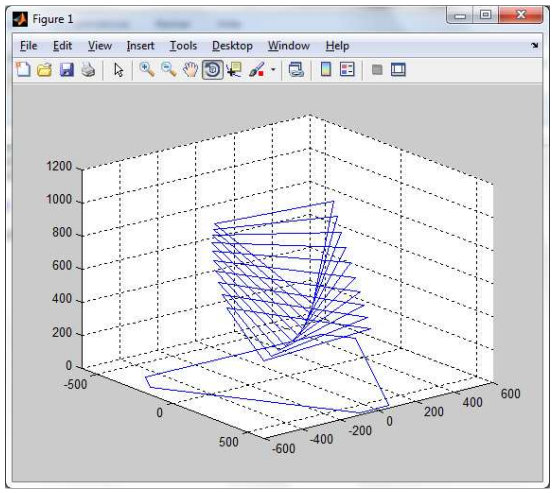
Genera la matriz transformación para obtener los desplazamientos de la plataforma TOP. Y calcula la longitudes de las piernas L1…L6 para cada punto de la trayectoria propuesta.

```
xt3 = (sqrt(3)/6)*a;
yt3 = -(1/2)*a;
zt3 = 0;

TTB = [cos(beta)*cos(gamma)+sin(alpha)*sin(beta)*sin(gamma) ...
        -cos(beta)*sin(gamma)+sin(alpha)*sin(beta)*cos(gamma) ...
        cos(alpha)*sin(beta) px;...

        cos(alpha)*sin(gamma)...
        cos(alpha)*cos(gamma)...
        -sin(alpha) py;...

        -sin(beta)*cos(gamma)+sin(alpha)*cos(beta)*sin(gamma) ...
         sin(beta)*sin(gamma)+sin(alpha)*cos(beta)*cos(gamma) ...
         cos(alpha)*cos(beta) pz;...

         0 0 0 1;...
        ];

XYZt1 = TTB * [xt1 yt1 zt1 1]';
XYZt2 = TTB * [xt2 yt2 zt2 1]';
XYZt3 = TTB * [xt3 yt3 zt3 1]';

Xt1=XYZt1(1);
Yt1=XYZt1(2);
Zt1=XYZt1(3);

Xt2=XYZt2(1);
Yt2=XYZt2(2);
Zt2=XYZt2(3);

Xt3=XYZt3(1);
Yt3=XYZt3(2);
Zt3=XYZt3(3);

L1 = sqrt((Xt1 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt1 - d/2)^2 + Zt1^2 );
L2 = sqrt((Xt1 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt1 - d/2 - b/2)^2 +
Zt1^2 );
L3 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 - b/2)^2 + Zt2^2 );
L4 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 + b/2)^2 + Zt2^2 );
L5 = sqrt((Xt3 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt3 + b/2 + d/2)^2 +
Zt3^2 );
L6 = sqrt((Xt3 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt3 + d/2)^2 + Zt3^2 );
```

Imágenes correspondientes a la trayectoria cargada en el script GraficoStewart.

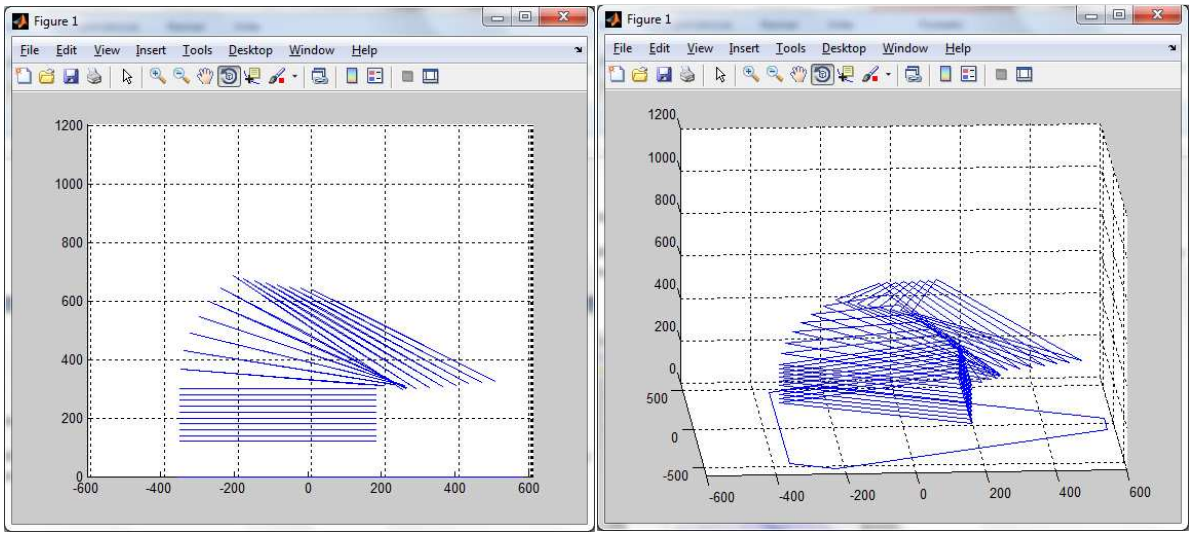




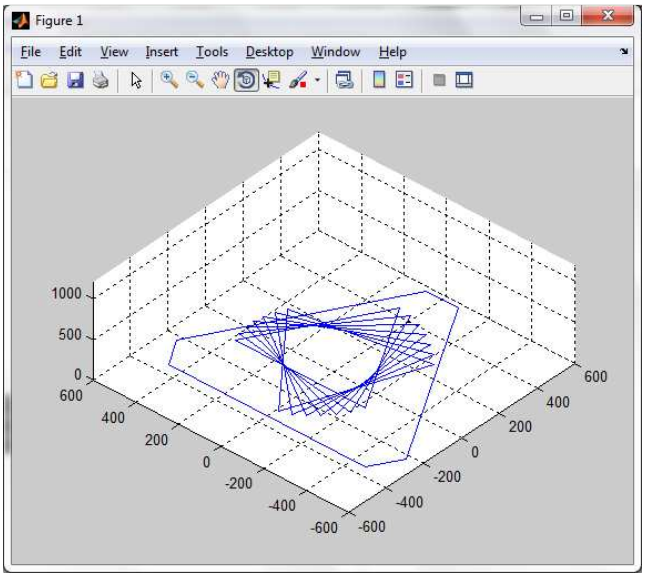


Otros ejemplos de trayectorias:

Traslación en z, luego rotación en beta y gamma, y luego traslación en x y menor rotación en beta y gamma.



Rotación simple sobre el eje Z de la base:



Implementación en DSP56800E

Se encuentra comentado el código de MATLAB correspondiente a cada instrucción del DSP.

```
/** ###################################################################
**     Filename  : tp17.C
**     Project   : tp17
**     Processor : 56F8367
**     Version   : Driver 01.14
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 26/04/2011, 06:28 p.m.
**     Abstract  :
**         Main module.
**         This module contains user's application code.
**     Settings  :
**     Contents  :
**         No public methods
**
** ###################################################################*/
/* MODULE tp17 */

/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "AFR1.h"
#include "MFR1.h"
#include "DFR1.h"
#include "MEM1.h"
#include "XFR1.h"
#include "TFR1.h"
#include "VFR1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include "stdio.h"

#define MXRAD 100
#define PULSE2RAD 32767/MXRAD

//Variables de dimensionamiento
Frac16 a,b,d;
//Variables de estado
Frac16 px, py, pz;
double alpha, beta, gama;

//Variables de actuadores
Word32 L1_32, L2_32, L3_32,
             L4_32, L5_32, L6_32;

Word16 L1, L2, L3,
             L4, L5, L6;
```

```
//Cofactores
Frac16 cosA, cosB, cosG, sinA ,sinB ,sinG;
Frac16 co11, co12, co13, co14,
       co21, co22, co23, co24,
       co31, co32,   co33, co34,
       co41, co42, co43, co44;

//Coordenadas BASE
Word16 Xb1, Xb2, Xb3, Xb4, Xb5, Xb6,
             Yb1, Yb2,Yb3, Yb4, Yb5, Yb6,
             Zb1, Zb2, Zb3, Zb4, Zb5, Zb6;

//Coordenadas TOP respecto de BASE
Word16 Xt1, Xt2, Xt3,
             Yt1, Yt2,Yt3,
             Zt1, Zt2, Zt3;

//Coordenadas TOP respecto de TOP
Word16 xt1, xt2, xt3,
             yt1, yt2,yt3,
             zt1, zt2, zt3;

Word32 aux32;
Word16 aux16;
Frac16 aux;

Frac16 c,i;
Frac16 pulse2rad=PULSE2RAD ,testResult[MXRAD],testResult2[MXRAD];
Word16 c16[MXRAD];
Word32 c32[MXRAD];

void main(void)
{
/* Write your local variable definition here */
/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
PE_low_level_init();
/*** End of Processor Expert internal initialization.                    ***/
/* Write your code here */

//Parametros BASE stewart

b=925  ;  //mm Lado largo de la base
d=150  ;  //mm Lado corto de la base
a=1225/2; //mm Lado TOP

//Traslacion de la plataforma
px=100;
py=100;
pz=100;

//Rotacion de la plataforma
//Completar en fraccion de rad.
//ej: alpha=0.1 equivale a 0.1 * pi => 18 grados
alpha = 0.1;
beta = 0.1;
gama = 0.1;

/*Calculo de coordenadas de BASE*/
```

```
//      Xb1 = (3^(1/2)/6)*(2*b+d);
        aux16 = FRAC16(0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(2*b+d));
        Xb1 = extract_h(aux32);

//      Yb1 = (1/2)*d;
        aux16 = FRAC16(0.5);//(1/2)
        aux32 = L_mult(aux16,d);
        Yb1 = extract_h(aux32);

//      Zb1 = 0;
        Zb1 = 0;

//      Xb2 = -(sqrt(3)/6)*(b-d);
        aux16 = FRAC16(-0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(b-d));
        Xb2 = extract_h(aux32);

//      Yb2 = (1/2)*(b+d);
        aux16 = FRAC16(0.5);//(1/2)
        aux32 = L_mult(aux16,b+d);
        Yb2 = extract_h(aux32);

//      Zb2 = 0;
        Zb2 = 0;

//      Xb3 = -(sqrt(3)/6)*(b+2*d);
        aux16 = FRAC16(-0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(b+2*d));
        Xb3 = extract_h(aux32);

//      Yb3 = (1/2)*b;
        aux16 = FRAC16(0.5);//(1/2)
        aux32 = L_mult(aux16,b);
        Yb3 = extract_h(aux32);

//      Zb3 = 0;
        Zb3 = 0;

//      Xb4 = -(sqrt(3)/6)*(b+2*d);
        aux16 = FRAC16(-0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(b+2*d));
        Xb4 = extract_h(aux32);

//      Yb4 = -(1/2)*b;
        aux16 = FRAC16(-0.5);//(1/2)
        aux32 = L_mult(aux16,b);
        Yb4 = extract_h(aux32);

//      Zb4 = 0;
        Zb4 = 0;

//      Xb5 = -(sqrt(3)/6)*(b-d);
        aux16 = FRAC16(-0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(b-d));
        Xb5 = extract_h(aux32);

//      Yb5 = -(1/2)*(b+d);
```

```
        aux16 = FRAC16(-0.5);//(1/2)
        aux32 = L_mult(aux16,b+d);
        Yb5 = extract_h(aux32);

//      Zb5 = 0;
        Zb5 = 0;

//      Xb6 = (sqrt(3)/6)*(2*b+d);
        aux16 = FRAC16(0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,(2*b+d));
        Xb6 = extract_h(aux32);

//      Yb6 = -(1/2)*d;
        aux16 = FRAC16(-0.5);//(1/2)
        aux32 = L_mult(aux16,d);
        Yb6 = extract_h(aux32);

//      Zb6 = 0;
        Zb6 = 0;

/*      Con subindice en minuscula es respecto a TOP
        Con mAyuscula respecto a la base.
*/

//      xt1 = (sqrt(3)/6)*a;
        aux16 = FRAC16(0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,a);
        xt1 = extract_h(aux32);

//      yt1 = (1/2)*a;
        aux16 = FRAC16(0.5);//(1/2)
        aux32 = L_mult(aux16,a);
        yt1 = extract_h(aux32);

//      zt1 = 0;
        zt1 = 0;

//      xt2 = -(sqrt(3)/3)*a;
        aux16 = FRAC16(-0.5773);//(3^(1/2)/3)
        aux32 = L_mult(aux16,a);
        xt2 = extract_h(aux32);
//      yt2 = 0;
//      zt2 = 0;
        yt2 = 0;
        zt2 = 0;
//      xt3 = (sqrt(3)/6)*a;
        aux16 = FRAC16(0.2886);//(3^(1/2)/6)
        aux32 = L_mult(aux16,a);
        xt3 = extract_h(aux32);
//      yt3 = -(1/2)*a;
        aux16 = FRAC16(-0.5);//(1/2)
        aux32 = L_mult(aux16,a);
        yt3 = extract_h(aux32);
//      zt3 = 0;
        zt3 = 0;
```

```
for(;;) {

/*Precalculo valores de sin y cos para la matriz*/
       cosA = TFR1_tfr16CosPIx(FRAC16(alpha));
       cosB = TFR1_tfr16CosPIx(FRAC16(beta));
       cosG = TFR1_tfr16CosPIx(FRAC16(gama));

       sinA = TFR1_tfr16SinPIx(FRAC16(alpha));
       sinB = TFR1_tfr16SinPIx(FRAC16(beta));
       sinG = TFR1_tfr16SinPIx(FRAC16(gama));

//Cargo los cofactores de la matriz TTB

       co11 =
add(extract_h(L_mult(cosB,cosG)),extract_h(L_mult(extract_h(L_mult(sinA,sinB)),sinG)));
       co12 =
add(extract_h(L_mult(negate(cosB),sinG)),extract_h(L_mult(extract_h(L_mult(sinA,sinB)),co
sG)));
       co13 = extract_h(L_mult(cosA,sinB));
       co14 = px;

       co21 = extract_h(L_mult(cosA,sinG));
       co22 = extract_h(L_mult(cosA,cosG));
       co23 = negate(sinA);
       co24 = py;

       co31 =
add(extract_h(L_mult(negate(sinB),cosG)),extract_h(L_mult(extract_h(L_mult(sinA,cosB)),si
nG)));
       co32 =
add(extract_h(L_mult(sinB,sinG)),extract_h(L_mult(extract_h(L_mult(sinA,cosB)),cosG)));
       co33 = extract_h(L_mult(cosA,cosB));
       co34 = pz;

       co41 = 0;
       co42 = 0;
       co43 = 0;
       co44 = 1;

/*TTB = [cos(beta)*cos(gama)+sin(alpha)*sin(beta)*sin(gama) ...
           -cos(beta)*sin(gama)+sin(alpha)*sin(beta)*cos(gama) ...
           cos(alpha)*sin(beta) px;...

           cos(alpha)*sin(gama)...
           cos(alpha)*cos(gama)...
           -sin(alpha) py;...

           -sin(beta)*cos(gama)+sin(alpha)*cos(beta)*sin(gama) ...
            sin(beta)*sin(gama)+sin(alpha)*cos(beta)*cos(gama) ...
            cos(alpha)*cos(beta) pz;...

            0 0 0 1;...
           ];
*/

/*
       XYZt1 = TTB * [xt1 yt1 zt1 1]'
       XYZt2 = TTB * [xt2 yt2 zt2 1]';
```

```
        XYZt3 = TTB * [xt3 yt3 zt3 1]';

        Xt1=XYZt1(1);
        Yt1=XYZt1(2);
        Zt1=XYZt1(3);

        Xt2=XYZt2(1);
        Yt2=XYZt2(2);
        Zt2=XYZt2(3);

        Xt3=XYZt3(1);
        Yt3=XYZt3(2);
        Zt3=XYZt3(3);
*/

//Posicion de los vertices de la plataforma TOP respecto de la base.

        Xt1 =
add(add(add(extract_h(L_mult(co11,xt1)),extract_h(L_mult(co12,yt1))),extract_h(L_mult(co1
3,zt1))),co14);
        Yt1 =
add(add(add(extract_h(L_mult(co21,xt1)),extract_h(L_mult(co22,yt1))),extract_h(L_mult(co2
3,zt1))),co24);
        Zt1 =
add(add(add(extract_h(L_mult(co31,xt1)),extract_h(L_mult(co32,yt1))),extract_h(L_mult(co3
3,zt1))),co34);

        Xt2 =
add(add(add(extract_h(L_mult(co11,xt2)),extract_h(L_mult(co12,yt2))),extract_h(L_mult(co1
3,zt2))),co14);
        Yt2 =
add(add(add(extract_h(L_mult(co21,xt2)),extract_h(L_mult(co22,yt2))),extract_h(L_mult(co2
3,zt2))),co24);
        Zt2 =
add(add(add(extract_h(L_mult(co31,xt2)),extract_h(L_mult(co32,yt2))),extract_h(L_mult(co3
3,zt2))),co34);

        Xt3 =
add(add(add(extract_h(L_mult(co11,xt3)),extract_h(L_mult(co12,yt3))),extract_h(L_mult(co1
3,zt3))),co14);
        Yt3 =
add(add(add(extract_h(L_mult(co21,xt3)),extract_h(L_mult(co22,yt3))),extract_h(L_mult(co2
3,zt3))),co24);
        Zt3 =
add(add(add(extract_h(L_mult(co31,xt3)),extract_h(L_mult(co32,yt3))),extract_h(L_mult(co3
3,zt3))),co34);

//Longitudes de los actuadores lineales.

//      L1 = sqrt((Xt1 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt1 - d/2)^2 + Zt1^2 );
        aux32 = L_add(L_negate(L_mult(d, FRAC16(0.2886))),L_negate(L_mult(b,
FRAC16(0.5773))));
        aux16 = add(extract_h(aux32),Xt1);
        L1_32 = L_mult(aux16,aux16);
        aux16 = add(Yt1, negate(extract_h(L_mult(FRAC16(0.5),d))));
        L1_32 = L_add(L_mult(aux16,aux16),L1_32);
        L1_32 = L_add(L_mult(Zt1,Zt1),L1_32);
        L1 = mfr32Sqrt(L1_32);      //L1 actuador lineal
```

```
//      L2 = sqrt((Xt1 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt1 - d/2 - b/2)^2 + Zt1^2 );
        aux32 = L_add(L_negate(L_mult(d, FRAC16(0.2886))),L_mult(b, FRAC16(0.2886)));
        aux16 = add(extract_h(aux32),Xt1);
        L2_32 = L_mult(aux16, aux16);
        aux16 = add(Yt1, negate(extract_h(L_mult(FRAC16(0.5),d))));
        aux16 = add(aux16, negate(extract_h(L_mult(FRAC16(0.5),b))));
        L2_32 = L_add(L_mult(aux16,aux16),L2_32);
        L2_32 = L_add(L_mult(Zt1,Zt1),L2_32);
        L2 = mfr32Sqrt(L2_32);


//      L3 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 - b/2)^2 + Zt2^2 );
        aux32 = L_add(L_mult(d, FRAC16(0.5773)),L_mult(b, FRAC16(0.2886)));
        aux16 = add(extract_h(aux32),Xt2);
        L3_32 = L_mult(aux16, aux16);
        aux16 = add(Yt2, negate(extract_h(L_mult(FRAC16(0.5),b))));
        L3_32 = L_add(L_mult(aux16,aux16),L3_32);
        L3_32 = L_add(L_mult(Zt2,Zt2),L3_32);
        L3 = mfr32Sqrt(L3_32);


//      L4 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 + b/2)^2 + Zt2^2 );
        aux32 = L_add(L_mult(d, FRAC16(0.5773)),L_mult(b, FRAC16(0.2886)));
        aux16 = add(extract_h(aux32),Xt2);
        L4_32 = L_mult(aux16, aux16);
        aux16 = add(Yt2, extract_h(L_mult(FRAC16(0.5),b)));
        L4_32 = L_add(L_mult(aux16,aux16),L4_32);
        L4_32 = L_add(L_mult(Zt2,Zt2),L4_32);
        L4 = mfr32Sqrt(L4_32);

//      L5 = sqrt((Xt3 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt3 + b/2 + d/2)^2 + Zt3^2 );
        aux32 = L_add(L_negate(L_mult(d, FRAC16(0.2886))),L_mult(b, FRAC16(0.2886)));
        aux16 = add(extract_h(aux32),Xt3);
        L5_32 = L_mult(aux16,aux16);
        aux16 = add(Yt3, extract_h(L_mult(FRAC16(0.5),d)));
        aux16 = add(aux16, extract_h(L_mult(FRAC16(0.5),b)));
        L5_32 = L_add(L_mult(aux16,aux16),L5_32);
        L5_32 = L_add(L_mult(Zt3,Zt3),L5_32);
        L5 = mfr32Sqrt(L5_32);


//      L6 = sqrt((Xt3 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt3 + d/2)^2 + Zt3^2 );
        aux32 = L_add(L_negate(L_mult(d, FRAC16(0.2886))),L_negate(L_mult(b,
FRAC16(0.5773))));
        aux16 = add(extract_h(aux32),Xt3);
        L6_32 = L_mult(aux16,aux16);
        aux16 = add(Yt3, extract_h(L_mult(FRAC16(0.5),d)));
        L6_32 = L_add(L_mult(aux16,aux16),L6_32);
        L6_32 = L_add(L_mult(Zt3,Zt3),L6_32);
        L6 = mfr32Sqrt(L6_32);
  }
}
```

Tabla comparativa de simulaciones:

Traslación y rotación deseada	Px	0 mm	0 mm	0 mm	10 mm
	Py	0 mm	0 mm	0 mm	10 mm
	Pz	100 mm	120 mm	130 mm	140 mm
	Alpha	0°	18°	27°	36°
	Beta	0°	0°	0°	0°
	gamma	0°	0°	0°	18°
Resultados Matlab	L1	473 mm	485 mm	493 mm	531 mm
	L2	473 mm	487 mm	496 mm	479 mm
	L3	473 mm	477 mm	480 mm	505 mm
	L4	473 mm	477 mm	480 mm	457 mm
	L5	473 mm	471 mm	471 mm	521 mm
	L6	473 mm	470 mm	460 mm	419 mm
Resultados DSP	L1	475 mm	504 mm	523 mm	638 mm
	L2	471 mm	515 mm	548 mm	521 mm
	L3	473 mm	477 mm	480 mm	547 mm
	L4	473 mm	477 mm	480 mm	391 mm
	L5	471 mm	469 mm	478 mm	608 mm
	L6	475 mm	458 mm	449 mm	325 mm

Conclusiones del análisis cinemático inverso:

Mediante el estudio de publicaciones con referencia a la plataforma Stewart, se logró la implementación de la cinemática inversa con la obtención de trayectorias. Luego al comparar los resultados de la simulación del DSP con nuestro script de Matlab, pudimos observar que si bien los resultados se comportan coherentemente, se observan errores notables cuando entran en juego las rotaciones. A efectos de la realización práctica de este Tp sería necesario utilizar un dsp con mayor precisión en el cálculo de los senos y cosenos, debido a que la propagación de los errores de los mismos es muy fuerte. Esto ocurre por la cantidad de operaciones realizadas en forma encadenada.

Segunda parte: Análisis dinámico.

Objetivos:

- Obtención del modelo dinámico inverso de una plataforma Stewart genérica.
- Implementación en Matlab.
- Resultados de la simulación.

Introducción:

La dinámica es la parte de la física que describe la evolución en el tiempo de un sistema físico en relación a las causas que provocan los cambios de estado físico y/o estado de movimiento.
El objetivo de la dinámica es describir los factores capaces de producir alteraciones de un sistema físico, cuantificarlos y plantear ecuaciones de movimiento o ecuaciones de evolución para dicho sistema.

Por lo tanto, el modelo dinámico de un robot tiene por objetivo conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo.
Esta relación se obtiene mediante el denominado modelo dinámico, que relaciona matemáticamente:
La localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración.
Las fuerzas y pares aplicados en las articulaciones (o en el extremo del robot).
Los parámetros dimensionales del robot, como longitud, masas e inercias de sus elementos.

Dinámica de la plataforma Stewart:

Las ecuaciones clásicas de movimiento pueden obtenerse de la ecuación de Lagrange.

$$\frac{d}{dt}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial k(q,\dot{q})}{dq} + \frac{\partial P(q)}{\partial q} = \tau$$

Donde "q" representa coordenadas, y "τ" representa fuerzas, $k(q,\dot{q})$ es la energía cinética y $P(q)$ es la energía potencial. Esta misma expresión puede reformularse de la siguiente manera:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = J^T F$$

Donde $M(q)$ y $C(q)$ son la Matriz de Inercia y la Matriz de Coriolis respectivamente. F representa las fuerzas de cada actuador o "pierna".

Para obtener la energía cinética y potencial, el estudio de la plataforma Stewart puede dividirse en dos partes, la plataforma superior y las seis piernas. Sin embargo en este trabajo, se despreciará la masa de los actuadores frente a la parte superior, como así también las fuerzas debidas al efecto Coriolis, por lo que la matriz C(q) no tendrá efecto en nuestra simulación.

Del estudio analítico que puede verse en la referencia [2] se obtiene la matriz M(q).

$$M(q)=\begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_X C_\gamma^2 + I_Y S_\gamma^2 & (I_X - I_Y) C_\alpha C_\gamma S_\gamma & 0 \\ 0 & 0 & 0 & (I_X - I_Y) C_\alpha C_\gamma S_\gamma & C_\alpha^2 (I_X S_\gamma^2 + I_Y C_\gamma^2) + I_Z S_\alpha^2 & -I_Z S_\alpha \\ 0 & 0 & 0 & 0 & -I_Z S_\alpha & I_Z \end{bmatrix}$$

Donde “m” es la masa de la plataforma superior y los términos Ix, Iy e Iz son componentes del tensor de inercias I = diag{Ix Iy Iz}.

Cálculo del Tensor de Inercia

El tensor de inercia es un tensor simétrico de segundo orden que caracteriza la inercia rotacional de un sólido rígido.
Expresado en una base ortonormal viene dado por una matriz simétrica, dicho tensor se forma a partir de los momentos de inercia según tres ejes perpendiculares y tres productos de inercia.
El tensor de inercia sólido rígido se define como un tensor simétrico de segundo orden tal que la forma cuadrática construida a partir del tensor y la velocidad angular Ω da la energía cinética de rotación, es decir:

$$E_{rot} = \frac{1}{2} \begin{pmatrix} \Omega_x & \Omega_y & \Omega_z \end{pmatrix} \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} \begin{pmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{pmatrix} = \frac{1}{2} \sum_j \sum_k I_{jk} \Omega_j \Omega_k$$

Donde las componentes de este tensor de inercia en una base ortonormal XYZ pueden calcularse a partir de los tres momentos de inercia según esos tres ejes perpendiculares:

$$\begin{cases} I_{xx} = \int_M d_x^2 dm = \int_V \rho(y^2 + z^2) dx dy dz \\ I_{yy} = \int_M d_y^2 dm = \int_V \rho(z^2 + x^2) dx dy dz \\ I_{zz} = \int_M d_z^2 dm = \int_V \rho(x^2 + y^2) dx dy dz \end{cases}$$

Donde ρ es la densidad del material.

Cálculo de la Matriz Potencial

$$P = mgZ = \begin{bmatrix} 0 \\ 0 \\ mg \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} X \\ Y \\ Z \\ \alpha \\ \beta \\ \gamma \end{bmatrix} = [0 \quad 0 \quad mg \quad 0 \quad 0 \quad 0]q$$

Entonces

$$G(q) = \frac{\partial P(q)}{\partial q} = [0 \quad 0 \quad mg \quad 0 \quad 0 \quad 0]$$

Cálculo de la Matriz Jacobiana

Las ecuaciones de L1…L6 vistas en el análisis cinemático, dan explícitamente las longitudes de los actuadores en función de las coordenadas de la plataforma superior q_T =(XT1, YT1, ZT1), y luego esta se encuentran en función del centro de la misma q = (X,Y,Z,alpha,beta,gamma). Por lo tanto sus velocidades serán:

$$\dot{L} = J_1(q).\dot{q_T}$$

$$\dot{q_T} = J_2(q).\dot{q}$$

$$\dot{L} = J(q).\dot{q}$$

Resolución de dinámica inversa

El método para resolver la dinámica inversa, es que "q" es identificado como una entrada y a la vez es función del tiempo. Por ello puede obtenerse su derivada primera y segunda, logrando así la velocidad y aceleración. Usando esto como dato, en la ecuación de Lagrange anteriormente planteada, se resuelve el sistema de ecuaciones para obtener las "F", que determinan una trayectoria.

Resolución numérica e Implementación en Matlab del análisis dinámico en conjunto con el análisis cinemático

Consideramos las dimensiones físicas de la plataforma como en la referencia [2].

a = 0.2 (m)
b = 0.4 (m)
d = 0.1 (m)
m = 1.36 (kg)
g = 9.8066 (m/s 2)

$$q = \begin{bmatrix} -0.1 \\ 0.1\mathrm{Sin}(\omega t) \\ 0.7 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \omega = 2\,(\mathrm{rad/s})$$

Partimos del script GraficoStewart.m utilizado para el análisis cinemático inverso, y agregamos los cálculos dinámicos.

Script GraficoStewart.m

```
syms alpha beta gamma ; %alpha: rot en x
                        %beta: rot en y
                        %gamma: rot en z

syms px py pz;  %Distancia al origen del TOP respecto de BASE

%Parametros BASE stewart
b=0.400;    %m Lado largo de la base
d=0.100;   %m Lado corto de la base

%Parametro de TOP
%Parametros para el tensor de inercia
dens = 7580; %kg/m3
espesor_chapa = 0.01036; %m
masa = 1.36; %masa kg
a=0.200;  %m Lado TOP
g=9.81; %gravedad m/seg2

%Calculo cinematica inversa. Determina las ecuaciones de L1...L6
Stewart;

%Calcula el tensor de inercia.
%Determina la los elementos de tensor de inercia. Ixx, Iyy, Izz.
i_tensor;

%Calcula expresion de matriz jacobiana para obtener velocidades de L y la
```

```
%deja cargada en Jsymbolic.
Jacobiana;

%Inercias para energia cinetica. Queda cargada en Msymbolic
MatrizInercia;

%Energia potencial. Queda cargada en Gsymbolic.
MatrizPotencial;

%Generar trayectorias. Y obtiene Fuerzas para los actuadores.
%Grafica.
Generador;
```

Script Stewart.m

Es el mismo script utilizado en el análisis cinemático.

```
Xb1 = (sqrt(3)/6)*(2*b+d);
Yb1 = (1/2)*d;
Zb1 = 0;

Xb2 = -(sqrt(3)/6)*(b-d);
Yb2 = (1/2)*(b+d);
Zb2 = 0;

Xb3 = -(sqrt(3)/6)*(b+2*d);
Yb3 = (1/2)*b;
Zb3 = 0;

Xb4 = -(sqrt(3)/6)*(b+2*d);
Yb4 = -(1/2)*b;
Zb4 = 0;

Xb5 = -(sqrt(3)/6)*(b-d);
Yb5 = -(1/2)*(b+d);
Zb5 = 0;

Xb6 = (sqrt(3)/6)*(2*b+d);
Yb6 = -(1/2)*d;
Zb6 = 0;

%Con subindice en minuscula es respecto a TOP
%Con mAyuscula respecto a la base
xt1 = (sqrt(3)/6)*a;
yt1 = (1/2)*a;
zt1 = 0;

xt2 = -(sqrt(3)/3)*a;
yt2 = 0;
zt2 = 0;

xt3 = (sqrt(3)/6)*a;
yt3 = -(1/2)*a;
```

```
zt3 = 0;


TTB = [cos(beta)*cos(gamma)+sin(alpha)*sin(beta)*sin(gamma) ...
       -cos(beta)*sin(gamma)+sin(alpha)*sin(beta)*cos(gamma) ...
       cos(alpha)*sin(beta) px;...

       cos(alpha)*sin(gamma)...
       cos(alpha)*cos(gamma)...
       -sin(alpha) py;...

       -sin(beta)*cos(gamma)+sin(alpha)*cos(beta)*sin(gamma) ...
        sin(beta)*sin(gamma)+sin(alpha)*cos(beta)*cos(gamma) ...
        cos(alpha)*cos(beta) pz;...

        0 0 0 1;...
       ];

XYZt1 = TTB * [xt1 yt1 zt1 1]';
XYZt2 = TTB * [xt2 yt2 zt2 1]';
XYZt3 = TTB * [xt3 yt3 zt3 1]';

Xt1=XYZt1(1);
Yt1=XYZt1(2);
Zt1=XYZt1(3);

Xt2=XYZt2(1);
Yt2=XYZt2(2);
Zt2=XYZt2(3);

Xt3=XYZt3(1);
Yt3=XYZt3(2);
Zt3=XYZt3(3);

L1 = sqrt((Xt1 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt1 - d/2)^2 + Zt1^2 );
L2 = sqrt((Xt1 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt1 - d/2 - b/2)^2 +
Zt1^2 );
L3 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 - b/2)^2 + Zt2^2 );
L4 = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt2 + b/2)^2 + Zt2^2 );
L5 = sqrt((Xt3 - d/(2*sqrt(3)) + b/(2*sqrt(3)))^2 + (Yt3 + b/2 + d/2)^2 +
Zt3^2 );
L6 = sqrt((Xt3 - d/(2*sqrt(3)) - b/sqrt(3))^2 + (Yt3 + d/2)^2 + Zt3^2 );
```

Script i_tensor.m

```
syms x y z

Ixx = dens * int(int(int((y^2+z^2),z,0,espesor_chapa),y, (x-xt2)*((yt3-
yt2)/(xt3-xt2)), (x-xt2)*((yt1-yt2)/(xt1-xt2))),x,xt2,xt1);
Iyy = dens * int(int(int((x^2+z^2),z,0,espesor_chapa),y, (x-xt2)*((yt3-
yt2)/(xt3-xt2)), (x-xt2)*((yt1-yt2)/(xt1-xt2))),x,xt2,xt1);
Izz = dens * int(int(int((y^2+x^2),z,0,espesor_chapa),y, (x-xt2)*((yt3-
yt2)/(xt3-xt2)), (x-xt2)*((yt1-yt2)/(xt1-xt2))),x,xt2,xt1);
```

Script Jacobiana.m

```
%Calcula la matriz jacobiana total para la obtencion de las velocidades de
%L1...L6
f1  = [L1; L2; L3; L4; L5; L6];
v1 =  [px, py, pz, alpha, beta, gamma];
Jsymbolic = jacobian(f1,v1);
```

Script MatrizInercia.m

```
%Matriz inercia M(q)

Msymbolic = [masa,      0,      0,      0,              0,      0;...
                0,      masa,   0,      0,              0,      0;...
                0,      0,      masa,   0,              0,      0;...
                0,      0,      0,      Ixx*cos(gamma)^2+Iyy*sin(gamma)^2,
(Ixx-Iyy)*cos(alpha)*cos(gamma)*sin(gamma), 0;...
                0,      0,      0,      (Ixx-
Iyy)*cos(alpha)*cos(gamma)*sin(gamma),
(cos(alpha)^2)*(Ixx*(sin(gamma)^2)+Iyy*(cos(gamma)^2))+Izz*(sin(alpha)^2), -
Izz*sin(alpha);...
                0,      0,      0,      0,              -Izz*sin(alpha),
Izz];
```

Script MatrizPotencial.m

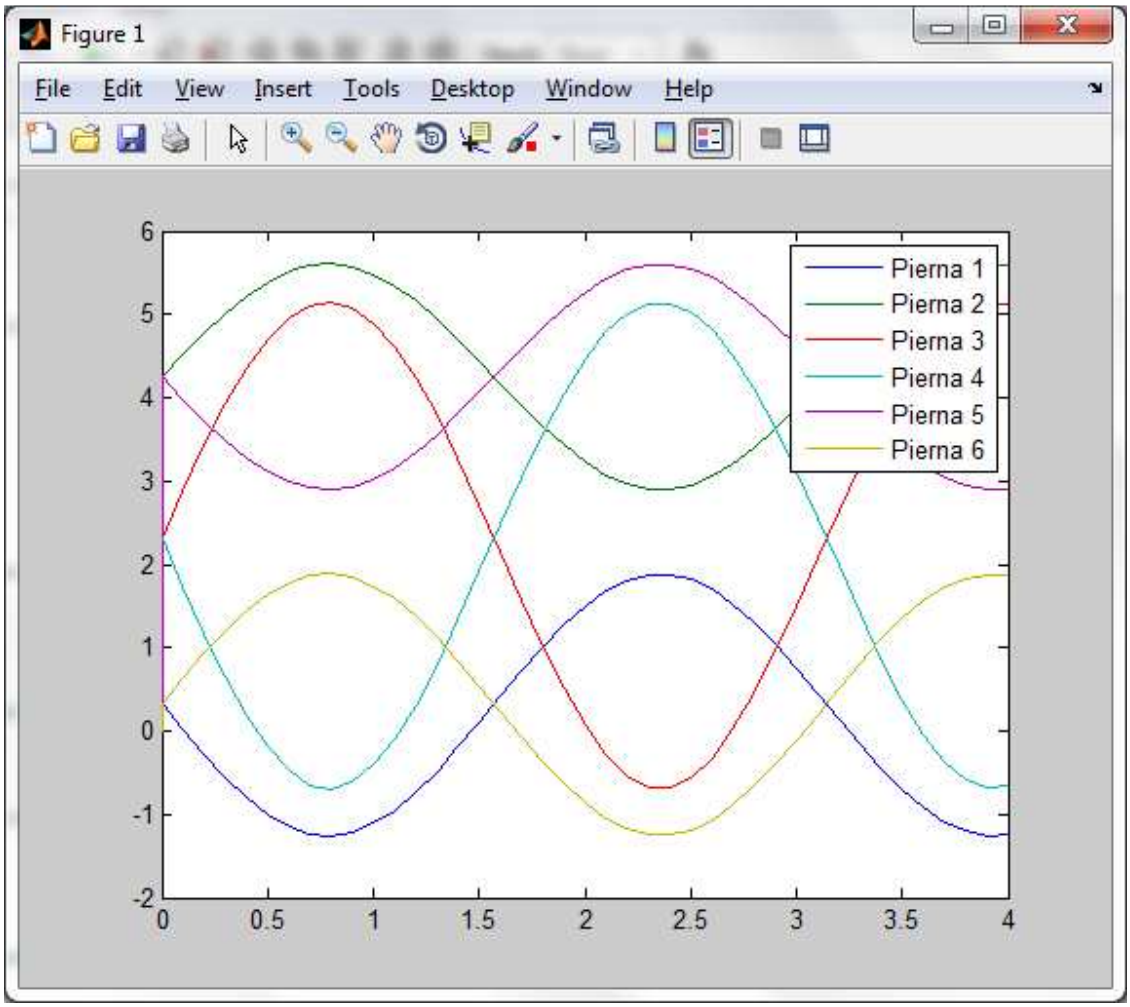
```
%Matriz de Energia potencial
Gsymbolic = [0; 0; masa*g; 0; 0; 0];
```

Script Generador.m

```
%Genreamos vector de estado. Con trayectoria.

w=2;%2 r/seg

figure(1);
xlim([0 2]);
ylim([-2 6]);

f_ant1=0;
t_ant1=0;
f_ant2=0;
t_ant2=0;
f_ant3=0;
t_ant3=0;
f_ant4=0;
t_ant4=0;
f_ant5=0;
t_ant5=0;
f_ant6=0;
t_ant6=0;
```

```
for t=0:0.1:4

px = -0.1;
py = 0.1*sin(w*t);
pz = 0.7;
alpha = 0;
beta = 0;
gamma = 0;

q   = [px; py; pz; alpha; beta; gamma];

qdd = [0; -0.4*sin(w*t); 0; 0; 0; 0];

B = eval(Msymbolic*qdd + Gsymbolic);
A = eval(Jsymbolic');

F = linsolve(A,B);

%Grafico las 6 curvas de fuerzas para cada pierna.
Faux1=[ f_ant1 F(1)];
taux1=[ t_ant1 t];
Faux2=[ f_ant2 F(2)];
taux2=[ t_ant2 t];
Faux3=[ f_ant3 F(3)];
taux3=[ t_ant3 t];
Faux4=[ f_ant4 F(4)];
taux4=[ t_ant4 t];
Faux5=[ f_ant5 F(5)];
taux5=[ t_ant5 t];
Faux6=[ f_ant6 F(6)];
taux6=[ t_ant6 t];

plot(taux1, Faux1, taux2, Faux2,taux3, Faux3,taux4, Faux4,taux5, Faux5,taux6,
Faux6);
hleg = legend('Pierna 1','Pierna 2','Pierna 3', 'Pierna 4', 'Pierna 5',
'Pierna 6');

t_ant1=t;
f_ant1=F(1);
t_ant2=t;
f_ant2=F(2);
t_ant3=t;
f_ant3=F(3);
t_ant4=t;
f_ant4=F(4);
t_ant5=t;
f_ant5=F(5);
t_ant6=t;
f_ant6=F(6);

hold on;
drawnow;

end
hold off;
```

Resultados Simulación



Conclusiones del análisis dinámico inverso

El análisis de la dinámica inversa en la plataforma Stewart, es un mapeo desde la posición y orientación de la plataforma superior en función del tiempo (“q”) hacia la fuerza aplicada por cada uno de los actuadores. Pudimos simular este sistema en matlab, y obtuvimos resultados idénticos a los publicados en el paper de la referencia [2], habiendo solo nosotros despreciado el efecto de la matriz de Coriolis, confirmando que hicimos bien en hacerlo.

Por otra parte, vimos que es sencillo expresar matemáticamente el método Lagrangiano utilizado, pero, no obstante, se vuelve un tanto lento porque para cada iteración se debe recalcular la matriz Jacobiana.

Tercera parte: Compilador.

Introducción

Un compilador es un programa que se encarga de buscar y ejecutar instrucciones que correspondan con un lenguaje determinado.

Mediante nuestro compilador, podemos ejecutar instrucciones de movimiento de la plataforma, de una manera más simple para el usuario, con un código de más alto nivel.

Para esto, utilizamos el software Antlr, con su interfaz gráfica AntlrWorks.

Antlr es una herramienta que opera sobre lenguajes, proporcionando un marco para construir reconocedores (parsers), intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos.

A partir de la descripción formal de la gramática de un lenguaje, Antlr genera un programa que determina si una palabra o sentencia pertenece a dicho lenguaje, utilizando algoritmos de parsing. Si a dicha gramática se le añaden acciones escritas en un lenguaje de programación, el reconocedor se transforma en un traductor o intérprete.

El objetivo fue generar código de matlab a partir de un código de más alto nivel. Para eso generamos código de Java que a su vez imprime en pantalla o en un archivo el código de matlab correspondiente a las instrucciones recibidas. Se implementaron manejos necesarios tales como sumas, restas, manejo de variables, asignaciones, etc., además de la función mover, que toma los parámetros de posición deseada y los traduce a llamadas al cálculo cinemático y dinámico en matlab.

Todo el código necesario para nuestro compilador está definido en el archivo de gramática Stewart.g

Bloques del programa

A continuación, citamos los bloques principales del programa.

Grammar Stewart.g
Con esto definimos el encabezado del fichero.

Tokens {…}
Acá se colocan los elementos que queremos que sean reconocidos por nuestro compilador.

@header {…}
Permite insertar el código de inicialización que querramos colocar al inicio del archivo Java.

@members {…}
Acá va el código de la aplicación resultante en Java.

Reglas del parser
Sintaxis del compilador. Se colocan etiquetas en minúscula para cada regla.

Reglas del lexer
En este bloque se coloca todo lo relacionado al léxico. Se colocan etiquetas en mayúscula.

Archivo de gramática *Stewart.g*:

```
grammar Stewart;

tokens
{
	//Aca definimos los simbolos que reconoce el compilador
	ALIAS		= 'ALIAS';
	AND		= 'AND';
	ARRAY		= 'ARRAY';
	ASSOCIATIVE	= 'ASSOCIATIVE';
	BEGIN		= 'BEGIN';
	BINDINGS	= 'BINDINGS';
	BY		= 'BY';
	CASE		= 'CASE';
	CONST		= 'CONST';
	DEFINITION	= 'DEFINITION';
	DIV		= 'DIV';
	DO		= 'DO';
	ELSE		= 'ELSE';
	ELSIF		= 'ELSIF';
	END		= 'END';
	EXIT		= 'EXIT';
	FOR		= 'FOR';
	FROM		= 'FROM';
	IF		= 'IF';
	IMPLEMENTATION = 'IMPLEMENTATION';
	IMPORT		= 'IMPORT';
	IN		= 'IN';
	LOOP		= 'LOOP';
	MINUS		= '-' ;
	MOD		= 'MOD';
	MODULE		= 'MODULE';
	NOT		= 'NOT';
	OF		= 'OF';
	OPAQUE		= 'OPAQUE';
	OR		= 'OR';
	PLUS		= '+' ;
	POINTER		= 'POINTER';
	PROCEDURE	= 'PROCEDURE';
	PRODUCT		= '*';
	RECORD		= 'RECORD';
	REPEAT		= 'REPEAT';
	RETURN		= 'RETURN';
	SET		= 'SET';
	THEN		= 'THEN';
	TO		= 'TO';
	TYPE		= 'TYPE';
```

```
	UNTIL		= 'UNTIL';
	VAR		= 'VAR';
	VARIADIC	= 'VARIADIC';
	WHILE		= 'WHILE';
	MOVE		= 'mover';
}

@header
{
	import java.util.HashMap;		//Importamos la clase de la tabla Java para almacenar variables
	//package com.ociweb.math;
}

@members
{
	 Integer i=0;
	HashMap variables = new HashMap();	//Tabla de java para almacenar variables
	HashMap valores = new HashMap();
	public static void main(String[] args) throws Exception
	{
		StewartLexer lex = new StewartLexer(new ANTLRFileStream(args[0]));
		CommonTokenStream tokens = new CommonTokenStream(lex);
		StewartParser parser = new StewartParser(tokens);
		try
		{
			parser.expr();
		}
		catch (RecognitionException e)
		{
			e.printStackTrace();
		}
	}
}

/*------------------------------------------------------------------
* PARSER RULES
*------------------------------------------------------------------*/
expr : ((asig2)|(asig))+;  //ver diagrama que asi puedo reconocer varias expresiones
asig : ID '=' op {//System.out.println($ID.text +"="+ $op.value);
			variables.put($ID.text, new Integer($op.value));};  //para debug
op returns [int value]
:
 e=factor {$value = $e.value;}
( PLUS e=factor {$value += $e.value;}
| MINUS e=factor {$value -= $e.value;}
| PRODUCT e=factor {$value *= $e.value;}
)*
;
```

```
factor returns [int value]
:
  NUMBER {$value = Integer.parseInt($NUMBER.text);}   //Con esto acepta un entero
| ID
{
        Integer v = (Integer)variables.get($ID.text);                   //Con esto acepta letras como enteros
        valores.put(i++, new Integer(v));
        //System.out.println("valor1="+(Integer)valores.get(i));
        if ( v!=null ) $value = v.intValue();
        else System.err.println("Variable no definida: "+$ID.text);
};

asig2 : MOVE '(' factor COMA factor COMA factor COMA factor COMA factor COMA factor ')'
{
        System.out.println("\%\% EJEMPO PARA TRADUCIR A CODIGO DE MATLAB");
        //Obtengo las longitudes de las piernas en ek vector L
        System.out.print("L = CinematicaInversa(");
        for(int z=0 ; z<5 ; z++)
        {
                System.out.print((Integer)valores.get(z)+",");
        }
        System.out.println((Integer)valores.get(5)+");");

        //Funcion para obtener los torques en vector T
        System.out.println("T = DinamicaInversa(L)");
};


/*------------------------------------------------------------------
* LEXER RULES
*------------------------------------------------------------------*/
ID : ('a'..'z'|'A'..'Z')+ ;
WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+ { $channel = HIDDEN; } ;
LEFT_PAREN: '(';
//LIST: 'list';
//PRINT: 'print';
RIGHT_PAREN: ')';
COMA   :          ',';
//VARIABLES: 'variables'; // for list command

//SIGN: '+' | '-';
NUMBER: FLOAT|INTEGER;
fragment FLOAT:INTEGER '.' '0'..'9'+;
fragment INTEGER: '0' | '1'..'9' '0'..'9'*;
//fragment INTEGER: '0' | SIGN? '1'..'9' '0'..'9'*;
NAME: LETTER (LETTER | DIGIT | '_')*;
STRING_LITERAL: '"' NONCONTROL_CHAR* '"';
```

```
fragment NONCONTROL_CHAR: LETTER | DIGIT | SYMBOL | SPACE;
fragment LETTER: LOWER | UPPER;
fragment LOWER: 'a'..'z';
fragment UPPER: 'A'..'Z';
fragment DIGIT: '0'..'9';
fragment SPACE: ' ' | '\t';

// Note that SYMBOL does not include the double-quote character.
fragment SYMBOL: '!' | '#'..'/' | ':'..'@' | '['..'`' | '{'..'~';
// Windows uses \r\n. UNIX and Mac OS X use \n.
// To use newlines as a terminator,
// they can't be written to the hidden channel!
//NEWLINE: ('\r'? '\n')+;
```

Ejemplo

Para las pruebas, se permite la inserción de código desde un archivo de texto, o desde la misma interfaz del programa. A continuación vemos un ejemplo corrido con el debug del AntlrWorks:

Archivo de entrada:

```
x=100
y=120
z=300
alpha=25
beta=10
gama=0
mover(x,y,z,alpha,beta,gama)
```

Archivo de salida:

```
%%CÓDIGO DE MATLAB
L = CinematicaInversa(100,120,300,25,10,0);
T = DinamicaInversa(L)
```

Referencias

[1] Kinematic Analysis of a Stewart Platform Manipulator. Kai Liu, *Member, IEEE,* John M. Fitzgerald, and Frank L. Lewis, *Senior Member, IEEE*

[2] Dynamics Analysis and Simulation of Parallel Robot Stewart Platform. Hamidreza Hajimirzaalian. Hasan Moosavi. Mehdi Massah.