ROBOTICA

Cinemática directa sobre DSP

Año 2012

Integrantes:

Alvarez Nahuel (109712-0)

Kowalczuk Ivan (115576-3)

Introducción general

Realizaremos en el presente documento el desarrollo de la cinemática del robot basado en una configuración propietaria que hemos diseñado.

Se trata de un manipulador con 4 grados de libertad de los cuales se puede mover en los tres ejes cartesianos (X; Y; Z) y orientarse en un solo ángulo de giro (Alfa).

Se muestra a continuación el esquema del mencionado robot.

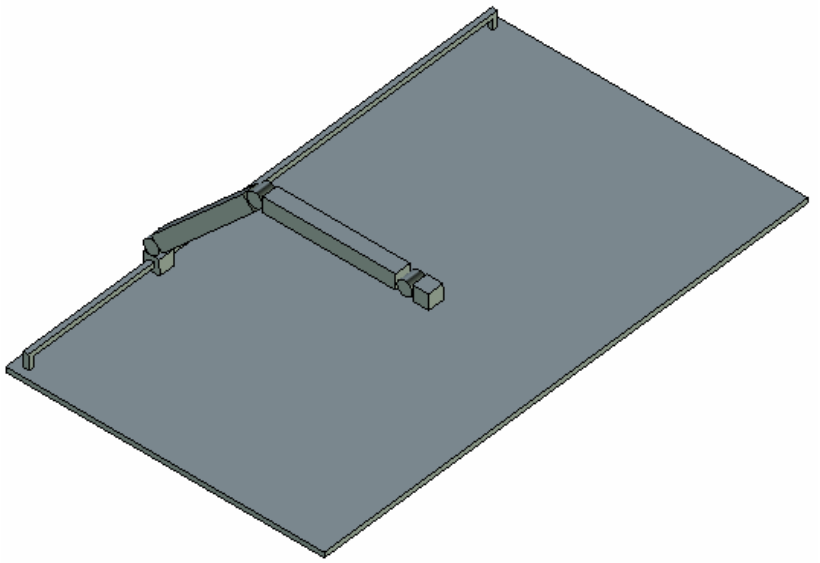


Figura 1. Modelo 3D del manipulador.

Teniendo en cuenta las diferentes herramientas matemáticas que existen para la modelización de las traslaciones del robot, hemos decidido optar por la matriz de transformación homogénea.

Este método permite obtener la localización del extremo del robot así como también su orientación. La misma se encuentra representada por una matriz de cuatro filas y cuatro columnas que relaciona los vectores de posición del sistema de coordenadas del extremo del robot con los del sistema de coordenadas de la base del robot.

La mencionada matriz homogénea puede componerse a partir de cuatro matrices que se muestran a continuación:

$$T = \begin{bmatrix} R\ (3x3) & P\ (3x1) \\ F(1x3) & W(1x1) \end{bmatrix} = \begin{bmatrix} Rotacion & Traslacion \\ Perspectiva & Escalado \end{bmatrix}$$

Fundamentalmente nos ocuparemos de las matrices de rotación y de traslación ya que como el desarrollo se encuentra orientado hacia un robot compuesto por eslabones rígidos que trabaja en 3 dimensiones reales no se observaran efectos de perspectiva y/o escalado.

Las matrices de Rotación en 3 dimensiones se expresan de la siguiente manera,

$$R(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & Cos(\alpha) & -Sen(\alpha) \\ 0 & Sen(\alpha) & Cos(\alpha) \end{bmatrix}$$

$$R(y,\beta) = \begin{bmatrix} Cos(\beta) & 0 & Sen(\beta) \\ 0 & 1 & 0 \\ -Sen(\beta) & 0 & Cos(\beta) \end{bmatrix}$$

$$R(z,\gamma) = \begin{bmatrix} Cos(\gamma) & -Sen(\gamma) & 0 \\ Sen(\gamma) & Cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Mientras que la matriz de traslación (designada como P por posición) de la siguiente manera,

$$P = \begin{bmatrix} Px \\ Py \\ Pz \end{bmatrix}$$

Debido a que buscamos llevar las coordenadas de los diferentes eslabones de nuestro robot hacia el sistema de coordenadas asociado a la base del mismo, utilizaremos el concepto de Pre-multiplicación.

Este método indica que si se quiere realizar sobre un vector determinado una rotación, luego una traslación y por ultimo otra rotación, entonces la multiplicación de matrices para lograr esto se deberán realizar en el mismo orden, la siguiente ecuación describe con claridad esta situación:

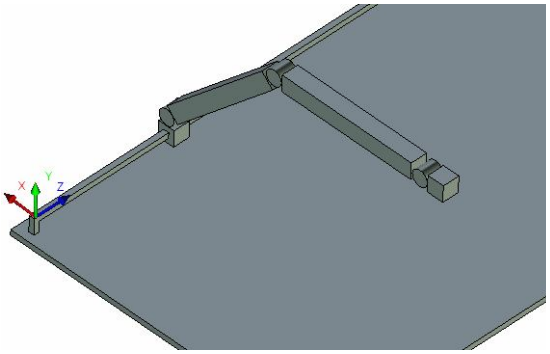
$$Movimiento = T(z, 90º) * T(Pxyz) * T(x, -90º)$$

$$Movimiento = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

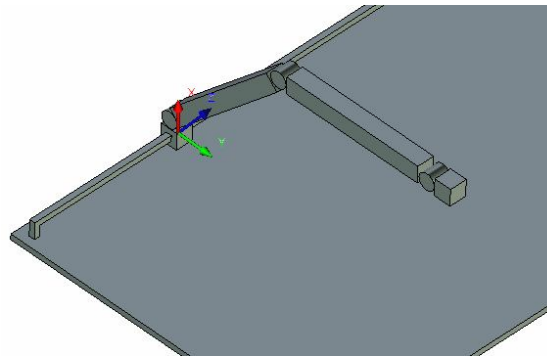
$$Movimiento = \begin{bmatrix} 0 & 0 & -1 & -x \\ 1 & 0 & 0 & y \\ 0 & -1 & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Las N matrices de transformación homogénea que describen las transformaciones entre los sistemas coordenados de cada eslabón pueden ser definidas a partir del algoritmo de Denavit-Hartenberg.

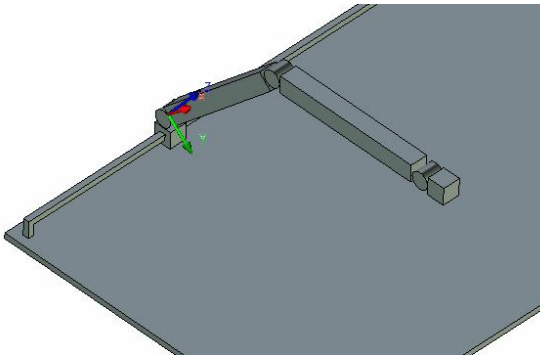
a



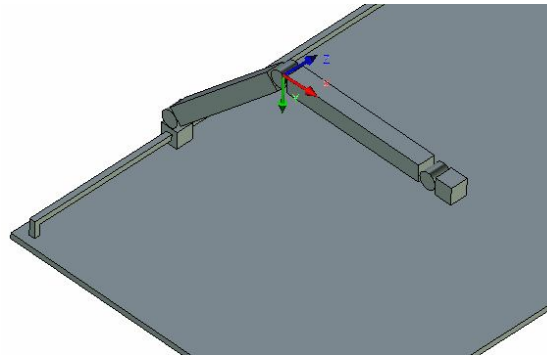
b



c



d



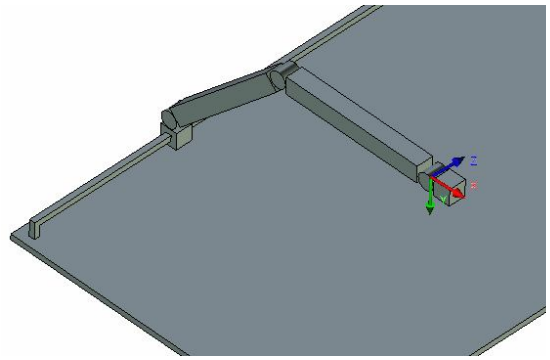
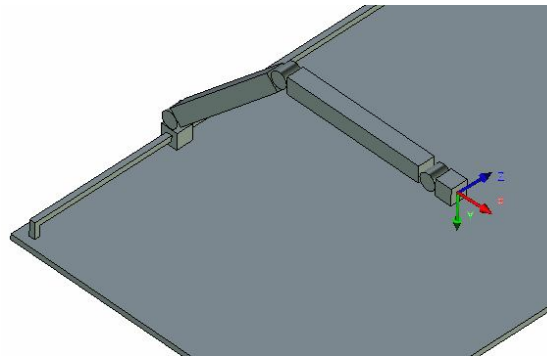
e  f 

Figura 2. Sistemas coordenados adoptados según el algoritmo de Denavit-Hartenberg. a) Sistema coordenado de la base. b) Sistema coordenado del eslabón 1. c) Sistema coordenado del eslabón 2. d) Sistema coordenado del eslabón 3. e) Sistema coordenado del eslabón 4. f) Sistema coordenado del extremo del manipulador.

Aplicando el dicho algoritmo a la configuración propuesta es que obtenemos la siguiente tabla para definir a las 4 articulaciones variables de nuestro robot, más una articulación fija que se ocupa de cambiar vincular dos sistemas coordenados que se encuentran desplazados uno del otro.

Articulación	Θ	d	a	α
1	90º	d1	0	0°
2	0º	0	a_2	0º
3	Θ_3	0	a_3	0º
4	Θ_4	0	a_4	0º
5	Θ_5	0	a_5	0º

Figura 3. Tabla de articulaciones obtenida a partir del algoritmo propuesto por Denavit-Hartenberg.

Simulación del espacio de trabajo en MATLAB.

Con el objetivo de simular las posiciones de nuestro robot para los diferentes valores que pueden tomar sus articulaciones se codificó las N matrices homogéneas obtenidas por el algoritmo de Denavit-Hartenberg y se obtuvo a partir de las mismas las cinco matrices que vinculan la posición de los extremos de cada eslabón referidos al sistema de coordenadas de la base.

Una vez realizada esta actividad fue posible visualizar las distintas posiciones que pueden ser alcanzadas por el extremo de nuestro robot. Las mismas fueron comprobadas por el sentido común, validando el modelo matemático hallado.

A continuación mostramos el script de MATLAB que hemos utilizado para desarrollar las diferentes transformaciones:

```
%% OBTENCION DE MATRIZ DE TRANSFORMACIÓN HOMOGÉNEA.

% Se obtiene la matriz de transformación homogénea aplicando el método de
% Denavit-Hartenberg para el esquema del ROBOT planteado.

%% DEFINO LAS VARIABLES DEL ROBOT
clear all;
close all;
syms t3 t4 t5;
syms d1;
syms a2 a3 a4 a5;

%% DEFINO LAS MATRICES HOMOGENEAS EN FUNCIÓN DE LAS VARIABLES
% Debo considerar que las transformaciones se logran como sigue:
% (i-1)A(i) = T(z,tita)T(0,0,di)T(ai,0,0)T(x,alfai);

% DEFINO LA MATRIZ 0A1 - Articulación 1
M_0A1 = ...
        [ 0           1          0          0 ...
        ;-1           0          0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          0   ...
        ; 0           1          0          0   ...
        ; 0           0          1          d1  ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 1A2 - Articulación 2
M_1A2 = ...
        [ 1           0          0          a2  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 2A3 - Articulación 3
M_2A3 = ...
        [ cos(t3)    -sin(t3)    0          0 ...
        ; sin(t3)     cos(t3)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a3  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 3A4 - Articulación 4
M_3A4 = ...
        [ cos(t4)    -sin(t4)    0          0 ...
        ; sin(t4)     cos(t4)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a4  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 4A5 - Articulación 5
M_4A5 = ...
        [ cos(t5)    -sin(t5)    0          0 ...
        ; sin(t5)     cos(t5)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a5  ...
        ; 0           1          0          0   ...
```

```
        ; 0           0          1           0   ...
        ; 0           0          0           1 ] ;

% SE COMPONE LA MATRIZ DE TRANSFORMACIÓN HOMOGENEA T.
% Obtengo la matríz de transformación.
M_0A5 = M_0A1 * M_1A2 * M_2A3 * M_3A4 * M_4A5;
T = M_0A5;

%% SIMULACIÓN DEL ESPACIO DE TRABAJO
% Cargo los parámetros
t5 = 0;
t3 = 0;
t4 = 0;
d1 = 50;
a2 = 10;
a3 = 56;
a4 = 56;
a5 = 10;
% Defino el origen de cualquier sistema.
X0 = [ 0 ; ...
       0 ; ...
       0 ; ...
       1 ] ;
% Luego, será posible obtener las coordenadas del extremo del ROBOT
% en el sistema de coordenadas base con el cálculo siguiente:
%   Xe = T*X0;
%   eval(Xe)
% PRESENTO EL ESPACIO DE TRABAJO TEORICO.
P = zeros(4,7986);
i = 0;
for d1 = 0:20:100
    for t3 = -pi/2:pi/10:pi/2
        for t4 = -pi/2:pi/10:pi/2
            for t5 = -pi/2:pi/10:pi/2
                i = i + 1;
                P(:,i) = eval(T) * X0;
            end
        end
    end
end
figure;
% Grafico el origen de cordenadas
plot3(0,0,0,'x','Color','black');
grid on;
box on;
xlabel('Eje X');
ylabel('Eje Y');
zlabel('Eje Z');
axis ([-200 200 -200 200 -200 200]);
hold on;
plot3(P(1,:), P(2,:), P(3,:),'Color', 'blue');
%% REALIZAMOS UNA SIMULACION DE LOS ESLABONES DEL ROBOT.
% Presento los nodos
P = zeros(4,6);
% Defino una interfaz grafica.
figure;
% Simulo la cadena cinemática.
for t4 = -pi/2:pi/2:pi/2
    for t3 = -pi/2:pi/2:pi/2
        for d1 = 0:10:100
            for t5 = -pi/2:pi/2:pi/2
                % Calculo la posición de las articulaciones
                P(:,1) = X0;
                T = M_0A1;
                P(:,2) = eval(T) * X0;
                T = T * M_1A2;
```

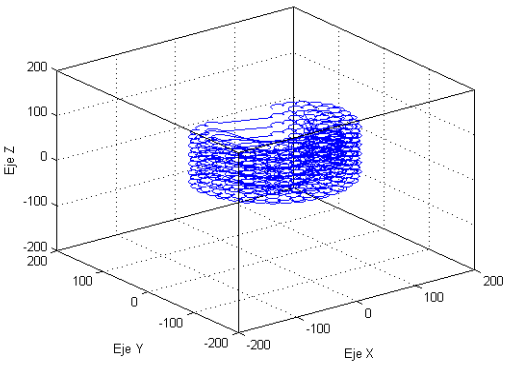
```
                P(:,3) = eval(T) * X0;
                T = T * M_2A3;
                P(:,4) = eval(T) * X0;
                T = T * M_3A4;
                P(:,5) = eval(T) * X0;
                T = T * M_4A5;
                P(:,6) = eval(T) * X0;
                % Grafico el origen de cordenadas
                plot3(0,0,0,'x','Color','black');
                grid on;
                box on;
                xlabel('Eje X');
                ylabel('Eje Y');
                zlabel('Eje Z');
                axis ([-100 100 -100 100 -100 100]);
                hold on;
                %plot3(X,Y,Z,'Color','red');
                i = 1;
                plot3(P(1,i:i+1), P(2,i:i+1), P(3,i:i+1), 'Color','black');
                i = 2;
                plot3(P(1,i:i+1), P(2,i:i+1), P(3,i:i+1), 'Color','blue');
                i = 3;
                plot3(P(1,i:i+1), P(2,i:i+1), P(3,i:i+1), 'Color','green');
                i = 4;
                plot3(P(1,i:i+1), P(2,i:i+1), P(3,i:i+1), 'Color','red');
                i = 5;
                plot3(P(1,i:i+1), P(2,i:i+1), P(3,i:i+1), 'Color','blue');
                hold off;
                pause(0.1);
            end
        end
    end
end
```

Figura 3. Código Fuente para MATLAB que compone la matriz de transformación homogénea y representa gráficamente el espacio de trabajo.

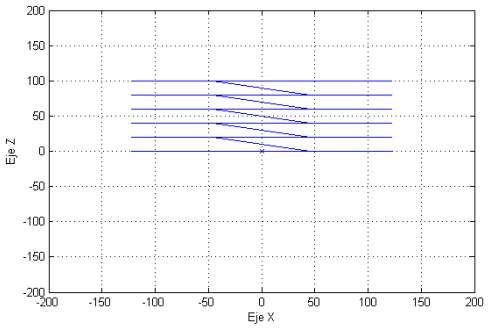
Los resultados de la simulación del espacio de trabajo obtenidos se presentan a continuación:

a

b



c



d

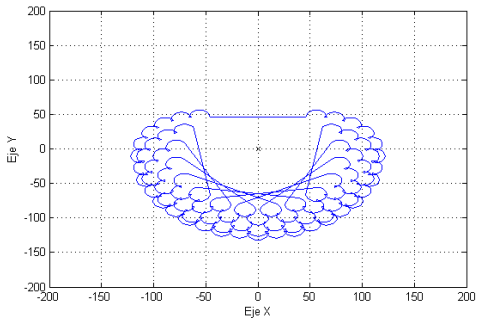


Figura 4. Espacio de trabajo. a) Modelo 3D del robot orientado en forma análoga a la representación espacial realizada por MATLAB. b) Espacio de trabajo obtenido a partir del modelo matemático definido en MATLAB. c) Vista superior de la mesa de trabajo. d) vista lateral de la mesa de trabajo.

Implementación de la matriz de transformación en CodeWarrior.

El código a implementar en el DSP pretende obtener las coordenadas del extremo del ROBOT al variar los ángulos de sus articulaciones. Utilizando MATLAB fue posible definir las ecuaciones que establecen los valores de las coordenadas del extremo del robot en función de los ángulos de rotación de las articulaciones rotacionales y los desplazamientos en las articulaciones prismáticas. Las mismas se presentan a continuación:

$$
\begin{aligned}
x_e = \quad & a_3 \sin(t_3) + \\
& a_5 \cos(t_5)[\cos(t_3)\sin(t_4) + \cos(t_4)\sin(t_3)] + \\
& a_5 \sin(t_5)[\cos(t_3)\cos(t_4) - \sin(t_3)\sin(t_4)] + \\
& a_4 \cos(t_3)\cos(t_4) + \\
& a_4 \cos(t_4)\sin(t_3) \\
y_e = \quad & a_5 \sin(t_5)[\cos(t_3)\sin(t_4) + \cos(t_4)\sin(t_3)] - \\
& a_3 \cos(t_3) - \\
& a_5 \cos(t_5)[\cos(t_3)\cos(t_4) - \sin(t_3)\sin(t_4)] - \\
& a_2 - \\
& a_4 \cos(t_3)\cos(t_4) + \\
& a_4 \sin(t_3)\sin(t_4) \\
z_e = \quad & d_1
\end{aligned}
$$

Se observa a simple vista que las ecuaciones presentadas son excesivamente complejas y difíciles de relevar a la hora de comparar resultados entre la simulación con la herramienta matemática y los valores obtenidos en la simulación del DSP. Para sortear esta dificultad, planteamos un código para DSP que aprovecha la facilidad de procesar matrices que poseen estos dispositivos. Como resultado obtuvimos un código fuente que resulta más fácil de comprender, corregir y mantener.

```
/** ###################################################################
** Filename : TPN1.C
** Project : TPN1
** Processor : 56F8367
** Version : Driver 01.14
** Compiler : Metrowerks DSP C Compiler
** Date/Time : 10/06/2012, 08:06 p.m.
** Abstract :
** Main module.
** This module contains user's application code.
** Settings :
** Contents :
** No public methods
**
** ###################################################################*/
/* MODULE TPN1 */
/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.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"
/* Defino los rangos de barrido de las variables. */
#define D1_INI 0
#define D1_PASO FRAC16(0.1333)/*20/150*/
#define D1_FIN FRAC16(0.6667)/*100/150*/
#define T_INI FRAC16(-0.5)
#define T_PASO FRAC16(0.1)
#define T_FIN FRAC16(0.5)
#define MENOS_UNO FRAC16(-1)
void main(void)
{
       /* Defino los elementos fijos del ROBOT */
       Frac16 a2=FRAC16(0.0667/*10/150*/);
       Frac16 a3=FRAC16(0.3733/*56/150*/);
       Frac16 a4=FRAC16(0.3733/*56/150*/);
       Frac16 a5=FRAC16(0.0667/*10/150*/);
       /* Write your local variable definition here */
       Frac16 d1;
       Frac16 t3, t4, t5;
       Frac16 M_0A1[4][4] = {
       {FRAC16(0),FRAC16(1),FRAC16(0),FRAC16(0)},
              {FRAC16(-1),FRAC16(0),FRAC16(0),FRAC16(0)},
              {FRAC16(0),FRAC16(0),FRAC16(1),0/*d1*/},
              {FRAC16(0),FRAC16(0),FRAC16(0),FRAC16(1)}
       };
       Frac16 M_1A2[4][4] = {
              {FRAC16(1),FRAC16(0),FRAC16(0),0/*a2*/},
```

```
		{FRAC16(0),FRAC16(1),FRAC16(0),FRAC16(0)},
		{FRAC16(0),FRAC16(0),FRAC16(1),FRAC16(0)},
		{FRAC16(0),FRAC16(0),FRAC16(0),FRAC16(1)}
	};
	Frac16 M_2A3[4][4] = {
		{0/*cos(t3)*/,0/*-sin(t3)*/,FRAC16(0),0/*a3*cos(t3)*/},
		{0/*sin(t3)*/,0/*cos(t3)*/,FRAC16(0),0/*a3*sin(t3)*/},
		{FRAC16(0),FRAC16(0),FRAC16(1),FRAC16(0)},
		{FRAC16(0),FRAC16(0),FRAC16(0),FRAC16(1)}
	};
	Frac16 M_3A4[4][4] = {
		{0/*cos(t4)*/,0/*-sin(t4)*/,FRAC16(0),0/*a4*cos(t4)*/},
		{0/*sin(t4)*/,0/*cos(t4)*/,FRAC16(0),0/*a4*sin(t4)*/},
		{FRAC16(0),FRAC16(0),FRAC16(1),FRAC16(0)},
		{FRAC16(0),FRAC16(0),FRAC16(0),FRAC16(1)}
	};
	Frac16 M_4A5[4][4] = {
		{0/*cos(t5)*/,0/*-sin(t5)*/,FRAC16(0),0/*a5*cos(t5)*/},
		{0/*sin(t5)*/,0/*cos(t5)*/,FRAC16(0),0/*a5*sin(t5)*/},
		{FRAC16(0),FRAC16(0),FRAC16(1),FRAC16(0)},
		{FRAC16(0),FRAC16(0),FRAC16(0),FRAC16(1)}
	};
	Frac16 M_0A2[4][4],M_0A3[4][4],M_0A4[4][4],M_0A5[4][4];
	Frac16 sent3, sent4, sent5, cost3, cost4, cost5;
	/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
	PE_low_level_init();
	/*** End of Processor Expert internal initialization. ***/
	/* Write your code here */
	/* Código propio para calcular los límites del espacio de trabajo. */
	M_1A2[0][3]=a2;
	for(d1=D1_INI;d1<=D1_FIN;d1+=D1_PASO) {
		M_0A1[2][3]=d1;
		for(t3=T_INI;t3<=T_FIN;t3+=T_PASO) {
			sent3 = tfr16SinPIx(t3);
			cost3 = tfr16CosPIx(t3);
			M_2A3[0][0] = cost3;
			M_2A3[0][1] = mult(MENOS_UNO,sent3);
			M_2A3[0][3] = mult(a3,cost3);
			M_2A3[1][0] = sent3;
			M_2A3[1][1] = cost3;
			M_2A3[1][3] = mult(a3,sent3);
			for(t4=T_INI;t4<=T_FIN;t4+=T_PASO) {
				sent4 = tfr16SinPIx(t4);
				cost4 = tfr16CosPIx(t4);
				M_3A4[0][0] = cost4;
				M_3A4[0][1] = mult(MENOS_UNO,sent4);
				M_3A4[0][3] = mult(a4,cost4);
				M_3A4[1][0] = sent4;
				M_3A4[1][1] = cost4;
				M_3A4[1][3] = mult(a4,sent4);
				for(t5=T_INI;t5<=T_FIN;t5+=T_PASO) {
					sent5 = tfr16SinPIx(t5);
					cost5 = tfr16CosPIx(t5);
					M_4A5[0][0] = cost5;
					M_4A5[0][1] = mult(MENOS_UNO,sent5);
					M_4A5[0][3] = mult(a5,cost5);
					M_4A5[1][0] = sent5;
```

```
                                M_4A5[1][1] = cost5;
                                M_4A5[1][3] = mult(a5,sent5);
                                xfr16Mult((Frac16 *)M_0A1,4,4,(Frac16 *)M_1A2,4,
                                        (Frac16 *)M_0A2);
                                xfr16Mult((Frac16 *)M_0A2,4,4,(Frac16 *)M_2A3,4,
                                        (Frac16 *)M_0A3);
                                xfr16Mult((Frac16 *)M_0A3,4,4,(Frac16 *)M_3A4,4,
                                        (Frac16 *)M_0A4);
                                xfr16Mult((Frac16 *)M_0A4,4,4,(Frac16 *)M_4A5,4,
                                        (Frac16 *)M_0A5);
                                printf("d1;%d;t3;%d;t4;%d;t5;%d;x;%d;y;%d;z;%d\n",
                                        d1,t3,t4,t5,M_0A5[0][3],M_0A5[1][3],M_0A5[2][3]);
                        }
                }
        }
        /* Watchdog */
        for(;;) {}
}
/* END TPN1 */
/*
** ###################################################################
**
** This file was created by Processor Expert 3.00 [04.35]
** for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/
```

Figura 5. Código Fuente en lenguaje C para el procesador DSP 56F6387 que utiliza las matrices modeladas en MATLAB para obtener las posiciones del extremo del manipulador.

El resultado obtenido fue exportado a MATLAB para su representación con el siguiente resultado:

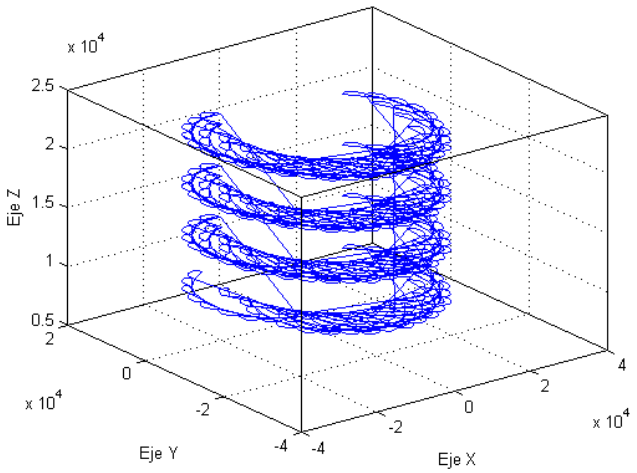


Figura 6. Representación del espacio de trabajo logrado con el DSP.

Comparando la figura 4 con la figura 6 se observan las similitudes entre los espacios de trabajo logrados con la herramienta de simulación matemática y el proceso implementado sobre el DSP. Las diferencias que existen entre una y otra se debe a que para la implementación del proceso sobre el DSP hubo que normalizar el espacio de trabajo y los puntos barridos por dicho código se redujeron sustancialmente.

En el presente trabajo el espacio de trabajo de simulado en MATLAB (-150 .. 150, -150 .. 150, -150 .. 150) fue normalizado a (-1 .. 1, -1 .. 1, -1 .. 1) para aprovechar al máximo la resolución brindada por las variables FracXX del DSP.

Conclusiones

En primera instancia, observamos la simplificación producida sobre la complejidad de la actividad de modelado matemático mediante la aplicación del algoritmo de Denavit-Hartenberg. Si bien la aplicación de dicho algoritmo es laboriosa y por demás abstracta, resulta mucho menos compleja que la deducción de algún método ad-hoc para lograr un modelo matemático que describa la configuración utilizada, sobre todo cuando en dicha configuración el número de eslabones es elevado.

En segunda instancia, el código fuente que obtuvimos al mantener las matrices como herramienta matemática para la transformación de los puntos de los diferentes sistemas coordenados al sistema coordenado de la base del robot presenta la información de una forma más clara y concisa. Fue en este escenario donde se pudieron explotar las funciones dedicadas del DSP quedando demostrada la ventaja tecnológica que este tipo de dispositivos provee para este tipo de aplicaciones.

Cabe destacar la importancia del cambio de filosófico que hubo que adoptar para implementar el proceso de cálculo sobre el DSP. Este dispositivo esta pensado para trabajar con señales adquiridas a través de conversores ADC y DAC, por lo cual utiliza tipos de variables especiales que maximizan la resolución del dato almacenado en las mismas. Dicho tipo de variable es conocido como **FracXX**, existiendo una versión de 16 y otra de 32 bits, y representan un número fraccional entre -1 y 1 como un valor que excursiona entre el valor mínimo y máximo de la representación lograda con la cantidad de bits que correspondan (Para el caso de 16 bits excursiona entre -32768 y 32767 subdividiendo el rango en 65535 partes) obteniendo así una excelente resolución.