**UNIVERSIDAD TECNOLOGICA NACIONAL**
**FACULTAD REGIONAL BUENOS AIRES**

# ROBOTICA

TP1: Cinemática directa sobre DSP

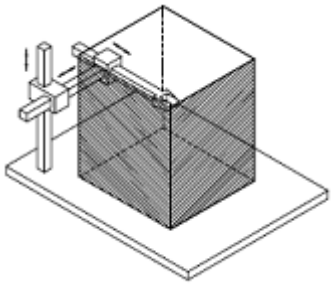
**AÑO:** 2010

**Integrantes:**

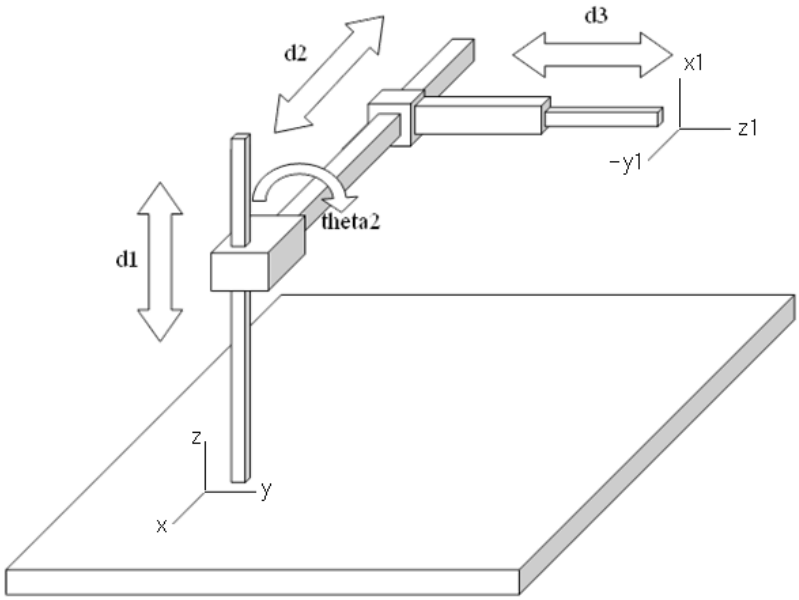
Baldo, Matías
Valenzuela, Fernando

## Introducción sobre la cinemática del Robot:

Para empezar mostraremos el robot a modelizar:



Este robot es un manipulador de 3 grados de libertad, nosotros lo que haremos será agregarle un grado más de libertad. Ésta modificación permitirá a uno de sus brazos girar sobre uno de sus ejes:




Como herramienta matemática para desarrollar las traslaciones de los diferentes ejes coordenados, usaremos la matriz homogénea de transformación. De esta manera cada eje transformado, obtendrá una matriz homogénea donde el producto sucesivo de estas nos convertirá a un punto al final del brazo en un punto espacial relativo a un sistema fijo en la base del robot. Para generar las diferentes matrices nos basaremos en la siguiente tabla:

| Articulación | θ | d | a | α |
|---|---|---|---|---|
| 1 | -90 | d1 | 0 | 90 |
| 2 | 90+alfa2 | 0 | 0 | -90 |
| 3 | 0 | d2 | 0 | 0 |
| 4 | 0 | d3 | 0 | 0 |

Como herramienta para manejar y generar las matrices se utilizó un script en Matlab:

```
syms theta;
syms d;
syms alfa;
syms a;
syms tetha2;
syms d1;
syms d2;
syms d3;

A = [cos(theta) -sin(theta)*cos(alfa) sin(alfa)*sin(theta)
a*cos(theta);
     sin(theta) cos(alfa)*cos(theta) -sin(alfa)*cos(theta)
a*sin(theta);
     0 sin(alfa) cos(alfa) d;
     0 0 0 1];

%A01
theta = -pi/2;
alfa = pi/2;
d=d1;
a=0;
A01 = eval(A)

%A02
theta = pi/2+tetha2;
alfa = 0;
d=0;
a=0;
A02 = eval(A)

%A03
theta = 0;
alfa = -pi/2;
d=d2;
a=0;
A03 = eval(A)


%A04
theta = 0;
alfa = 0;
d=d3;
a=0;
A04 = eval(A)


%Una vez resuelta y redondeando los números nos da:

A01 = [  0, 0, -1,  0;
        -1, 0,  0,  0;
         0, 1,  0, d1;
         0, 0,  0,  1]
```

```
A02 = [ -sin(tetha2),  -cos(tetha2),  0, 0 ;
         cos(tetha2),  -sin(tetha2),  0, 0 ;
                   0,             0,  1, 0 ;
                   0,             0,  0, 1 ]


A03 = [ 1,  0, 0,  0;
        0,  0, 1,  0;
        0, -1, 0, d2;
        0,  0, 0,  1]

A04 = [ 1,  0, 0,  0;
        0,  1, 0,  0;
        0,  0, 1, d3;
        0,  0, 0,  1]

%Matriz transformada final:
T = A01*A02*A03*A04

syms x1;
syms y1;
syms z1;

X3 = T*[x1;y1;z1;1]

%Como nos interesa conocer la transformada en el punto final del
robot, evaluamos la matriz en el origen del ultimo sistema de
coordenadas:
x1=0;
y1=0;
z1=0;

eval(X3)
```

Al ejecutar dicho script nos devuelve:

```
A01 =                    A02 =

[  0, 0, -1,  0]         [ -sin(tetha2), -cos(tetha2), 0, 0]
[ -1, 0,  0,  0]         [  cos(tetha2), -sin(tetha2), 0, 0]
[  0, 1,  0, d1]         [            0,            0, 1, 0]
[  0, 0,  0,  1]         [            0,            0, 0, 1]




A03 =                    A04 =

[ 1,  0, 0,  0]           [ 1, 0, 0,  0]
[ 0,  0, 1,  0]           [ 0, 1, 0,  0]
[ 0, -1, 0, d2]           [ 0, 0, 1, d3]
[ 0,  0, 0,  1]           [ 0, 0, 0,  1]
```

```
T =

[           0, 1,            0,                  -d2]
[ sin(tetha2), 0,  cos(tetha2),      d3*cos(tetha2)]
[ cos(tetha2), 0, -sin(tetha2), d1 - d3*sin(tetha2)]
[           0, 0,            0,                    1]


X3 =

                                                  y1 - d2
      d3*cos(tetha2) + z1*cos(tetha2) + x1*sin(tetha2)
 d1 - d3*sin(tetha2) + x1*cos(tetha2) - z1*sin(tetha2)
                                                        1



X3 =

                 -d2
      d3*cos(tetha2)
 d1 - d3*sin(tetha2)
                   1
```

Esta última ecuación final es la que nos interesa, ya que esta es la que nos permite obtener la coordenada de la punta del brazo en el sistema de coordenadas origen. Esto último se lo puede ver de la siguiente forma:

X = -d2
Y = d3*cos(tetha2)
Z = d1 - d3*sin(tetha2)

## Desarrollo e implementación en Codewarrior DSP 56800-E:

Para corroborar la correcta implementación de las matrices se programó sobre un DSP una rutina que recorre el espacio geométrico que genera los posibles movimientos del brazo.

El primer paso que tomamos fue el de generar una rutina que simplemente ejecute los posibles movimientos solo teniendo en cuenta la limitación física de que la punta del brazo no puede atravesar el piso, es decir Z>0 para todo movimiento:

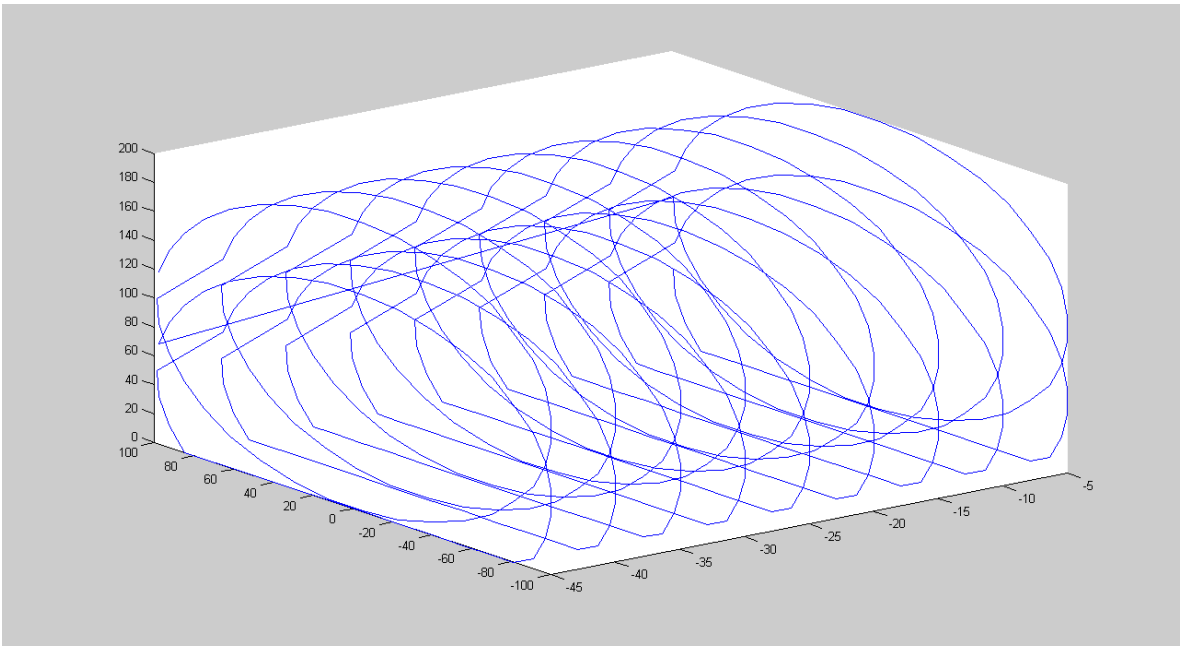
```
#define MXRAD 35
#define PULSE2RAD 32767/MXRAD

#define MAX_INT 65535

#define MIN_D1 5
#define MIN_D2 5
#define MIN_D3 5

#define MAX_D1 100
#define MAX_D2 100
#define MAX_D3 100

#define STEP_D1       5
#define STEP_D2       5
#define STEP_D3       5


Frac16 c;
Frac16 pulse2rad=PULSE2RAD;

int i;

void main(void)
{

        Word16 local_c16;
        Word32 local_c32;

        unsigned int d1, d2, d3,theta2;
        Frac16 fd1, fd2, fd3, aux;

        Frac16 x, y, z;
        Frac16 seno, coseno;

        PE_low_level_init();



/*
Esta rutina prueba los limites del manipulador:
        d3: Fijo en la posición máxima
        d1: Fijo en dos valores característicos (punto medio y punto máximo)
        d2: Barrido desde el mínimo al máximo, de a pasos discretos
        theta2: Barrido de toda la circunferencia, de a pasos discretos
*/

        d3 = MAX_D3;
        d1 = 0;
```

```
	do
	{
		i++;
		d1 += 50;
		for(d2=MIN_D2;d2<MAX_D2;d2+=STEP_D2)
		{
			for(theta2=0;theta2<MXRAD;theta2++)
			{

					local_c32=(L_mult(pulse2rad,theta2));
					local_c16=extract_l(local_c32);

					//	x = -d2;
					//	y = cos(theta2)*d3;
					//	z = -sin(theta2)*d3 + d1

					seno=TFR1_tfr16SinPIx(local_c16);
					coseno=TFR1_tfr16CosPIx(local_c16);

					x = negate(d2);
					y = mult(coseno,d3);
					aux = mult(seno,d3);
					z = sub(d1, aux);

					if(z<=0)
						z = 0;

					printf ("%d\t%d\t%d\n",x, y, z);

			}

		}

	}while(i<2);

	printf("TERMINE!\n");
	while(1);
}
```

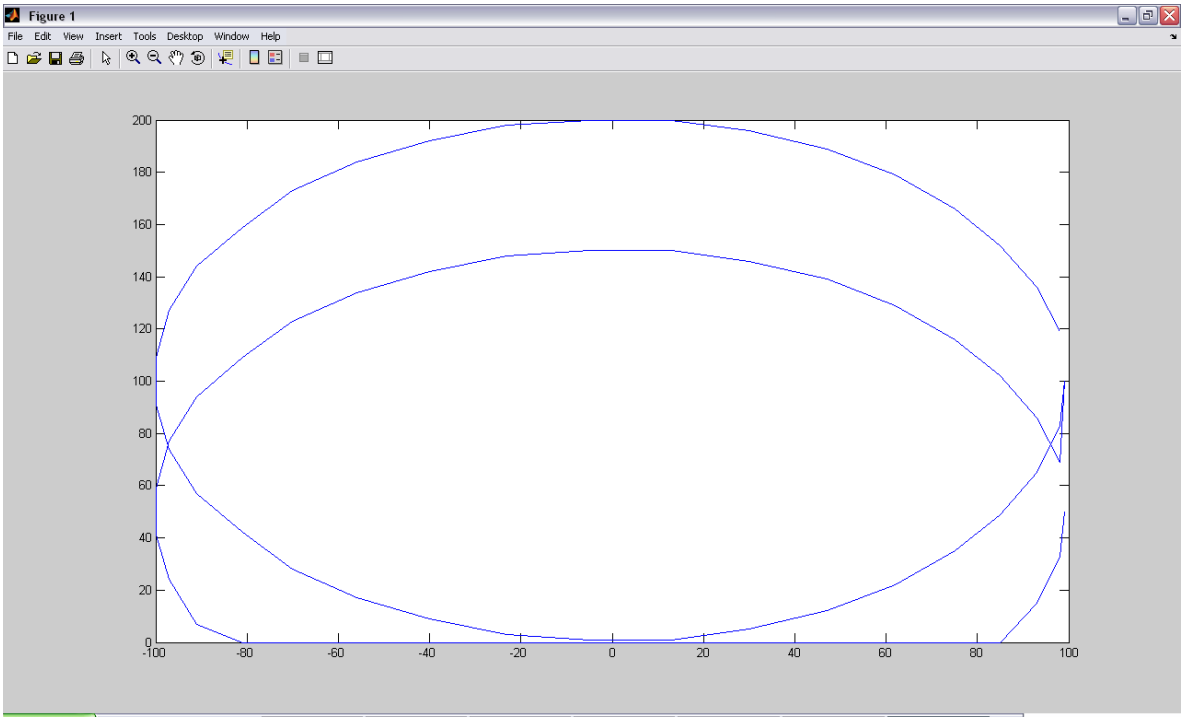
El código del ejemplo genera unos puntos característicos del manipulador:

- La longitud d3 del brazo fija en su posición máxima
- Un barrido de 360 grados sobre la rotación del ángulo θ2
- Un barrido de la longitud d2 entre 5 y 45
- Dos puntos característicos de d1: Uno en su punto máximo y otro en un punto intermedio donde se debe limitar el valor de d3 de manera que la coordenada en z no resulte negativa.

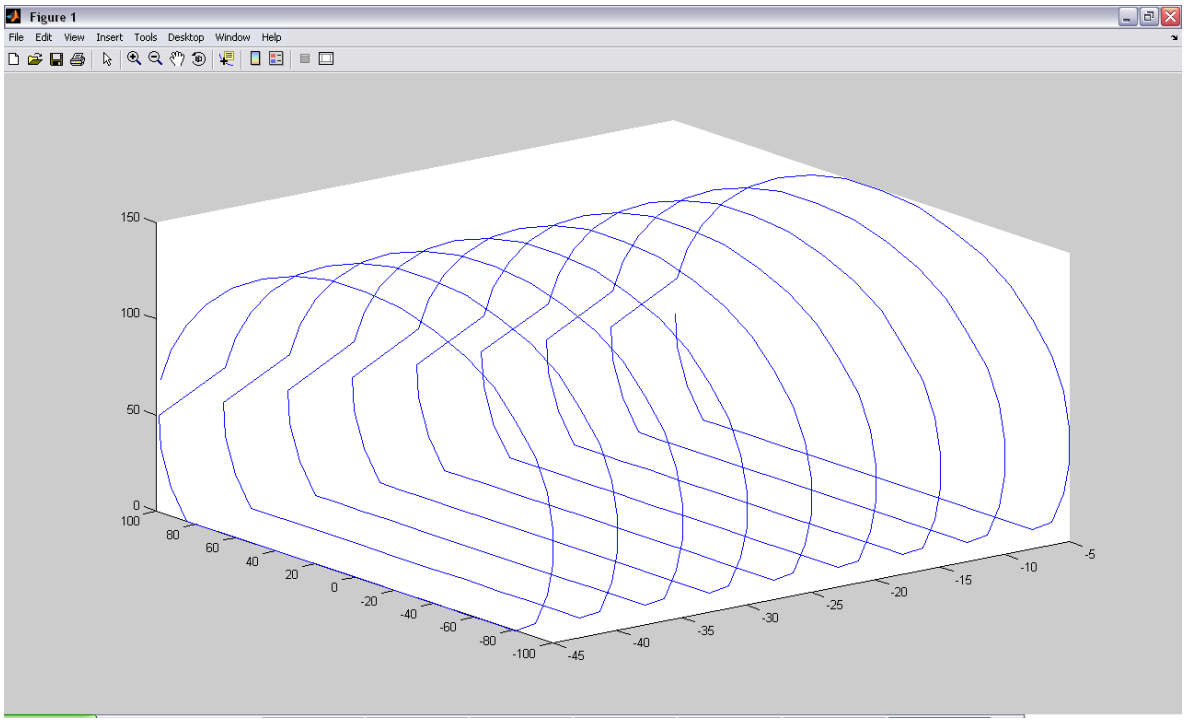
La consola de la simulación es guardada en un archivo .xls, que luego es importado en Matlab para graficar los puntos resultantes. En la figura siguiente, podemos observar como se diferencian claramente dos figuras, una circunferencia en la parte superior y una circunferencia truncada sobre el eje y. Estas dos figuras son extendidas sobre el eje x para lograr una figura cuasi-cilíndrica.



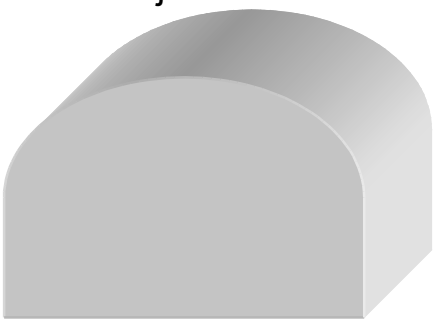

Un corte sobre el plano Y-Z muestra la forma anterior:




Y en la siguiente figura, podemos observar una simplificación del caso, donde solamente se traza el camino que realiza la punta del robot para un valor determinado de d1:

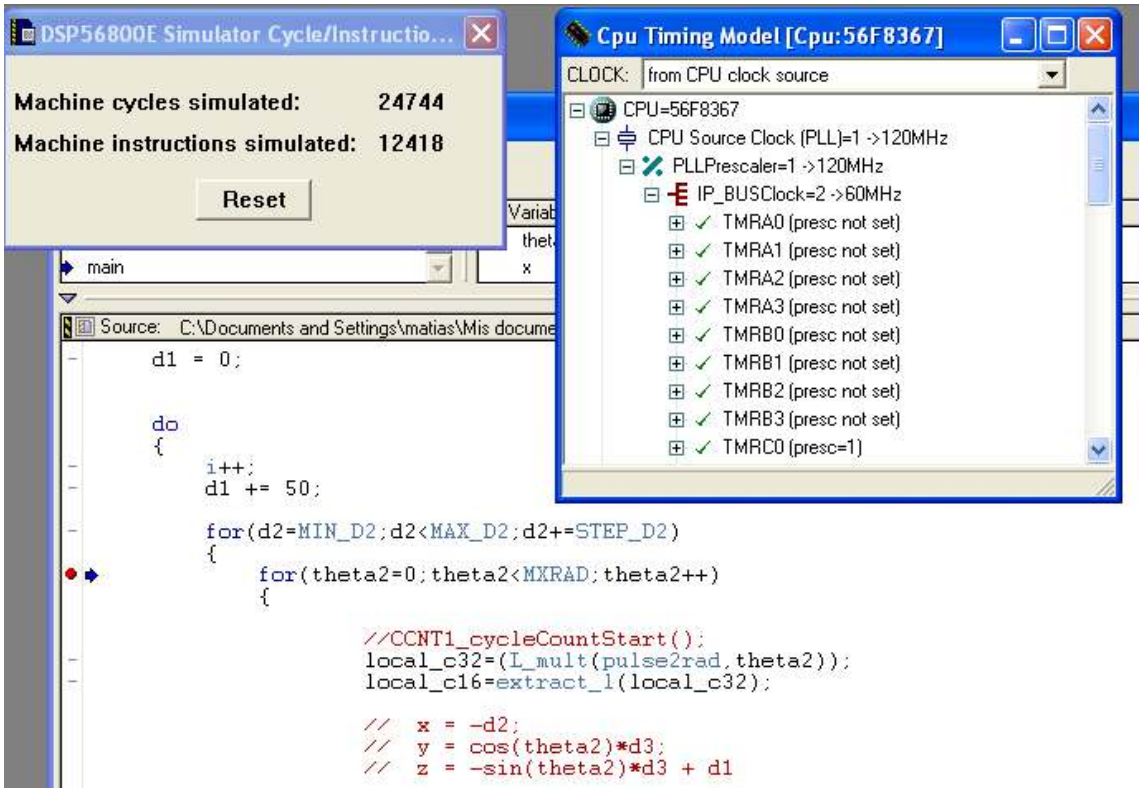



En el gráfico siguiente se puede observar en forma cualitativa la apariencia del área de trabajo.



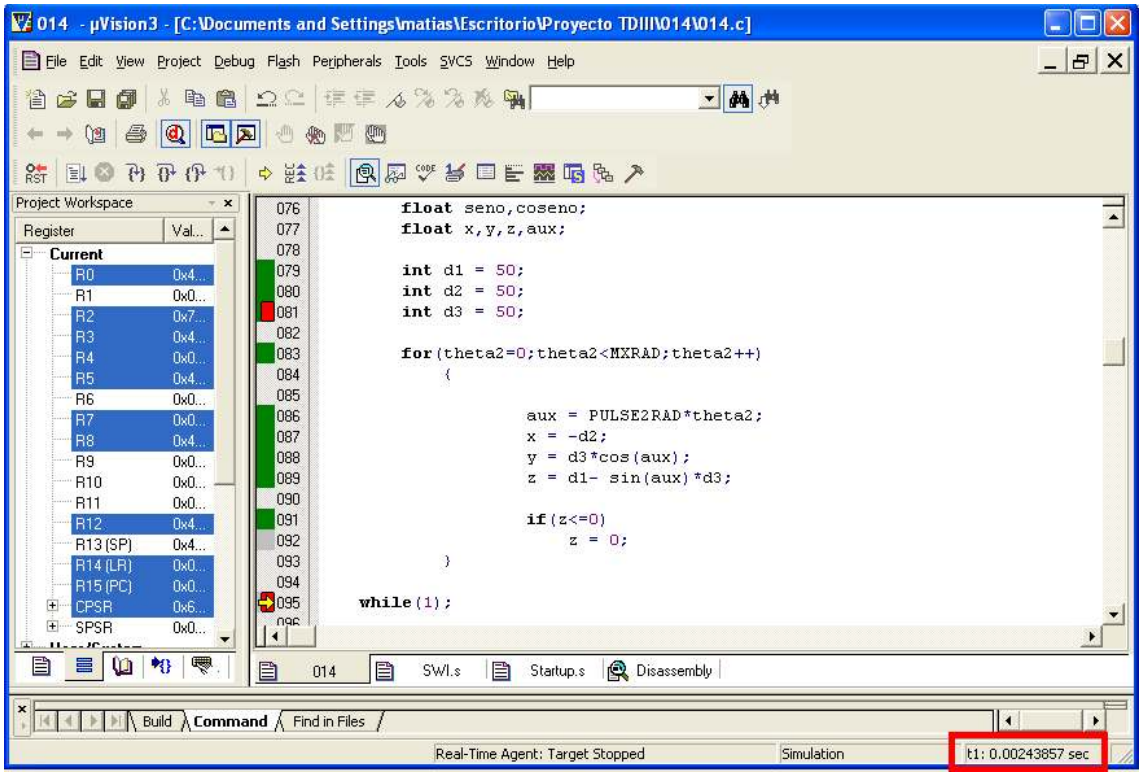
Es de notar que el valor mínimo de d3 limita el movimiento en el interior de esa figura, lo que provoca que un área cilíndrica en medio de la figura marcada no sea accesible.

Se realizó una simulación para evaluar los tiempos de ejecución de este procesador:

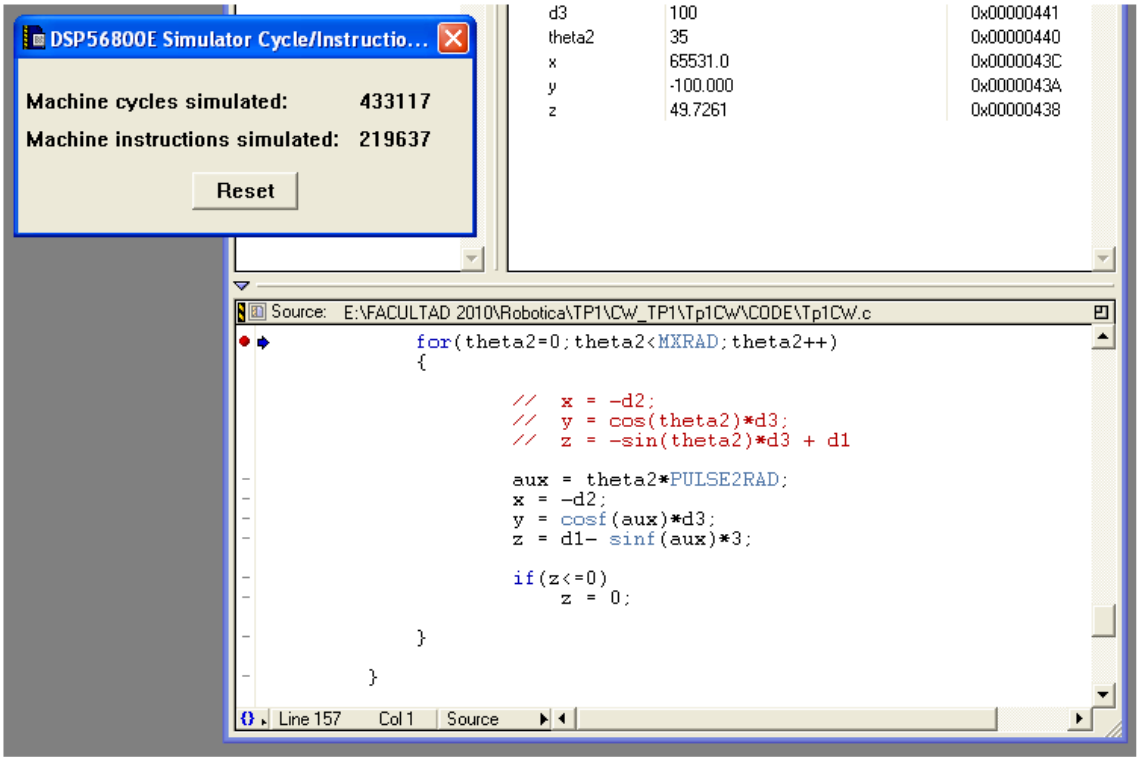



En esta podemos ver que se realizo un barrido de 360 grados sobre el angulo theta2 en 35 pasos discretos. Dicho barrido se realizó 24744 ciclos de maquina, este corre a 60 MHz ejecutandose sobre la memoria flash. Como resultado tenemos que tarda 410useg aproximadamente.

Para realizar una comparación con un microcontrolador sin extensiones DSP, con un core ARM7-TDMII LPC2148 corriendo casualmente a 60MHz, nos dá un tiempo de ejecución de 2,4 mseg. Tambien es importante recalcar que se utilizaron variables del tipo flotante y no fraccionales, dado que el entorno de desarrollo no provee dichas facilidades.




Para sacar una conclusión más decidimos correr la misma rutina pero utilizando punto flotante, y de esta manera comparar arquitecturas. Obtuvimos el siguiente resultado:




Calculando el tiempo de ejecución, llegamos a que para ejecutar la rutina demora 7,2 mseg aproximadamente.

Debido a la lentitud con la que el simulador genera una tabla con los datos de los movimientos decidimos usar el DSP tan solo para ejecutar un script, manipulando solo las ecuaciones sin tomar en cuenta los limites físicos del brazo. Por eso armamos una segunda función donde tiene en cuenta los límites físicos del robot y si se encuentra un punto peligroso para este no lo deja moverse:

```
#define MIN_D1 5
#define MIN_D2 5
#define MIN_D3 5

#define MAX_D1 100
#define MAX_D2 100
#define MAX_D3 100

#define MAX_ALPHA 10
#define MIN_ALPHA -10

#define ZMIN    0

#define ZONA_X  -50
#define ZONA_Y  -70



int VerificaMovimiento(d1,d2,d3,alpha2)
{
        if(d1>MAX_D1)       // Verifica que esten dentro de los limites
                return 0;      // posible de movimiento del brazo
        if(d2>MAX_D2)
                return 0;
        if(d3>MAX_D3)
                return 0;

        if(d1<MIN_D1)
                return 0;
        if(d2<MIN_D2)
                return 0;
        if(d3<MIN_D3)
                return 0;

        if(alpha2>MAX_ALPHA)
                return 0;

        if(alpha2<MIN_ALPHA)
                return 0;


        seno=TFR1_tfr16SinPIx(local_c16);
        coseno=TFR1_tfr16CosPIx(local_c16);

        x = negate(d2);
        y = mult(coseno,d3);
        aux = mult(seno,d3);
        z = sub(d1, aux);      // Obtengo las coordenadas donde
                               // se encuentra el brazo

        // Verifica condiciones fisicas

        if(z<ZMIN)                    // Que no cruce el piso
                return 0;

                // El siguiente codigo es opcional
        if(x< ZONA_X)                         // Creo una zona rectangular donde
                if(y < ZONA_Y)                // no puede trabajar el brazo
                        return 0;

        return 1;
}
```

En el main se ejecuta:

```
for(d1=MIN_D1;d1<MAX_D1;d1+=STEP_D1)
{
        for(d2=MIN_D2;d2<MAX_D2;d2+=STEP_D2)
        {
                for(theta2=0;theta2<MXRAD;theta2++)
                {

                        for(d3=MIN_D3;d3<MAX_D3;d3+=STEP_D3)
                        {

                                if(VerificaMovimiento(d1,d2,d3,theta2))
                                {
                                        RealizaMovientos(d1,d2,d3,theta2)
                                }

                        }

                }

        }

 }
```

En este caso estaría usándose el DSP para calcular el punto final, si se encuentra dentro de una zona de trabajo admisible, realiza el movimiento.

## Conclusiones:

Durante el desarrollo de la primera parte pudimos aplicar desde cero el algoritmo Denavit-Hartenberg para obtener el modelo cinemático completo del sistema. Durante la resolución del metodo y para acelerar el tiempo de resolucion pudimos utilizar satisfactoriamente Matlab en el manejo de las matrices.

Una vez obtenidas las ecuaciones que modelan al sistema, pasamos a trabajar sobre el entorno CodeWarrior y de esta forma programar el DSP requerido. Dicho entorno tiene integradas una gran cantidad de funciones/componentes que aceleran los tiempos de programación y ayudan de gran manera la performance del DSP, haciendo uso de sus caracteristicas. En nuestro caso se empleamos, durante todo el proceso, la notación fraccional; de esta manera, el procesador, trabaja entre los valores de 1 y -1 y no tiene la complejidad que impone un sistema como el flotante, logrando tiempos de ejecución más veloces. También se utilizaron funciones que aprovechan este tipo de dato para el cálculo del seno, coseno y operaciones matemáticas.

A la vez se ejecuto el mismo algoritmo sobre un microcontrolador que trabaja a la misma frecuencia que el DSP. Si bien el algoritmo que se ejecuto es el mismo, en el programa del microcontrolador usamos variables del tipo flotante, ya que el entorno de programación utilizado no poseía un tipo fraccional. Pudimos comparar los tiempos de ejecución en ambos casos, en el DSP los tiempos se encuentran en el orden de los 410us; en cambio en el ARM7 hablamos de 2,4ms. Ahora, cuando comparamos las arquitecturas utilizando el mismo tipo de datos, flotante de 32 bits, vemos que el ARM7 sale ventajoso, ya que tarda 2,4ms mientras que el DSP tarda 7,2ms. También es importante recalcar que el ARM7 es un microcontrolador de 32 bits mientras que el DSP es de 16 bits.

En definitiva, lo que hace ventajoso al DSP es el tipo de dato fraccional, aunque en el mercado existen microcontroladores con características para el procesamiento de datos. Más allá de la arquitectura utilizada, es necesario utilizar un sistema que cuyos tiempos de ejecución no sea mayor al de los movimientos físicos. Es fundamental elegir correctamente el sistema numerico a utilizar, ya que como vimos este baja increíblemente la performance aunque ambos métodos nos den resultados equivalentes.