TESIS FINAL

ANÁLISIS CINEMÁTICO Y DINÁMICO E IMPLEMENTACIÓN EN DSP DEL ROBOT M5

Materia: Robótica

División: R6055

Tema: Tesis final de Robótica

Profesor: Ing. Hernán Gianetta

JTP: Ing. Damian Granzella

Alumnos: Emiliano Statello
Leandro Arcusin

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ÍNDICE

Tabla de contenido

1. INTRODUCCIÓN

En el presente informe se desarrollará la tesis final de la materia de robótica del departamento de electrónica con sede UTN-FRBA.

En el curso se trabajo con los ***M5*** módulos de brazos robóticos didácticos del departamento de electrónica fabricados por Skaymec de 5 grados de libertad.

1.1. OBJETIVO

El objetivo de esta tesis es mostrar el trabajo realizado a lo largo de la cursada en donde se utilizaron conceptos de cinemática, dinámica, electrónica, mecánica, implementación de distintos software tales como Matlab®, programas de cads mecánicos, programación en DSP y FPGA para el control de los motores brushless. Aquí se notará cómo el desarrollo de la modelización del brazo robótico logra integrar los distintos conceptos adquiridos a lo largo de la carrera de ingeniería electrónica.

En nuestro caso particular se hará hincapié en la presentación de una herramienta ***GNU descriptor de software*** **GHDL** y de las herramientas para generar ***el compilador,*** **FLEX (scanner) y BISON (parser)**

1.2. ORGANIZACIÓN DEL TRABAJO

Este trabajo está dividido en tres partes principales.

En la primera parte se desarrollará la cinemática del brazo robótico, en la segunda respecto a la dinámica del robot y en la tercera sobre la generación del compilador.

Cada parte contará con una breve introducción al tema a desarrollar, la implementación con el modelo del brazo robótico, programación si la hubiese, y los resultados en Matlab.

2. CINEMÁTICA

2.1. LA CINEMÁTICA DIRECTA

La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia sin considerar las fuerzas que intervienen. Así, la cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo, y en particular por las relaciones entre la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares.

Existen dos problemas fundamentales a resolver en la cinemática del robot; el primero de ellos se conoce como el problema cinemático directo, y consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot; el segundo, denominado problema cinemático inverso, resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas.

En esta primera parte del trabajo, vamos a utilizar la cinemática directa para encontrar la posición en cada instante del extremo manipulador del robot.

Para la resolución del problema cinemático directo mediante matrices de transformación homogénea se utiliza fundamentalmente el álgebra vectorial y matricial para representar y describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo. Dado que un robot se puede considerar como una cadena cinemática formada por objetos rígidos o eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo situado en la base del robot y describir la localización de cada uno de los eslabones con respecto a dicho sistema de referencia. De esta forma, el problema cinemático directo se reduce a encontrar una matriz de transformación homogénea T que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base del mismo. Esta matriz T será función de las coordenadas articulares.

Con el modelo y los sistemas de referencia de cada una de las articulaciones planteados estamos en condiciones de obtener los parámetros de Denavit-Hartenberg que nos permitirán obtener las matrices transferencia parciales de cada articulación.

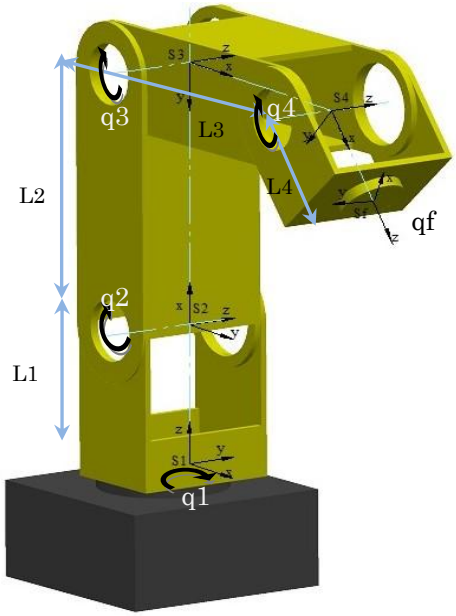
A continuación detallamos las 16 reglas que posee el algoritmo para que sea más clara la comprensión del resultado:

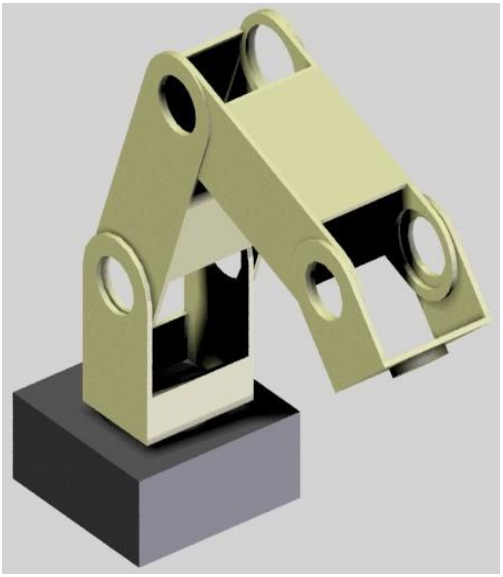
- **DH1.**Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerara como eslabón 0 a la base fija del robot.
- **DH2.**Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad y acabando en n).
- **DH3.**Localizar el eje de cada articulación. Si esta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.
- **DH4.**Para i de 0 a n-1, situar el eje Zi, sobre el eje de la articulación i+1.
- **DH5.**Situar el origen del sistema de la base (S0) en cualquier punto del eje Z0. Los ejes X0 e Y0 se situaran dé modo que formen un sistema dextrógiro con Z0.
- **DH6.**Para i de 1 a n-1, situar el sistema (Si) (solidario al eslabón i) en la intersección del eje Zi con la línea normal común a Zi-1 y Zi. Si ambos ejes se cortasen se situaría (Si) en el punto de corte. Si fuesen paralelos (Si) se situaría en la articulación i+1.
- **DH7.**Situar Xi en la línea normal común a Zi-1 y Zi.
- **DH8.**Situar Yi de modo que forme un sistema dextrógiro con Xi y Zi.
- **DH9.**Situar el sistema (Sn) en el extremo del robot de modo que Zn coincida con la dirección de Zn-1 y Xn sea normal a Zn-1 y Zn.
- **DH10.**Obtener Øi como el ángulo que hay que girar en torno a Zi-1 para que Xi-1 y Xi queden paralelos.
- **DH11.**Obtener Di como la distancia, medida a lo largo de Zi-1, que habría que desplazar (Si-1) para que Xi y Xi-1 quedasen alineados.
- **DH12.**Obtener Ai como la distancia medida a lo largo de Xi(que ahora coincidiría con Xi-1) que habría que desplazar el nuevo (Si-1) para que su origen coincidiese con (Si).
- **DH13.**Obtener ai como el ángulo que habría que girar entorno a Xi (que ahora coincidiría con Xi-1), para que el nuevo (Si-1) coincidiese totalmente con (Si).
- **DH14.**Obtener las matrices de transformación i-1Ai.
- **DH15.**Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot T = 0Ai, 1A2... n-1An.
- **DH16.**La matriz T define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las n coordenadas articulares.

2.2. APLICACIÓN DE LA CINEMATICA EN NUESTRO MODELO

Aplicando el algoritmo de Denavit-Hartenberg ubicamos los sistemas de referencia de todos los grados de libertad que posee el modelo del M5.

En las siguientes imágenes se ven dibujados los modelos que se realizó en el programa Solidedge ST 3





Articulación	Θ	d	a	α
1	q1	L1	0	90°
2	q2	0	L2	0
3	q3	0	L3	0
4	q4+90	0	L4	90°

Armando las matrices y recordando que la forma general de las matrices A es:

$$ {}^{i-1}A_i = \begin{bmatrix} Cos\theta_i & -Cos\alpha_i.Sen\theta_i & Sen\alpha i.Sen\theta i & a_i.Cos\theta_i \\ Sen\theta_i & Cos\alpha_i.Cos\theta_i & -Sen\alpha i.Cos\theta i & a_i.Sen\theta_i \\ 0 & Sen\alpha_i & Cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Aplicando la misma para ${}^0A_1$, ${}^1A_2$, ${}^2A_3$, ${}^3A_4$

$$ {}^0A_1 = \begin{bmatrix} Cos(q_1) & 0 & Sen(q_1) & 0 \\ Sen(q_1) & 0 & -Cos(q_1) & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1A_2 = \begin{bmatrix} Cos(q_2) & -Sen(q_2) & 0 & L_2.Cos(q_2) \\ Sen(q_2) & Cos(q_2) & 0 & L_2.Sen(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^2A_3 = \begin{bmatrix} Cos(q_3) & -Sen(q_3) & 0 & L_3.Cos(q_3) \\ Sen(q_3) & Cos(q_3) & 0 & L_3.Sen(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3A_4 = \begin{bmatrix} Cos(q_4+90) & -Sen(q_4+90) & 0 & L_4.Cos(q_4+90) \\ Sen(q_4+90) & Cos(q_4+90) & 0 & L_4.Sen(q_4+90) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Ahora para calcular la matriz homogénea debemos multiplicar todas la matrices tal que

$$ T = {}^0A_1.\ {}^1A_2.\ {}^2A_3.\ {}^3A_4.\ {}^4A_5 $$

$$ {}^{0}A_2 = {}^{0}A_1 . {}^{1}A_2 = \begin{bmatrix} Cos(q_1) & 0 & Sen(q_1) & 0 \\ Sen(q_1) & 0 & -Cos(q_1) & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} Cos(q_2) & -Sen(q_2) & 0 & L_2.Cos(q_2) \\ Sen(q_2) & Cos(q_2) & 0 & L_2.Sen(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}A_2 = \begin{bmatrix} Cos(q_1).Cos(q_2) & -Cos(q_1)Sen(q_2) & Sen(q_1) & L_2.Cos(q_2).Cos(q_1) \\ Sen(q_1).Cos(q_2) & -Sen(q_1).Sen(q_2) & -Cos(q_1) & L_2.Sen(q_2).Sen(q_1) \\ Sen(q_2) & Cos(q_2) & 0 & L_2.Sen(q_2)+L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}A_3 = \begin{bmatrix} Cos(q_3) & -Sen(q_3) & 0 & L_3.Cos(q_3) \\ Sen(q_3) & Cos(q_3) & 0 & L_3.Sen(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}A_4 = \begin{bmatrix} Cos(q_1).Cos(q_2)Cos(q_4) - Cos(q_1)Sen(q_2)Sen(q_4) & -Cos(q_1)Sen(q_2)Sen(q_4) - Cos(q_1)Sen(q_2)Cos(q_4) & Sen(q_1) & Cos(q_1).Cos(q_2)L_4.Cos(q_4) - Cos(q_1)Sen(q_2)L_4.Sen(q_4) + L_2.Cos(q_2).Cos(q_1) \\ Sen(q_1).Cos(q_2)Cos(q_4) - Sen(q_1).Sen(q_2)Sen(q_4) & -Sen(q_1).Sen(q_2)Sen(q_4) - Sen(q_1).Sen(q_2)Cos(q_4) & -Cos(q_1) & Sen(q_1).Cos(q_2)L_4.Cos(q_4) - Sen(q_1).Sen(q_2)L_4.Sen(q_4) + L_2.Sen(q_2).Sen(q_1) \\ Sen(q_2)Cos(q_4) + Cos(q_2)Sen(q_4) & -Sen(q_4)Sen(q_2) + Cos(q_2)Cos(q_4) & 0 & Sen(q_1)L_4.Cos(q_4) + Cos(q_2)L_4.Sen(q_4) + L_2.Sen(q_2)+L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}A_4 = \begin{bmatrix} Cos(q_4+90) & -Sen(q_4+90) & 0 & L_4.Cos(q_4+90) \\ Sen(q_4+90) & Cos(q_4+90) & 0 & L_4.Sen(q_4+90) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ T = {}^{0}A_3 . {}^{3}A_4 $$

$$ T = \begin{bmatrix} nz & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & 0 & pz \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ nx = \left(Cos(q_1).Cos(q_2)Cos(q_2) - Cos(q_1)Sen(q_2)Sen(q_2)\right).Cos(q_4+90) - \left(Cos(q_1)Sen(q_2)Sen(q_2) - Cos(q_2)Sen(q_2)Cos(q_2)\right).Sen(q_4+90) $$

$$ ny = \left(Sen(q_1).Cos(q_2)Cos(q_2) - Sen(q_1).Sen(q_2)Sen(q_2)\right).Cos(q_4+90) - \left(Sen(q_1).Sen(q_2)Sen(q_2) - Sen(q_1).Sen(q_2)Cos(q_2)\right).Sen(q_4+90) $$

$$ nz = \left(Sen(q_2)Cos(q_2) + Cos(q_2)Sen(q_2)\right).Cos(q_4+90) - \left(Sen(q_2)Sen(q_2) + Cos(q_2)Cos(q_2)\right).Sen(q_4+90) $$

$$ ox = -\left(Cos(q_1).Cos(q_2)Cos(q_2) - Cos(q_1)Sen(q_2)Sen(q_2)\right).Sen(q_4+90) - \left(Cos(q_1)Sen(q_2)Sen(q_2) - Cos(q_1)Sen(q_2)Cos(q_2)\right).Cos(q_4+90) + Sen(q_1) $$

$$ oy = \left(Sen(q_1).Cos(q_2)Cos(q_2) - Sen(q_1).Sen(q_2)Sen(q_2)\right).Sen(q_4+90) - \left(Sen(q_1).Sen(q_2)Sen(q_2) - Sen(q_1).Sen(q_2)Cos(q_2)\right).Cos(q_4+90) - Cos(q_1) $$

$$ oz = \left(Sen(q_2)Cos(q_2) + Cos(q_2)Sen(q_2)\right).Sen(q_4+90) - \left(Sen(q_2)Sen(q_2) + Cos(q_2)Cos(q_2)\right).Cos(q_4+90) $$

$$ ax = 0 \qquad ay = 0 $$

$$px = -\left(Cos(q_1).Cos(q_2)Cos(q_2) - Cos(q_1)Sen(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Cos(q_1)Sen(q_2)Sen(q_2) - Cos(q_1)Sen(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Cos(q_1).Cos(q_2)L_2.Cos(q_2) - Cos(q_1)Sen(q_2)L_2.Sen(q_2) + L_2.Cos(q_2).Cos(q_1)$$

$$py = \left(Sen(q_1).Cos(q_2)Cos(q_2) - Sen(q_1).Sen(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Sen(q_1).Sen(q_2)Sen(q_2) - Sen(q_1).Sen(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Sen(q_1).Cos(q_2)L_2.Cos(q_2) - Sen(q_1).Sen(q_2)L_2.Sen(q_2) + L_2.Sen(q_2).Sen(q_1)$$

$$pz = \left(Sen(q_2)Cos(q_2) + Cos(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Sen(q_2)Sen(q_2) + Cos(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Sen(q_2)L_2.Cos(q_2) + Cos(q_2)L_2.Sen(q_2) + L_2.Sen(q_2) + L_1$$

Ahora hallamos las posiciones respecto de la posición de referencia que en este caso es el origen del sistema de referencia (x_0, y_0, z_0):

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T.\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T.\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$x = -\left(Cos(q_1).Cos(q_2)Cos(q_2) - Cos(q_1)Sen(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Cos(q_1)Sen(q_2)Sen(q_2) - Cos(q_1)Sen(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Cos(q_1).Cos(q_2)L_2.Cos(q_2) - Cos(q_1)Sen(q_2)L_2.Sen(q_2) + L_2.Cos(q_2).Cos(q_1)$$

$$y = \left(Sen(q_1).Cos(q_2)Cos(q_2) - Sen(q_1).Sen(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Sen(q_1).Sen(q_2)Sen(q_2) - Sen(q_1).Sen(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Sen(q_1).Cos(q_2)L_2.Cos(q_2) - Sen(q_1).Sen(q_2)L_2.Sen(q_2) + L_2.Sen(q_2).Sen(q_1)$$

$$z = \left(Sen(q_2)Cos(q_2) + Cos(q_2)Sen(q_2)\right).L_4.Cos(q_4+90) - \left(Sen(q_2)Sen(q_2) + Cos(q_2)Cos(q_2)\right).L_4.Sen(q_4+90) + Sen(q_2)L_2.Cos(q_2) + Cos(q_2)L_2.Sen(q_2) + L_2.Sen(q_2) + L_1$$

2.3. PROGRAMACIÓN CON EL DSP56800/E

Para poder realizar la prueba, generamos un programa en C utilizando el CodeWarrior, el cual nos ofrece una plataforma de trabajo sobre el DSP56800/E. En dicho programa, fuimos variando los parámetros de las articulaciones para generar todos los casos posibles y analizar el comportamiento del robot.

Para simplificar el análisis, decidimos usar para los valores de q1, q2 y q3 el tipo de dato frac16, la cual es la notación utilizada para los puntos flotantes en este tipo de microprocesadores.

Por simplicidad, a continuación transcribimos el código genérico utilizado. Se debe tener en cuenta las restricciones de variabilidad de los parámetros para cada caso.

```
   /**###################################################
   **  Filename  : TP1.C
   **  Project   : TP1
       **  Processor : 56F8367
   **  Version   : Driver 01.13
   **  Compiler  : Metrowerks DSP C Compiler
   **  Date/Time : 24/05/2011, 10:38
   ###################################################*/
   /* MODULE TP1 */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "TFR1.h"
#include "MFR1.h"
#include "MEM1.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 90
#define PULSE2RAD 32767/MXRAD
#define L1 75
#define L2 125
#define L3 126
#define L4 45
Frac16 c,i,;
Frac16 pulse2rad=PULSE2RAD,testResult[MXRAD],testResult2[MXRAD];



/* Ángulo q1 */ Frac16 q1[MXRAD];
/* Ángulo q2*/ Frac16 q2[MXRAD];
/* Ángulo q3*/ Frac16 q3[MXRAD];
/* Ángulo q4*/ Frac16 q4[MXRAD];

/*Matriz Homogenea*/
 Frac16 Total[4][4];
Frac16
x,y,z,aux,aux1,aux2,aux3,aux4;

Word16 c16[MXRAD]; Word32
c32[MXRAD];

void main(void)
{
 int j;

  PE_low_level_init();
 /*** End of Processor Expert internal initialization.    ***/
```

```
  /* Write your code here */

for(i=0;i<MXRAD;i++)
  {
    /*Valorizacion de los argumentos*/
    C32[i]=(L_mult(pulse2rad,i));
    q1[i]=extract_l(c32[i]);
    q2[i]=extract_l(c32[i]);
    q[i]=extract_l(c32[i]);
    q4[i]=extract_l(c32[i]);

 /*Elemento Fila 1 Columna 4 (x)*/
 aux= mult(TFR1_tfr16CosPIx(q2[i]), TFR1_tfr16CosPIx(q3[i]));
 aux2=mult(TFR1_tfr16SenPIx(q2[i]), TFR1_tfr16SenPIx(q3[i]));
 aux1=add(aux,negate(aux2));
  /*1° termino*/
 aux3=mult(aux1, L4*TFR1_tfr16SenPIx(q4[i]));
 /*2° termino*/
  aux4=mult(add(aux2,negate(mult(TFR1_tfr16SenPIx(q2[i]),
TFR1_tfr16CosPIx(q3[i])))),L4*TFR1_tfr16CosPIx(q4[i]));
 aux5=add(aux3,add(aux4,aux1*L3));
 Total[0][3]=mult(TFR1_tfr16CosPIx(q1[i]),add(aux5,L2* TFR1_tfr16CosPIx(q2[i])));


 /*Elemento Fila 2 Columna 4 (y) */
 aux5=add(negate(aux3),add(aux4,aux1*L3));
 Total[1][3]=mult(TFR1_tfr16SenPIx(q1[i]),add(aux5,L2* TFR1_tfr16SenPIx(q2[i])));

 /*Elemento Fila 3 Columna 4 (z)*/
 aux1=add(aux,aux2);
 /*2° termino*/
 aux3=mult(aux1, (L4*TFR1_tfr16CosPIx(q4[i])));
 aux= mult(TFR1_tfr16CosPIx(q2[i]), TFR1_tfr16SenPIx(q3[i]));
 aux2=mult(TFR1_tfr16SenPIx(q2[i]), TFR1_tfr16CosPIx(q3[i]));
 aux1=add(aux,(aux2));
 /*1° termino*/
 aux4=mult(aux1,(L4* TFR1_tfr16SenPIx(q4[i])));
 aux5= add(negate(aux4),negate(aux3));
 Total[2][3]=add(aux4,add((aux1*L3),add((TFR1_tfr16SenPIx(q2[i
 ])*L2),L1)));

 /* Calculamos las coordenadas cartesianas del punto extremo del robot ( X Y
 Z)*/

 x=Total[0][3];
 y=Total[1][3];
 z=Total[2][3];

 /*Imprimo X,Y,Z  y los argumentos q1,q2,l3*/
 printf ("%d   %d   %d\n",x,y,z);
 Print("Q1= %d\t Q2=%d\t Q3= %d\t Q4=%d\n"q1[i],q2[i],q3[i],q4[i]);
```

```
        }
    }

    /* END TPN1 */
```

2.4. GRÁFICOS EN MATLAB

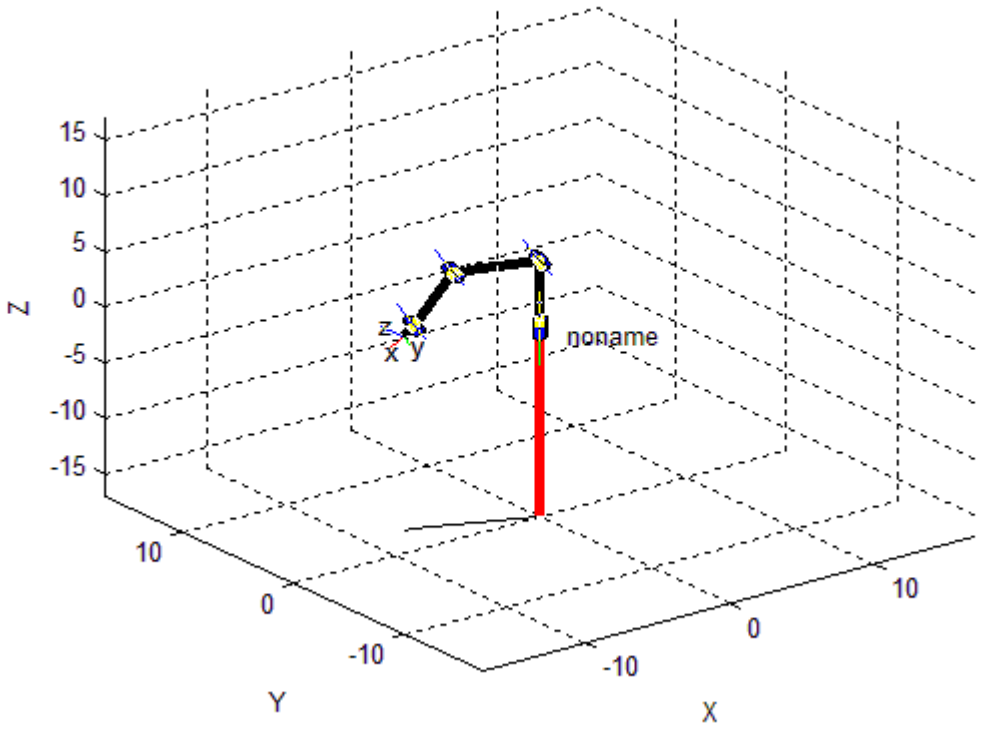
En primera instancia con los toolbox de robótica de Matlab chequeamos que la matriz este correcta .

Escribiendo el siguiente código:

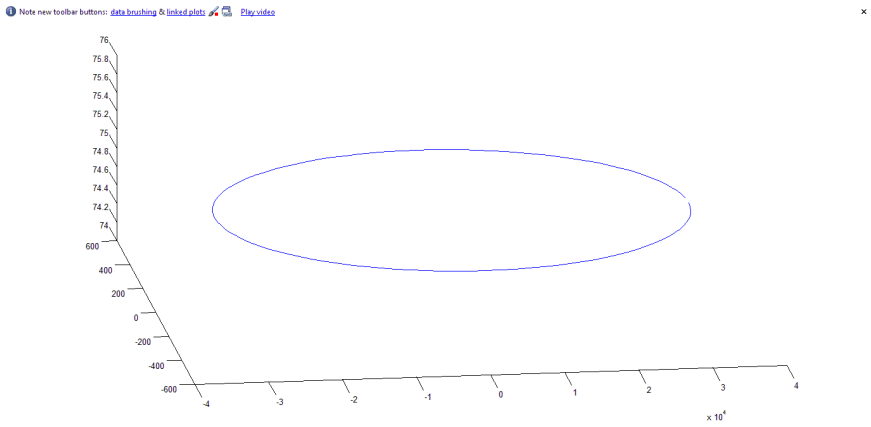
```
O1=0;
O2=0;
O3=0;
O4=0;
L1=link([(pi/2) 0 O1 6 0], 'standard')
L2=link([0 5 O2 0 0], 'standard')
L3=link([0 5 O3 0 0], 'standard')
L4=link([(pi/2) 1 (O4+pi/2) 0 0], 'standard')
r=robot({L1 L2 L3 L4})
syms q1 q2 q3 q4
plot(r,[0 0 0 0])
drivebot(r)
```

Tomando los valores obtenidos por el dsp graficamos los puntos x,y,z

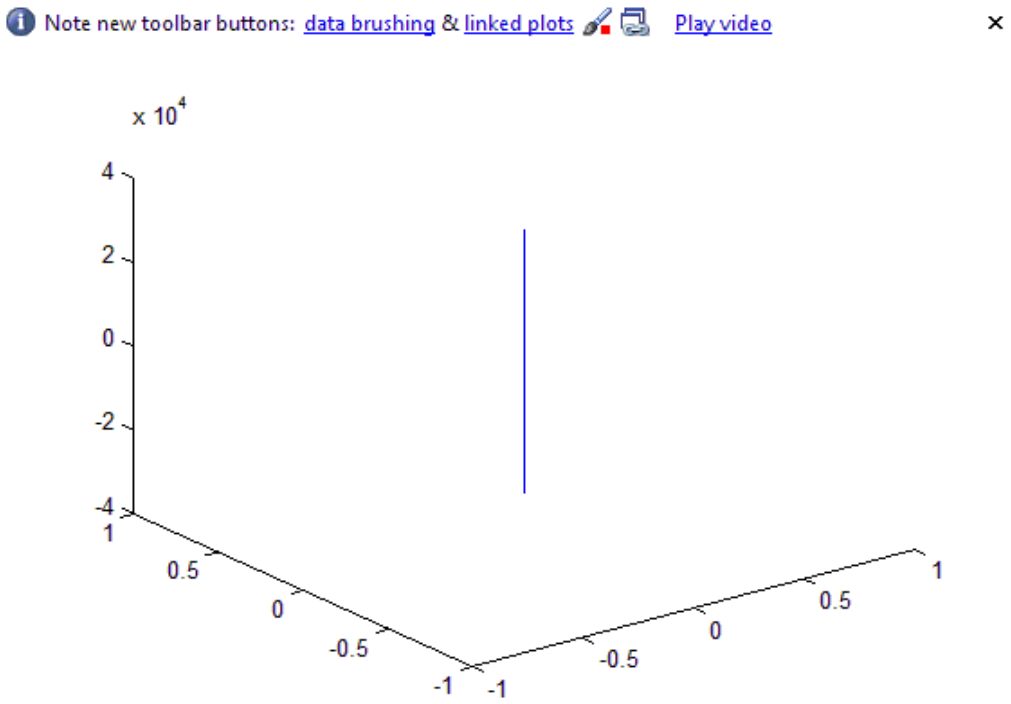
Se visualiza la modelización del robot.



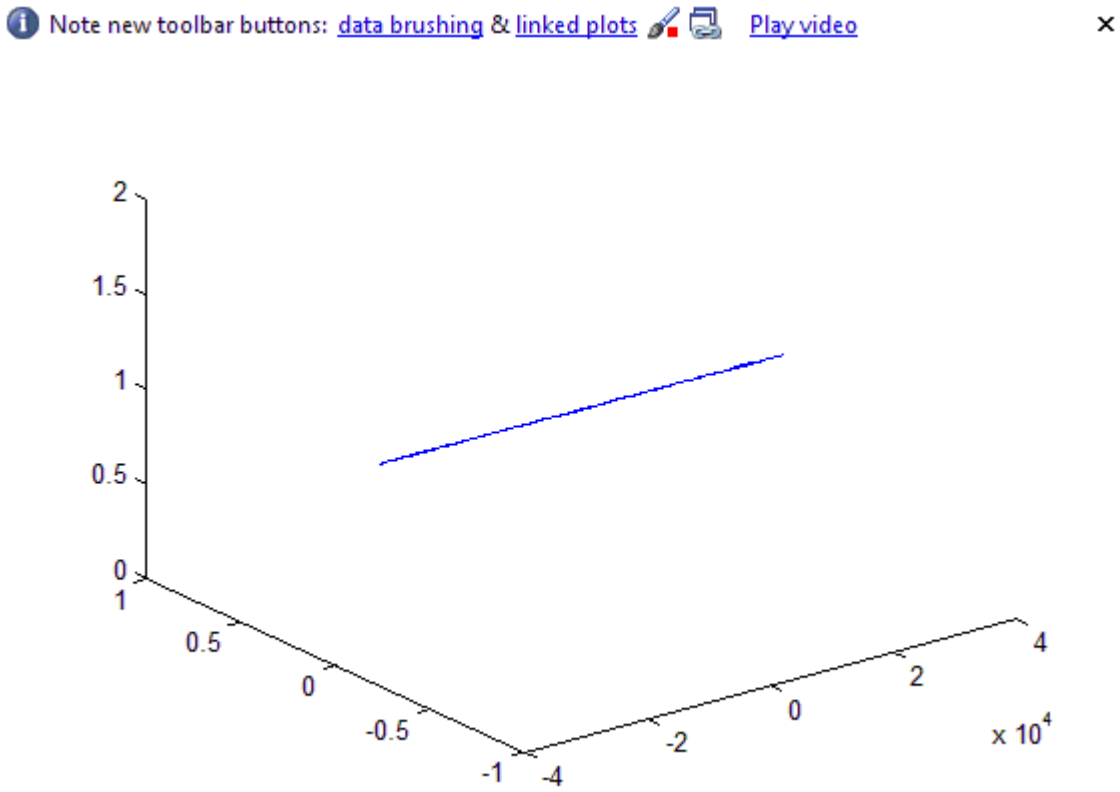
Haciendo variar Q1 se tiene



Haciendo variar Q2 solamente se tiene:



Haciendo variar Q4



Ahora si hacemos mover todas las articulaciones.

La utilización del CodeWarrior fue una herramienta excelente, ya que nos permite desarrollar y simular virtualmente nuestros programas sobre un DSP. Una de las ventajas también importantes es que viene con un set de funciones ya armadas para la utilización en nuestros programas, lo cual nos permite enfocarnos directamente sobre el problema.

También la utilización del Matlab con el toolbox de robótica para verificar y graficar los resultados, lo cual nos permitió de una forma fácil, observar cómo se iba desplazando el brazo del robot, de acuerdo a los parámetros que iban variando.

Es importante destacar que cuando resolvemos los ejercicios matemáticamente, no tenemos en cuenta los límites físicos que tiene el robot. Gracias a que pudimos graficar el desplazamiento, pudimos ver y tomar precauciones en el rango de variación de los parámetros.

Por último, de acuerdo a todos los gráficos anteriores, podemos observar que no es posible generar un movimiento en línea recta variando los 3 parámetros al mismo tiempo. Para lograr esto, se debería utilizar la teoría de la cinemática inversa, y calcular para una trayectoria dada, los parámetros necesarios de las articulaciones.

3. DINÁMICA DEL ROBOT

Ahora se hará del uso de las herramientas de análisis dinámico que tiene la robótica para analizar un diseño de un robot, en el caso particular será el M5 de SkyMec.

El trabajo se divide en 3 partes. La primera es una introducción teórica a las herramientas de análisis dinámico, la segunda es un ejemplo de análisis dinámico resuelto en Matlab con las herramientas del toolbox Corke y la tercera es una implementación del control de motores BLDC hecha en VHDL. Este último punto esta resuelto con herramientas de código abierto y se entrega en este mismo trabajo un pequeño manual dónde se explica como realizar las simulaciones.

3.1. HERRAMIENTAS DE ANÁLISIS DINÁMICO

La dinámica es la rama de la física que estudia los movimientos y las causas que los producen. 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. En nuestro caso un robot de 5 GDL.

El objetivo último del análisis dinámico será la generación de las señales de control para los motores que gobiernen nuestro robot.

Estas ecuaciones relacionan 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 .

Como es de suponer, la complejidad para obtener estas relaciones aumenta conforme aumenta la complejidad del robot. Tanto es así que a veces no se puede obtener una ecuación diferencial de segundo orden que resuelva nuestro modelo sino que se deben apelar al cálculo numérico iterativo para resolverlo.

3.2. PARÁMETROS IMPORTANTES EN EL ANÁLISIS DINÁMICO

- **Centro de Masa**

El centro de masa se calcula como:

$$\mathbf{r}_{\mathrm{cm}} = \frac{\int \mathbf{r}\ dm}{\int dm} = \frac{1}{M}\int \mathbf{r}\ dm$$

y es el punto donde la energía potencial se puede considerar cómo:

$$K = \frac{1}{2}mv^2 + K_{rot}$$

Siendo "m" la masa total del cuerpo, "v" la velocidad de traslación del centro de masas y Krot la energía de rotación del cuerpo, expresable en términos de la velocidad angular y el tensor de inercia

- **Momento Angular**

El momento angular es una magnitud física importante porque en muchos sistemas físicos constituye una magnitud conservada, a la cual bajo ciertas condiciones sobre las fuerzas es posible asociarle una ley de conservación.

Se define como:

$$\mathbf{L}_O = \int_V \rho(\mathbf{r}_O \times \mathbf{v}_O) \quad dV$$

- **Momento de Inercia**

Es una medida de la inercia rotacional. Es análogo a la masa de un cuerpo en un sistema traslacional y permite relacionar el momento aplicado a dicho cuerpo con la aceleración angular que va a desarrollar.

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} = \mathbf{r} \times m\mathbf{v}$$

- **Tensor de Inercia**

Es una entidad que caracteriza la inercia rotacional de un sólido rígido. A partir del tensor de inercia, cuyas componentes se calculan:

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

Se puede calcular la energía de rotación:

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

- **Fuerza Centrípeta**

La fuerza centrípeta aparece para producir una aceleración a un cuerpo que está girando. Su expresión es la siguiente:

$$\mathbf{a} = -\frac{v^2}{r} \left(\frac{\mathbf{r}}{r} \right) = -\frac{v^2}{r} \hat{\mathbf{u}}_r = -\omega^2 \mathbf{r}$$

- **Fuerza de Coriolis**

La fuerza de Coriolis es una fuerza ficticia que aparece cuando un cuerpo está en movimiento con respecto a un sistema en rotación y se describe su movimiento en ese referencial. La fuerza de Coriolis es diferente de la fuerza centrífuga. La fuerza de Coriolis siempre es perpendicular a la dirección del eje de rotación del sistema y a la dirección del movimiento del cuerpo vista desde el sistema en rotación.

- **Ecuación del Torque**

  Esta ecuación es la más importante, y es la que permite relacionar todo lo anterior.

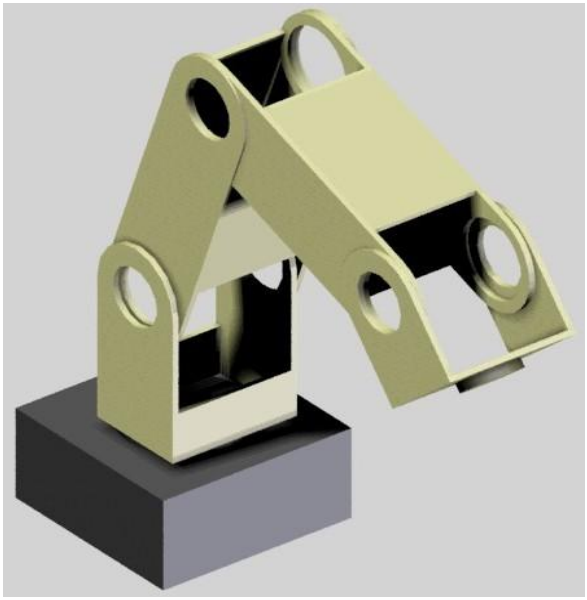
$$\tau = D\ddot{q} + H + C$$

Y su resolución la haremos a través del toolbox Corke que calcula los valores a través de un método llamado Recursive Newton-Euler.

3.3. ANÁLISIS DINÁMICO DEL ROBOT M5

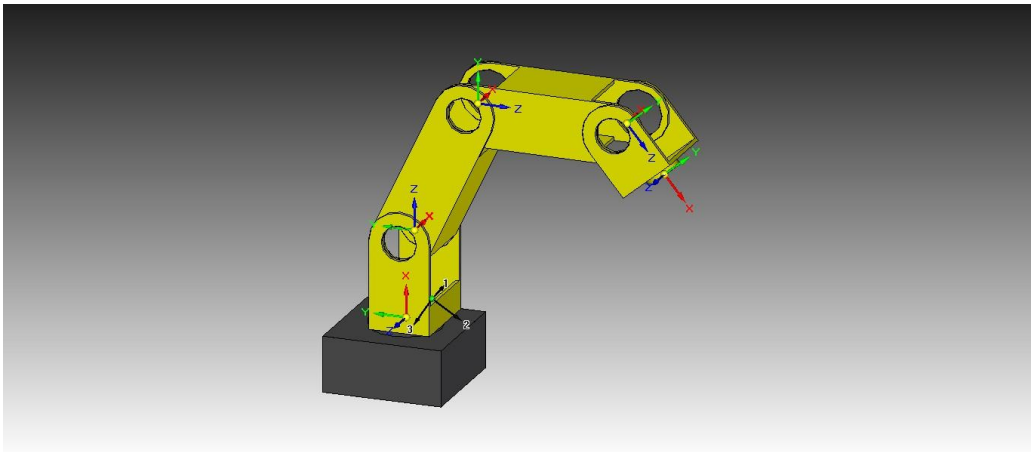
Mecánica de los eslabones

El análisis en el toolbox Corke se hace a partir de conocer la mecánica de los eslabones del robot. Para esto dibujamos el robot en el Solid Works, asignamos materiales a las distintas piezas y obtuvimos los datos de cada una de ellas.



Como se observa, el dibujo no está hecho al detalle, pues es un diseño complejo y quisimos centrar el trabajo en en análisis y no en aprender a utilizar la herramienta de diseño que es un desafío en si misma.

En la próxima imagen se ven los ejes coordenados que se tomaron como referencia:



Estos fueron los parámetros obtenidos para cada eslabón, tal como el software CAD los detalla:

Eslabón 1:

Physical Properties Report

mass= 813,307 g

With respect to the Articulacion1 Coordinate System.

Center of Mass:

X= 1,37 mm

Y= -14,40 mm

Z= -56,30 mm

Mass Moments of Inertia:

Ixx= 110161,77 g-cm^2

Iyy= 86104,66 g-cm^2

Izz= 36543,45 g-cm^2

Ixy= 405,76 g-cm^2

Ixz= -1776,14 g-cm^2

Iyz= -22668,19 g-cm^2

Eslabón 2:

Physical Properties Report

mass= 813,307 g

With respect to the Articulacion2 Coordinate System.

Center of Mass:

X= 1,37 mm

Y= -169,58 mm

Z= -37,63 mm

Mass Moments of Inertia:

Ixx= 328091,03 g-cm^2

Iyy= 45835,79 g-cm^2

Izz= 294741,57 g-cm^2

Ixy= -3045,34 g-cm^2

Ixz= -975,78 g-cm^2

Iyz= 80927,40 g-cm^2

Eslabón 3:

Physical Properties Report

mass= 813,307 g

With respect to the Articulacion 3 Coordinate System.

Center of Mass:

X= 1,37 mm

Y= -234,35 mm

Z= 24,72 mm

Mass Moments of Inertia:

Ixx= 534345,19 g-cm^2

Iyy= 26284,94 g-cm^2

Izz= 520546,59 g-cm^2

Ixy= -3782,04 g-cm^2

Ixz= 801,91 g-cm^2

Iyz= -65230,63 g-cm^2

Eslabón 4:

Physical Properties Report

mass= 813,307 g

With respect to the Articulacion4 Coordinate System.

Center of Mass:

X= -25,28 mm

Y= -234,35 mm

Z= -1,37 mm

Mass Moments of Inertia:

Ixx= 520546,50 g-cm^2

Iyy= 26512,06 g-cm^2

Izz= 534572,40 g-cm^2

Ixy= 30069,82 g-cm^2

Ixz= -244,85 g-cm^2

Iyz= 3773,94 g-cm^2

3.4. ANÁLISIS CON MATLAB

Para la simulación en Matlab se ha utilizado el toolbox Corke. Aquí se trata de obtener las ecuaciones de ***cupla, torque o par*** necesarios para los cuatro motores brushless que operan las cuatro articulaciones de nuestro robot M5 (por considerar solo cuatro grados de libertad) a fin de saber que cupla debo aplicarle a cada motor para obtener una respuesta satisfactoria al requerimiento solicitado a nuestro robot M5.

Como resultado obtuvimos los siguientes pares motores. O sea, que de acuerdo a los valores de **posición, velocidad y aceleración** que necesitemos obtener del robot, podemos calcular los pares motores que se deberán aplicar a las articulaciones del robot.

A continuación listamos el código de Matlab que utilizamos:

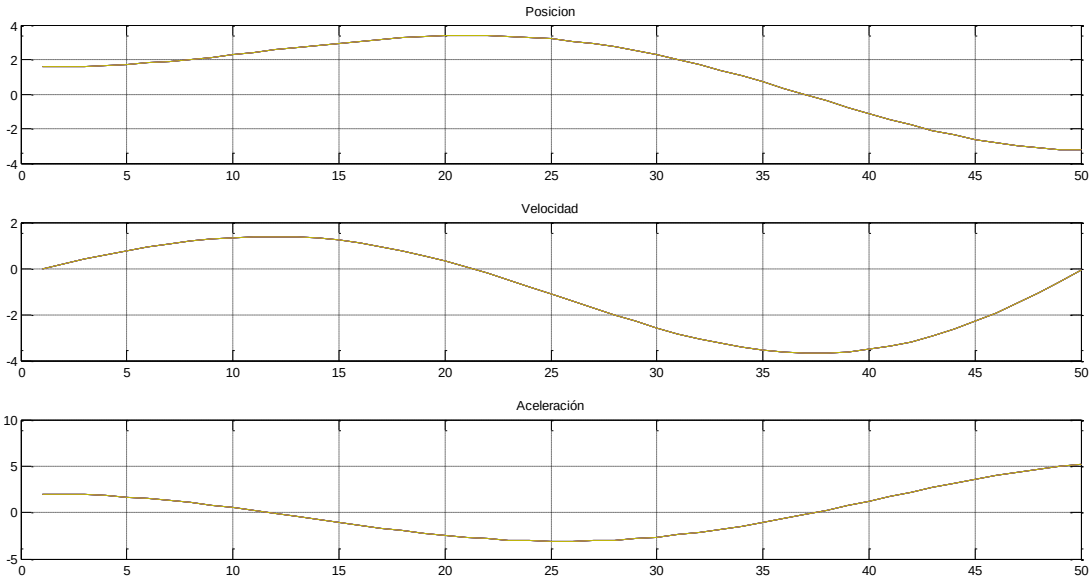
```
% Robot5_6 define los parametros del robot en cuestión.

clear ALL;
clc;
syms t

% ================================================================
%                          Datos del Robot
% ================================================================

%Datos de las longitudes de los eslabones [m]

L1=75/1000;
L2=125/1000;
L3=126/1000;
L4=45/1000;

% Parametros DH.
% L =LINK([alpha A theta D sigma])

L{1}=link([pi/2 0 L1 0 0],'standard');
L{2}=link([0 L2 0 0 0],'standard');
L{3}=link([0 L3 0 0 0],'standard');
L{4}=link([pi/2 L4 pi/2 0 0],'standard');

L{1}.m = 813.3/1000;
L{2}.m = 813.3/1000;
L{3}.m = 813.3/1000;
L{4}.m = 813.3/1000;

% Coordenadas de los centros de masa de cada eslabon [m].

L{1}.r = [1.37e-3 -14.4e-3 -56.3e-3];
L{2}.r = [1.37e-3 -169.6e-3 -37.6e-3];
L{3}.r = [1.37e-3 -234.35e-3 24.72e-3];
L{4}.r = [-25.3e-3 -234.35e-3 -1.4e-3];
```

```
% Tensores de inercia (Ixx, Iyy, Izz, Ixy, Iyz, Ixz) [Kg x (m)^2].

L{1}.I = [0 0 0 0 0 0];
L{2}.I = [0 0 0 0 0 0];
L{3}.I = [0 0 0 0 0 0];
L{4}.I = [0 0 0 0.3 0 0];

% Inercia del motor que acciona al elemento.

L{1}.Jm = 0;
L{2}.Jm = 0;
L{3}.Jm = 0;
L{4}.Jm = 0;

% Coeficiente de reduccion del actuador del elemento.

L{1}.G = 1;
L{2}.G = 1;
L{3}.G = 1;
L{4}.G = 1;

% Rozamiento viscoso del actuador del elemento.

L{1}.B = 0;
L{2}.B = 0;
L{3}.B = 0;
L{4}.B = 0;

% Rozamiento de Coulomb en la direccion positiva y negativa del actuador
% del elemento.

L{1}.Tc = [ 0 0 ];
L{2}.Tc = [ 0 0 ];
L{3}.Tc = [ 0 0 ];
L{4}.Tc = [ 0 0 ];

% Definicion del robot.
% ROBOT Robot object constructor.
% function r = robot(L, a1, a2, a3).

Robot5_6 = robot(L, 'ROBOTM5', 'SkyMec', 'Robot 5 GDL');

% Dibuja torques y fuerzas para una determinada trayectoria utilizando la
% funcion "rne".

% Se define la gravedad con respecto al sistema {S0}.

grav = [ 0 0 -9.8];

% ============================================================
%                Determina la trayectoria
% ============================================================
```

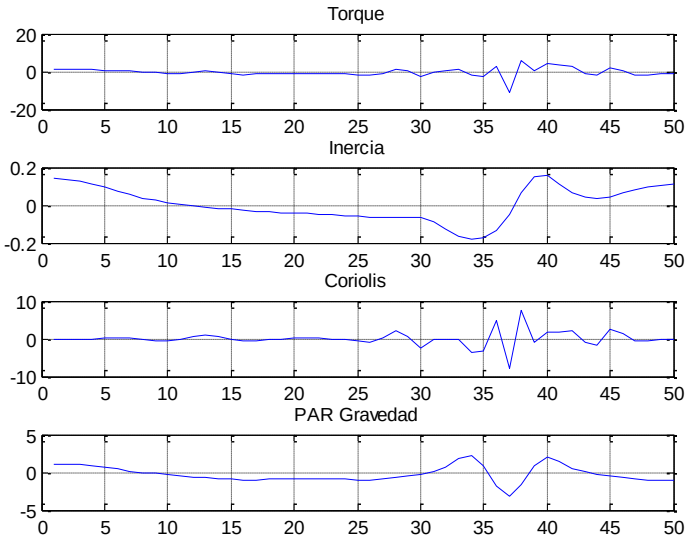
```
% Definicion simbolica de la posicion.

for i=1:6
     sq(i,1)= sin(t) * t + pi/2;
end

% Obtención simbólica de la velocidad y aceleración.

for i=1:6
    sqd(i,1) = diff(sq(i,1),'t');
    sqdd(i,1) = diff(sqd(i,1), 't');
end


% ============================================================
% Evaluacion numerica de posicion, velocidad, aceleracion y XY.
% ============================================================

for tk = 1:1:50

t = (tk - 1)/10;

    for i=1:4

    q(tk,i) = eval(sq(i,1));
    qd(tk,i) = eval(sqd(i,1));
    qdd(tk,i) = eval(sqdd(i,1));
    XY(tk,i) = q(tk,i) * cos(q(tk,i));

    end

end

% ============================================================
%                        Dibuja la trayectoria.
% ============================================================

figure(7);

subplot(3,1,1);
plot(q);
title('Posicion');
grid;
subplot(3,1,2);
plot(qd);
title('Velocidad');
subplot(3,1,3);
grid;
plot(qdd);
title('Aceleración');
grid;
```

```
% ================================================================
%              Calcula matriz gravedad, coriolis, torque
% ================================================================

Tau = rne(Robot5_6, q, qd, qdd, grav);
TauG = GRAVLOAD(Robot5_6, q, grav);
TauC = CORIOLIS(Robot5_6, q, qd);
TauI = ITORQUE(Robot5_6, q, qdd);

% ================================================================
%              Grafica gravedad, coriolis, torque
% ================================================================for i=1:2:5
for i=1:2:5
    figure(i)
    subplot(4,1,1);
    plot(Tau(:,i));
    title('Torque');
    subplot(4,1,2);
    plot(TauI(:,i));
    title('Inercia');
    subplot(4,1,3);
    plot(TauC(:,i));
    title('Coriolis');
    subplot(4,1,4);
    plot(TauG(:,i));
    title('PAR Gravedad');
end
```

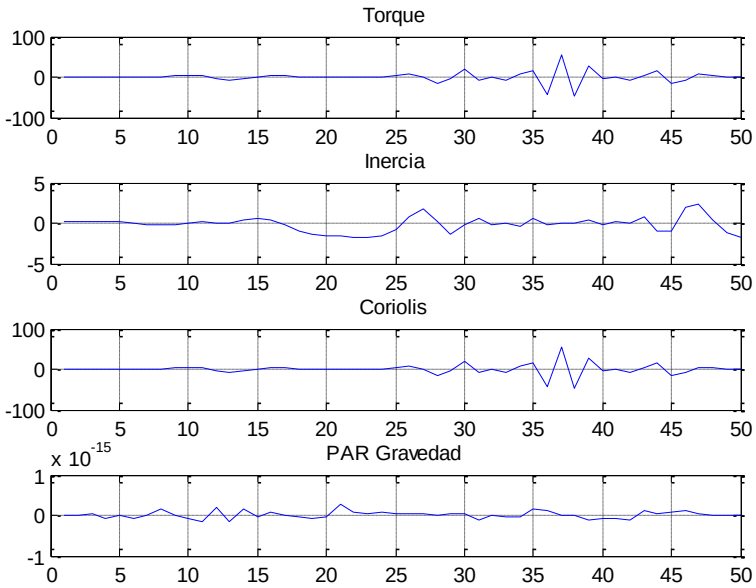
A continuación se muestran los gráficos que obtuvimos:



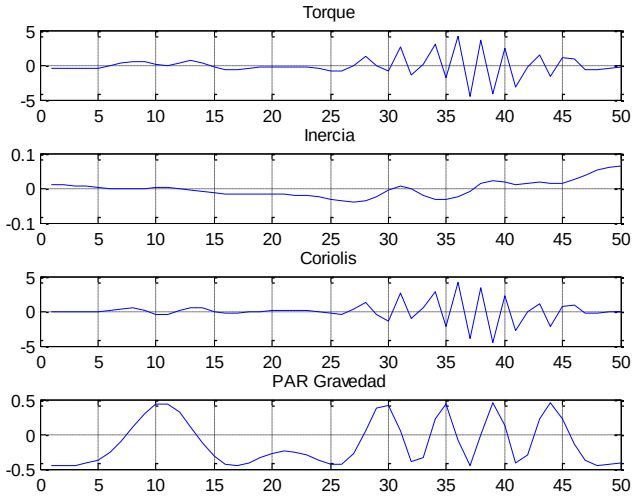
Trayectoria que supusimos para todos los eslabones.



Torque para la base del robot



Torque para el primer eslabon



Torque para el tercer eslabón

Los demás eslabones no los simulamos porque supusimos que los torques más interesantes eran los de los primros eslabones, debido a que son los que soportan la mayor carga.

4. GHDL

4.1. INTRODUCCIÓN A GHDL

GHDL es un compilador del lenguaje VHDL. Toda la información encontrada en este manual fue tomada de http://ghdl.free.fr el sitio oficial del proyecto.

4.2. INSTALACIÓN

Para instalar ghdl en ubuntu ingresamos en una consola:

```
sudo apt-get install ghdl
```

también utilizaremos el GTKWave para visuarlizar las formas de onda generadas

```
sudo apt-get install gtkwave
```

Se recomienda crear una estructura de archivos como la siguiente:

/directorio_de_trabajo

/directorio_de_trabajo/work: directorio para los archivos de trabajo del ghdl, aquí guarda los productos intermedios de la compilación.

/directorio_de_trabajo/src: archivos fuente de nuestro diseño.

/directorio_de_trabajo/tbench: archivos de testbench para nuestro diseño.

4.3. COMANDOS DE GHDL

La sintáxis para invocar comandos de ghdl es la siguiente:

```
ghdl -comando [--opciones]
```

4.4. ANÁLISIS

```
ghdl -a [opciones] archivos
```

Compila los archivos indicados y genera un archivo de código objeto por cada uno de ellos.

4.5. ELABORACIÓN

```
ghdl -e [opciones ] unidad_principal [unidad_secundaria ]
```

Este comando crea un archivo ejecutable (sólo en Linux) conteniendo el código VHDL de los archivos fuente y los archivos de simulación.

Se puede especificar el nombre de la librería de trabajo con la opción

```
--work=NOMBRE
```

y se puede definir el directorio de trabajo con la opción

```
--workdir=DIRECTORIO
```

4.6. CHEQUEAR LA SINTAXIS

```
ghdl -a [opciones] archivos
```

Chequea la sintaxis de un determinado archivo, sin generar código objeto.

4.7. EJECUTAR

```
ghdl -r unidad_principal [opciones_de_simulacion]
```

o simplemente

```
./unidad_principal
```

La opción –-vcd =archivo genera un archivo de salida con las formas de onda.

4.8. IMPORTAR

```
ghdl -i [opciones] archivos
```

Agrega archivos a la librería de trabajo. En un proyecto con múltiples archivos (fuentes, simulación, funciones) es conveniente utilizar esta opción.

4.9. MAKE

```
ghdl -m [opciones] unidad_principal
```

Analiza automáticamente los archivos desactualizados y elabora un diseño.

Los archivos tiene que haber sido previamente incluidos en la librería de trabajo, ya sea con la función Importar o Analizar.

Ejemplo 1: Hola Mundo

El siguiente ejemplo tiene el objetivo de demostrar el uso de los comandos vistos recientemente. (El programa fue tomado del manual de GHDL, que puede encontrarse en el sitio del proyecto: http://ghdl.free.fr

Como primer paso crearemos la estructura de directorios para trabajar, elegimos el directorio donde queremos guardar el proyecto y creamos la siguiente estructura:

```
/helloworld/src
/helloworld/work
/helloworld/tbench
```

Este último no lo vamos a utilizar en este ejemplo, pero lo creamos igualmente.

Copiamos el siguiente código en un editor de texto y lo guardamos en */helloworld/src* con el nombre *hello.vhdl*

```
-- Hello world program.

-- Defines a design entity, without any ports.
use std.textio.all; --Imports the standard textio package.

entity hello_world is
end hello_world;

architecture behaviour of hello_world is
begin
     process
          variable l : line;
     begin
          write (l, String'("Hello world!"));
          writeline (output, l);
          wait;
     end process;
end behaviour;
```

- **Análisis**

Para analizar el código nos paramos en el directorio /helloworld y en una terminal ingresamos:

```
ghdl -a --workdir=work  ./src/*.vhdl
```

Si todo salió correctamente no deberíamos obtener ninguna respuesta. Sino probablemente haya algún error en el código del archivo o en el comando ingresado.

Entonces si listamos los archivos en /helloworld/work vemos los archivos:

```
hello.o
work-obj93.cf
```

Donde *hello.o* es el código objeto y el archivo *work-obj93.cf* es el archivo de trabajo para éste proyecto. El contenido de este último archivo no es objeto de análisis de este trabajo.

- **Elaboración**

Luego creamos el archivo ejecutable:

```
ghdl -e --workdir=work hello_world
```

Aparece en el directorio principal del proyecto un archivo llamado hello_wolrd, que es ejecutable.

Notamos que el objeto de la compilación se llama **hello_world** en vez de **hello.o** pues el parámetro que recibe el comando ghdl -e es el nombre de la entidad que está compilando, **no del archivo.**

- **Ejecución**

Para ejecutar el ejemplo tipeamos:

```
ghdl -r hello_world
```

o simplemente

```
./hello_world
```

Ejemplo 2: Atmel_PWM

En este ejemplo tendremos la oportunidad de ver el uso de GHDL para un proyecto más grande que el anterior, con varios archivos y visualización de de señales.

Los archivos *vhdl* con los que vamos a trabajar se encuentran en el campus virtual de la materia y son los archivos:

pwm_fpga. vhd

user_pkg_inc_dec. vhd

pwm_pretb. vhd

Dejamos afuera el archivo pwm_posttb.vhd porque en nuestro caso no estamos sintetizando el diseño, sólo simulándolo, de forma que no tiene sentido hacer este análisis.

- **Estructura de trabajo**

El directorio de trabajo sera *pwmrobot.* Alli creamos los directorios *src, tbench y work*.

En el directorio *src* copiamos los archivos *pwm_fpga.vhd* y *user_pkg_inc_dec.vhd.*

En el directorio tbench copiamos el archivo *pwm_pretb.vhd.*

- **Creación de la librería**

En este ejemplo podemos ver que hay una dependencia jerárquica: el módulo de testbench usa una instancia de la entidad pwm_fpga que ni siquiera se encuentra en el mismo archivo.

Para salvar los conflictos que pudiera generar esta jerarquía vamos a *importar* los archivos a una librería de trabajo con el siguiente comando:

```
ghdl -i --workdir=work ./src/*.vhd ./tbench/*.vhd
```

Si listamos el contenido del directorio work, entonces encontraremos el archivo work-obj93.cf pero ningun archivo objeto, pues sólo definimos los archivos con los que vamos a trabajar, pero no los compilamos todavía.

- **Compilación**

Ahora vamos a compilar los archivos y crear un ejecutable con el comando *make*.

La entidad que queremos elaborar (es decir, el nombre de la entidad que define el archivo de testbench) se llama pwm_fpga_test_bench y este será el objeto de nuestra compilación.

El comando es:

```
ghdl -m --ieee=synopsys --workdir=work pwm_fpga_test_bench
```

Dónde la opción --ieee=synopsys indica al GHDL que incluya algunas funciones no estandar de la librería ieee como *'std_logic_arith', 'std_logic_signed', 'std_logic_unsigned', 'std_logic_textio'* [según el manual de GHDL].

Ahora si listamos los archivos veremos que en el directorio principal (pwmrobot) se creó el archivo ejecutable *pwm_fpga_test_bench* y en el directorio */pwmrobot/work* se crearon los archivos objeto (*.o*))para cáda archivo fuente.

- **Ejecución**

Si analizamos el archivo de testbench, podemos notar que el tiempo de ejecución es de 150,1 μs:

```
    sig_reset <= '1';
    wait for 100 ns;
    sig_reset <= '0';
    sig_data_value <= "11000000";
    wait for 50 us;
    sig_data_value <= "10000000";
    wait for 50 us;
    sig_data_value <= "01000000";
    wait for 50 us;
```

Sin embargo, la simulación no es capaz de interpretar esto para finalizar por si misma, sino que tendremos que indicarle el tiempo por el que se debe extender. Esto se hace con la opción *--stop-time=TIEMPO*

Entonces corremos la simulación con:

```
./pwm_fpga_test_bench --stop-time=160us --vcd=signals.vcd
```

La simulación termina con un mensaje:

```
./pwm_fpga_test_bench:info: simulation stopped by --stop-time
```

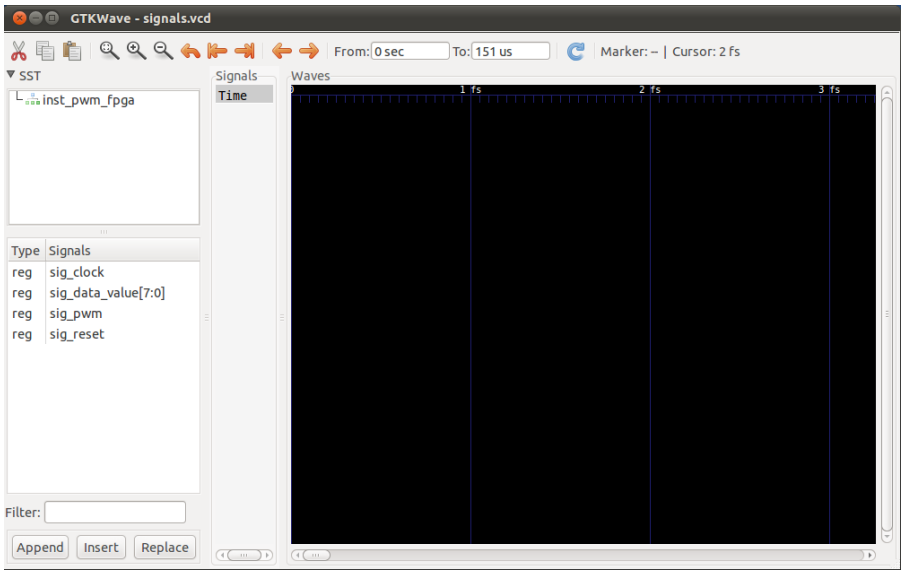
y en el directorio de trabajo se crea el archivo *signals.vcd*.

- **Visualización de la señales**

Para visualizar las señales generadas por la simulación corremos el programa GTKWave

```
gtkwave signals.vcd
```

Vemos una pantalla como la siguiente:

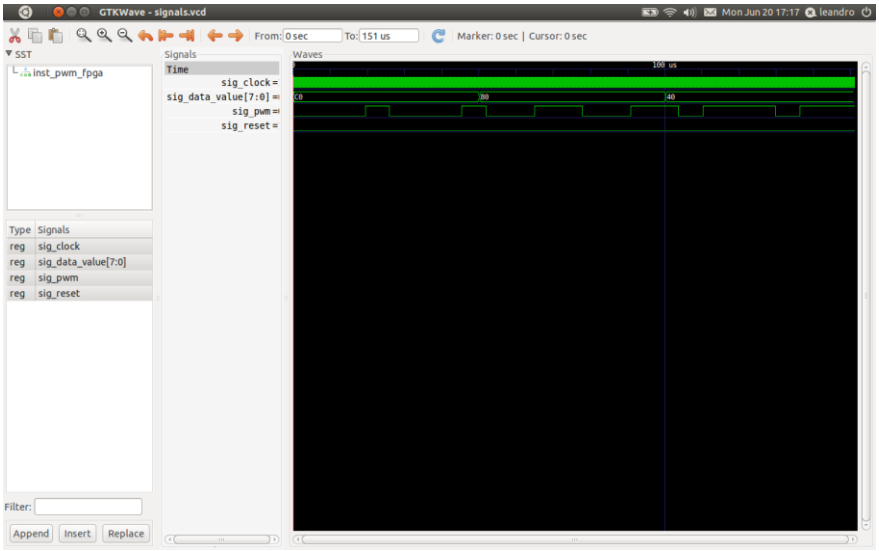


Ninguna señal se muestra, pues tenemos que agregarlas con el botón append que aparece abajo a la izquierda.

Una vez seleccionadas podemos hacer click en el botón de escala “FIT” para visualizar las señales a lo largo de toda la pantalla.

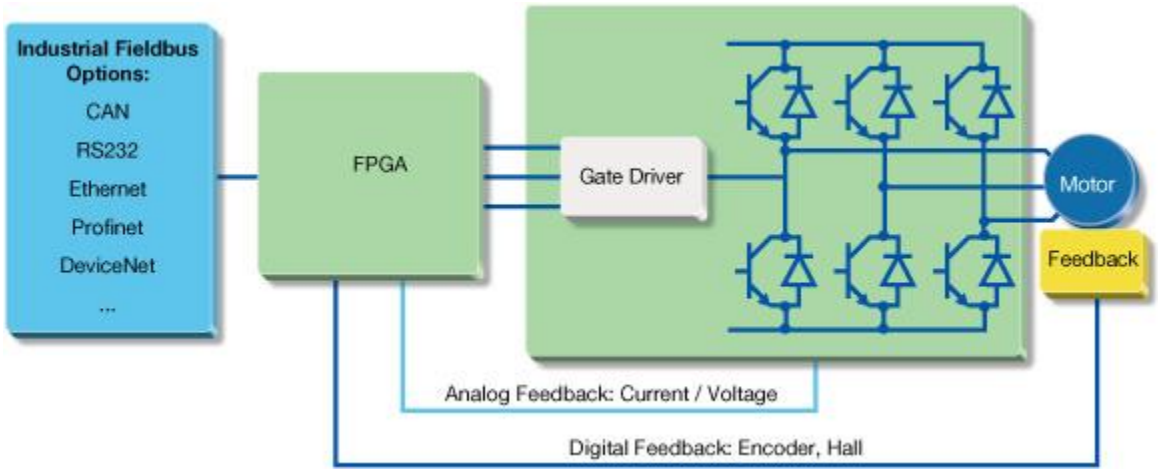
No hacemos una descripción del GTKWave pues su uso es bastante intuitivo y escapa al objetivo de este manual.

Una vez que hayamos agregado las señales y acomodado la escala, podemos analizar el funcionamiento de nuestro modelo:



Ejemplo 3: BLDC Driver

A continuación se adjunta el código del driver para motores brushless y la arquitectura que implementamos fue la siguiente:



Tal como se supone nuestra implementación se centra en el programa de la FPGA, excluyendo el módulo que se indica como Gate Driver y el módulo de comunicaciones.

Para realizar esta función, tomamos el módulo generador de PWM del ejemplo anterior y agregamos la realimentación de los sensores de efecto hall, de forma de generar la secuencia que se muestra a continuación:

Hall Sensor			Outputs					
A	B	C	S1	S2	S3	S4	S5	S6
1	0	0	1	0	0	PWM	0	0
1	1	0	1	0	0	0	0	PWM
0	1	0	0	0	1	0	0	PWM
0	1	1	0	PWM	1	0	0	0
0	0	1	0	PWM	0	0	1	0
1	0	1	0	0	0	PWM	1	0

Esto lo logramos con el siguente bloque:

/src/driver_mosfet.vhdl

```
library IEEE;
```

```
USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all ;

USE   work.user_pkg.all;


ENTITY driver_mosfet IS

PORT (  s1 :out STD_LOGIC;

            s2 :out STD_LOGIC;

            s3 :out STD_LOGIC;

            s4 :out STD_LOGIC;

            s5 :out STD_LOGIC;

            s6 :out STD_LOGIC;

            hall_sensor :in  std_logic_vector(2 downto 0);

            reset :in STD_LOGIC;

            clock :in STD_LOGIC;

            pwm_in :in STD_LOGIC

     );


END driver_mosfet;


ARCHITECTURE arch_driver OF driver_mosfet IS


-- Internal signal declaration


BEGIN


Cntrl_PWM_puenteH: PROCESS (reset, hall_sensor, pwm_in) IS


BEGIN

      IF (reset = '1') THEN
```

```
            s1<= '0'; s2<= '0'; s3<= '0'; s4<= '0'; s5<= '0'; s6<= '0';
      ELSE
            CASE hall_sensor IS
                  WHEN "100" =>                 --Estado 1
                        s1<= '1'; s2<= '0'; s3<= '0'; s4<= pwm_in; s5<= '0'; s6<= '0';

                  WHEN "110" =>                 --Estado 2
                        s1<= '1'; s2<= '0'; s3<= '0'; s4<= '0'; s5<= '0'; s6<= pwm_in;

                  WHEN "010" =>                 --Estado 3
                        s1<= '0'; s2<= '0'; S3<= '1'; s4<= '0'; s5<= '0'; s6<= pwm_in;

                  WHEN "011" =>                 --Estado 4
                        s1<= '0'; s2<= pwm_in; s3<= '1'; s4<= '0'; s5<= '0'; s6<= '0';

                  WHEN "001" =>                 --Estado 5
                        s1<= '0'; s2<= pwm_in; s3<= '0'; s4<= '0'; s5<= '1'; s6<= '0';

                  WHEN "101" =>                 --Estado 6
                        s1<= '0'; s2<= '0'; s3<= '0'; s4<= pwm_in; s5<= '1'; s6<= '0';

                  WHEN OTHERS =>
                        s1<= '0'; s2<= '0'; s3<= '0'; s4<= '0'; s5<= '0'; s6<= '0';
            END CASE;
      END IF;

END PROCESS;

END ARCHITECTURE arch_driver;
```

Además forman parte del proyecto el generador de PWM:

/src/pwm_fpga.vhdl

```
library IEEE;
USE   ieee.std_logic_1164.all;
USE     ieee.std_logic_arith.all ;
USE   work.user_pkg.all;



ENTITY pwm_fpga IS

PORT (  clock,reset                :in  STD_LOGIC;
        Data_value            :in  std_logic_vector(7 downto 0);
        pwm                   :out STD_LOGIC
     );

END pwm_fpga;

ARCHITECTURE arch_pwm OF pwm_fpga IS

SIGNAL reg_out                            : std_logic_vector(7 downto 0);
SIGNAL cnt_out_int                        : std_logic_vector(7 downto 0);
SIGNAL pwm_int, rco_int                   : STD_LOGIC;

BEGIN

 -- 8 BIT DATA REGISTER TO STORE THE MARKING VALUES .
 -- THE MARKING VALUES WILL DETERMINE THE DUTY CYCLE OF PWM OUTPUT

 PROCESS(clock,reg_out,reset)
```

```
          BEGIN
                IF (reset ='1') THEN
                reg_out <="00000000";
                ELSIF (rising_edge(clock)) THEN
                      reg_out <= data_value;
                END IF;
          END PROCESS;


 -- 8 BIT UPDN COUNTER. COUNTS UP OR DOWN BASED ON THE PWM_INT SIGNAL AND GENERATES
 -- TERMINAL COUNT WHENEVER COUNTER REACHES THE MAXIMUM VALUE OR WHEN IT TRANSISTS
 -- THROUGH ZERO. THE TERMINAL COUNT WILL BE USED AS INTERRUPT TO AVR FOR GENERATING
 -- THE LOAD SIGNAL.
 -- INC and DEC are the two functions which are used for up and down counting. They are
defined in sepearate user_pakge library


 PROCESS (clock,cnt_out_int,rco_int,reg_out)


     BEGIN


      IF (rco_int = '1') THEN
          cnt_out_int <= reg_out;
      ELSIF  rising_edge(clock) THEN
           IF (rco_int = '0' and pwm_int ='1' and cnt_out_int <"11111111") THEN
            cnt_out_int <= INC(cnt_out_int);
           ELSE
               IF (rco_int ='0' and pwm_int ='0' and cnt_out_int > "00000000") THEN
               cnt_out_int <= DEC(cnt_out_int);
           END IF;
         END IF;
```

```
      END IF;
 END PROCESS;



-- Logic to generate RCO signal


 PROCESS(cnt_out_int, rco_int, clock,reset)
      BEGIN


           IF (reset ='1') THEN
             rco_int <='1';
          ELSIF rising_edge(clock) THEN
              IF ((cnt_out_int = "11111111") or (cnt_out_int ="00000000")) THEN
                 rco_int <= '1';
              ELSE
               rco_int <='0';
            END IF;
         END IF;


END PROCESS;



-- TOGGLE FLIP FLOP TO GENERATE THE PWM OUTPUT.


      PROCESS (clock,rco_int,reset)
      BEGIN
                  IF (reset = '1') THEN
                      pwm_int <='0';
              ELSIF rising_edge(rco_int) THEN
                pwm_int <= NOT(pwm_int);
```

```
              ELSE
                pwm_int <= pwm_int;
            END IF;
      END PROCESS;
      pwm <= pwm_int;

END arch_pwm;
```

Las funciones INC y DEC:

/src/usr_pkg.vhdl

```
-- ****************************************************************************
--
-- Company              : ATMEL CORPORATION
-- File Name            : user_pkg_inc_dec.vhd
-- Title                : USER DEFINED PACKAGE FILE
-- Version              : 1.0
-- Last updated         : 05/24/01
-- Target               :
--
-- Support email        : fpga@atmel.com
-- Support Hotline      : +1(408)436-4119 (USA)
--
-- Revision             : 1.0
--
-- OVERVIEW
-- This code describes the function used in the pwm_fpga.vhd file
--
-- ****************************************************************************
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE user_pkg IS
    function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
    function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
END user_pkg ;

PACKAGE BODY user_pkg IS

function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
  variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);
  begin
    XV := X;
    for I in 0 to XV'HIGH LOOP
      if XV(I) = '0' then
         XV(I) := '1';
         exit;
      else XV(I) := '0';
      end if;
    end loop;
  return XV;
end INC;

function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
    variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);
    begin
  XV := X;
  for I in 0 to XV'HIGH LOOP
```

```
      if XV(I) = '1' then
         XV(I) := '0';
         exit;
      else XV(I) := '1';
      end if;
  end loop;
return XV;


end DEC;


END user_pkg;
```

Y el testbench:

/tbench/pwm_pretb.vhd

```
LIBRARY ieee;


use ieee.std_logic_1164.all;




ENTITY pwm_fpga_test_bench IS


generic(clk_period:TIME:=100 ns);


END pwm_fpga_test_bench;


ARCHITECTURE arch_test_bench OF pwm_fpga_test_bench IS


COMPONENT pwm_fpga
PORT (
      clock: in std_logic;
      reset: in std_logic;
      data_value: in std_logic_vector(7 downto 0);
      pwm: out std_logic
);
END COMPONENT;


component driver_mosfet
PORT (
            s1 :out STD_LOGIC;
            s2 :out STD_LOGIC;
            s3 :out STD_LOGIC;
            s4 :out STD_LOGIC;
```

```
            s5 :out STD_LOGIC;
            s6 :out STD_LOGIC;
            hall_sensor :in  std_logic_vector(2 downto 0);
            reset :in STD_LOGIC;
            clock :in STD_LOGIC;
            pwm_in :in STD_LOGIC
);
end component;

-- Internal signal declaration
SIGNAL sig_clock  : std_logic;
SIGNAL sig_reset  : std_logic;
SIGNAL sig_data_value   : std_logic_vector(7 downto 0);
SIGNAL sig_pwm    : std_logic;
SIGNAL hall_sens : std_logic_vector(2 downto 0);
SIGNAL sig_s1 : std_logic;
SIGNAL sig_s2 : std_logic;
SIGNAL sig_s3 : std_logic;
SIGNAL sig_s4 : std_logic;
SIGNAL sig_s5 : std_logic;
SIGNAL sig_s6 : std_logic;
shared variable ENDSIM: boolean:=false;

BEGIN

-- Instantiating top level design Component pwm_fpga
inst_pwm_fpga : pwm_fpga
PORT MAP(
            clock => sig_clock,
```

```
            reset => sig_reset,
            data_value  => sig_data_value,
            pwm   => sig_pwm
      );
inst_mosfet_driver: driver_mosfet
PORT MAP(
            clock => sig_clock,
            reset => sig_reset,
            pwm_in => sig_pwm,
            hall_sensor => hall_sens,
            s1 => sig_s1,
            s2 => sig_s2,
            s3 => sig_s3,
            s4 => sig_s4,
            s5 => sig_s5,
            s6 => sig_s6
      );

clk_gen: process
            BEGIN

            If ENDSIM = FALSE THEN
                  sig_clock <= '1';
                  wait for clk_period/2;
                  sig_clock <= '0';
                  wait for clk_period/2;
            else
                  wait;
            end if;
      end process;
```

```
stimulus_process: PROCESS

BEGIN

      sig_reset <= '1';
      wait for 100 ns;
      sig_reset <= '0';
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="100";
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="110";
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="010";
```

```
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="011";
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="001";
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      hall_sens<="101";
      sig_data_value <= "11000000";
      wait for 150 us;
      sig_data_value <= "10000000";
      wait for 150 us;
      sig_data_value <= "01000000";
      wait for 150 us;
      wait;

 END PROCESS stimulus_process;
```

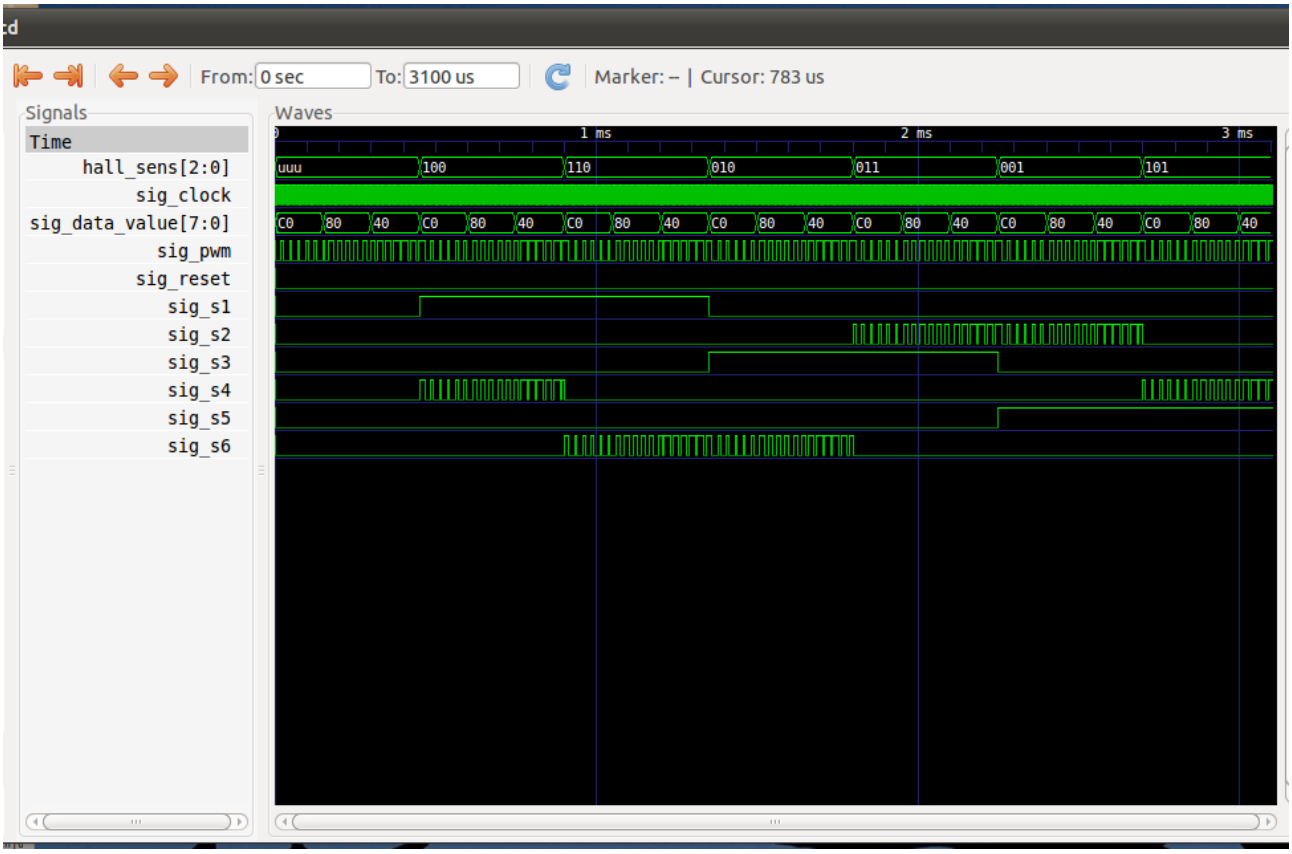
```
END arch_test_bench;
```

4.10. COMPILACIÓN Y SIMULACIÓN DEL POYECTO

Luego creamos una librería y la compilamos. Simulamos el funcionamiento del módulo descripto en el testbench y vemos los resultados como se observa a continuación:

```
ghdl -i --workdir=work ./src/*.vhd ./tbench/*.vhd
ghdl -m --ieee=synopsys --workdir=work pwm_fpga_test_bench
./pwm_fpga_test_bench --vcd=senales.vcd -stop-time=1300us
gtkwave senales.vcd
```

Si el procedimiento fué exitoso entonces se deberá ver una pantalla como la siguiente:



5. COMPILADOR

5.1. INTRODUCCIÓN AL COMPILADOR

El objetivo del presente trabajo es implementar un compilador para el robot M5 como caso de estudio.

Además pretendemos generar una guía rápida para aquellos que quieran hacer su propia experiencia con un parser y scanner open source.

En el trabajo práctico 1 hemos analizado la cinemática directa del robot.

En el trabajo práctico 2 analizamos la dinámica del robot, y presentamos una solución al control de motores, tal como si estuviéramos diseñando el robot.

Continuando con esta línea de trabajo vamos a implementar un compilador, cuyo objetivo es permitir a un supuesto operador manipular el robot.

Si se tratara de un compilador completo, su arquitectura debería ser:

Ilustración 1: Arquitectura del compilador real

Sin embargo la generación del código objeto está fuera del alcance de nuestro trabajo. El producto de salida de nuestro compilador será entonces información en la pantalla. Dicha información podría ser volcada a un archivo de texto y usando otro programa, que genere código objeto para la FPGA, se podría generar el código objeto del robot.

5.2. PARSER Y SCANNER

Para cumplir con el objetivo de nuestro compilador debemos usar dos programas. Un parser y un scanner.

El parser es un programa que recibe una serie de símbolos o TOKENS y analiza su sintaxis. Además la hace coincidir con una estructura previamente definida y en base a eso ejecuta una serie de comandos.

Escribir un parser es una tarea compleja, pero por suerte no hay que realizarla desde cero. Para esto utilizamos un software que se llama Bison. Este programa toma un archivo que explicaremos más adelante y genera como salida otro archivo .c que luego compilaremos con el GNU/GCC.

El scanner es un programa que recibe una serie de palabras, y a través del uso de expresiones regulares las convierte en tokens. Por ejemplo:

Entrada	Token
Números del 0 al 9	DIGITO
Palabra "movq"	FUNCIONMOVQ

Su producto de salida es la entrada del parser. Para generar el código del parser samos un software llamado Flex.

Ambos, el Flex y el Bison son programas de código abierto, que se pueden instalar en cualquier distribución de GNU/Linux.

El Bison proviene de un soft anterior llamado Yacc y el Flex de un soft llamado Lex.

Sintetizamos lo explicado en la siguiente figura:

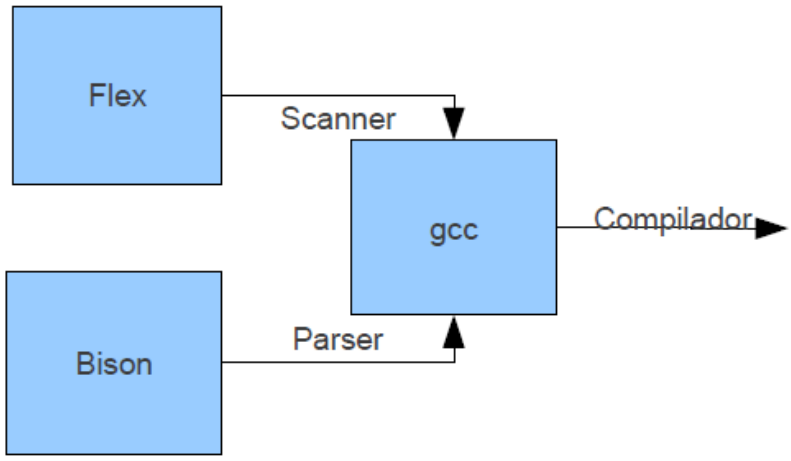


Ilustración 2: Arquitectura del compilador implementado

5.3. INSTALACIÓN

En cualquier distribución compatible con Debian, tipeamos en una consola las siguientes líneas para instalar el Flex y el Bison

```
sudo apt- get install flex
sudo apt-get install bison
```

5.4. ARCHIVOS FLEX

El formato de un archivo de Flex es el siguiente:

Declaraciones en C y del scanner (includes, defines, etc.)

Expresiones regulares y Tokens y los archivos se compilan de la siguiente forma:

```
flex nombrearchivo.l
```

Normalmente los archivos de Flex son de extensión .l, pues su formato original se llama Lex.

La salida del flex es, como se dijo anteriormente un archivo .c con su archivo .h asociado.

5.5. ARCHIVOS BISON

Los archivos de Bison tienen la siguiente estructura:

- Declaraciones en C y del parser
- Reglas gramáticas y acciones
- Subrutinas en C

Para generar un archivo de salida con el Bison, debemos ejecutar la siguiente línea de comando:

```
bison -d nombrearchivo.y
```

y tendremos entonces un archivo .c y su archivo .h asociado.

La extensión .y proviene del fomato Yacc.

5.6. MAKEFILE

También consideramos apropiado incluir en el trabajo un Makefile, archivo que se utiliza con el comando make de GNU/Linux para generar el compilador.

```
.PHONY : clean

all: flex bison source


source:

      gcc -lm -o robotica *.c

flex: *.l

      flex *.l


bison: *.y

      bison -d *.y


clean :

      rm -f *.c *.h robotica *tab*
```

5.7. COMPILADOR

Nuestro compilador sigue el formato de instrucciones del lenguaje C, al que ya estamos acostumbrados.

Implementa 2 funciones principales:

movq() y movq2q().

La primera, toma los valores de las 4 articulaciones del robot (estamos despreciando la muñeca) y calcula la posición del actuador final en un sistema cuyo origen de coordenadas se encuentra en la base del robot.

La segunda función, toma 2 juegos de valores de las articulaciones y calcula los puntos por los que pasará el actuador al pasar de los valores iniciales a los finales, en una cantidad de pasos que también es argumento de la función.

Además implementamos la función delay(ms) que genera una demora de tiempo en milisegundos y la función help que muestra un menú de ayuda.

Mostramos a continuación una captura de pantalla del proceso completo de utilización, desde la compilación al uso.

```
leandro@vilma:~/UTN/Robotica/Compila/moveq$ make

flex *.l

bison -d *.y

gcc -lm -o robotica *.c

leandro@vilma:~/UTN/Robotica/Compila/moveq$ ./robotica

help

Funciones soportadas:

      movq(q1,q2,q3,q4):

            Dados los 4 angulos de las articulaciones, en grados, indica la posición X Y
Z del robot

      movq2q(qi1,qi2,qi3,qi4,qf1,qf2,qf3,qf4,pasos):

            Dados 4 angulos iniciales, 4 angulos finales y la cantidad de pasos, calcula
la trayectoria del extremo final del robot

      delay(ms):

            Detiene la ejecucion del programa durante un tiempo expresado en ms

      exit:

            Sale del compilador

movq(10,20,30,40);

q1: 10.0 q2: 20.0 q3: 30.0 q4: 40.0=> x: 218.04 y: 19.01 z: 158.12

movq2q(0,0,0,0,10,20,30,40,10);

#0 q1: 0.0 q2: 0.0 q3: 0.0 q4: 0.0=> x: 250.96 y: 0.00 z: 30.00

#1 q1: 1.0 q2: 2.0 q3: 3.0 q4: 4.0=> x: 254.98 y: 2.24 z: 45.18

#2 q1: 2.0 q2: 4.0 q3: 6.0 q4: 8.0=> x: 257.52 y: 4.52 z: 59.96

#3 q1: 3.0 q2: 6.0 q3: 9.0 q4: 12.0=> x: 258.50 y: 6.78 z: 74.28

#4 q1: 4.0 q2: 8.0 q3: 12.0 q4: 16.0=> x: 257.82 y: 8.98 z: 88.07

#5 q1: 5.0 q2: 10.0 q3: 15.0 q4: 20.0=> x: 255.44 y: 11.08 z: 101.30

#6 q1: 6.0 q2: 12.0 q3: 18.0 q4: 24.0=> x: 251.33 y: 13.04 z: 113.92

#7 q1: 7.0 q2: 14.0 q3: 21.0 q4: 28.0=> x: 245.49 y: 14.83 z: 125.92

#8 q1: 8.0 q2: 16.0 q3: 24.0 q4: 32.0=> x: 237.95 y: 16.43 z: 137.28

#9 q1: 9.0 q2: 18.0 q3: 27.0 q4: 36.0=> x: 228.77 y: 17.82 z: 148.01

#10 q1: 10.0 q2: 20.0 q3: 30.0 q4: 40.0=> x: 218.04 y: 19.01 z: 158.12
```

```
delay(100);

Sleeping for 100 ms...

exit

Thanks for using this compiler, see you soon
```

A continuación listamos el código de los archivos para generar el parser y el scanner: /robotica. l

```
%{

#include <stdio.h>

#include "robotica.tab.h"

extern YYSTYPE yylval;



%}

digit [0-9]

%%

-?{digit}+        {

                        yylval.entero=atoi(yytext);

                        return(INTNUM);

                  }

-?{digit}+"."{digit}*         {

                        yylval.flotante=atof(yytext);

                        return(FLOATNUM);

                  };



movq                     return MOVQFUNC;

movq2q                   return MOVQ2QFUNC;

delay                    return DELAYFUNC;

help              return HELPFUNC;

exit              return EXITFUNC;

"("               return OPAR;
```

```
")"               return CPAR;
";"               return SEMICOLON;
","               return COMA;
"\n"                      { return (yytext[0]);}
[ \t]+                  /* ignore whitespace */;


%%
```

/robotica.y

```
%{

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>


#define L1 75
#define L2 125
#define L3 126
#define L4 45
#define PI 3.14
//Variables que uso despues, van aca o en el main?
float q1deg,q2deg,q3deg,q4deg;
//float qi1,qi2,qi3,qi4,qf1,qf2,qf3,qf4;


void yyerror(const char *str)
{
        fprintf(stderr,"error: %s\n",str);
}


int yywrap()
```

```
{
        return 1;
}

main()

{
        yyparse();
}

%}

%union{
      int entero;
      float flotante;
}

%token <entero>INTNUM
%token <flotante>FLOATNUM
%token MOVQFUNC MOVQ2QFUNC DELAYFUNC HELPFUNC EXITFUNC OPAR CPAR SEMICOLON COMA

%%
commands: /* empty */
        | commands command
        ;

command: '\n'
      |
      movq
```

```
        |
        movq2q
        |
        delay
      |
      help
      |
      exit
        ;


movq:
      MOVQFUNC OPAR INTNUM COMA INTNUM COMA INTNUM COMA INTNUM CPAR SEMICOLON '\n'
      {
            float x,y,z;
            float q1,q2,q3,q4;
            q1deg=$3;
            q2deg=$5;
            q3deg=$7;
            q4deg=$9;


            q1=q1deg*PI/180;
            q2=q2deg*PI/180;
            q3=q3deg*PI/180;
            q4=q4deg*PI/180;


            x=-( cos(q1)*cos(q2)*cos(q3)-cos(q1)*sin(q2)*sin(q3) )*L4*cos(q4+PI/2)-(
cos(q1)*sin(q2)*sin(q3)-cos(q1)*sin(q2)*cos(q3)
)*L4*sin(q4+PI/2)+cos(q1)*cos(q2)*L3*cos(q3)-
cos(q1)*sin(q2)*L3*sin(q3)+L2*cos(q2)*cos(q1);



            y= ( sin(q1)*cos(q2)*cos(q3)-sin(q1)*sin(q2)*sin(q3) )*L4*cos(q4+PI/2)-(
sin(q1)*sin(q2)*sin(q3)-sin(q1)*sin(q2)*cos(q3) )*L4*sin(q4 +
```

```
PI/2)+sin(q1)*cos(q2)*L3*cos(q3)-sin(q1)*sin(q2)*L3*sin(q3)+L2*sin(q2)*sin(q1);



            z= ( sin(q2)*cos(q3)+cos(q2)*sin(q3) )*L4*cos(q4+PI/2)-(
sin(q3)*sin(q2)+cos(q2)*cos(q3)
)*L4*sin(q4+PI/2)+sin(q2)*L3*cos(q3)+cos(q2)*L3*sin(q3)+L2*sin(q2)+L1;



            printf("q1: %.1f q2: %.1f q3: %.1f q4: %.1f=> x: %.2f y: %.2f z: %.2f
\n",q1deg,q2deg,q3deg,q4deg,x,y,z);




      };

movq2q:

      MOVQ2QFUNC OPAR INTNUM COMA INTNUM COMA INTNUM COMA INTNUM COMA INTNUM COMA INTNUM
COMA INTNUM COMA INTNUM COMA INTNUM CPAR SEMICOLON '\n'

      {



            float qi1=$3,qi2=$5,qi3=$7,qi4=$9,qf1=$11,qf2=$13,qf3=$15,qf4=$17;

            int steps=$19;

            int i;

            float x,y,z;

            float step1, step2, step3, step4;

            float step1deg, step2deg, step3deg, step4deg;

            float q1deg, q2deg, q3deg, q4deg;

            float q1,q2,q3,q4;



            step1deg=(qf1-qi1)/steps;

            step2deg=(qf2-qi2)/steps;

            step3deg=(qf3-qi3)/steps;

            step4deg=(qf4-qi4)/steps;



            q1deg=qi1;

            q2deg=qi2;
```

```
            q3deg=qi3;

            q4deg=qi4;


            qi1=qi1*PI/180;

            qi2=qi2*PI/180;

            qi3=qi3*PI/180;

            qi4=qi4*PI/180;


            qf1=qf1*PI/180;

            qf2=qf2*PI/180;

            qf3=qf3*PI/180;

            qf4=qf4*PI/180;


            step1=(qf1-qi1)/steps;

            step2=(qf2-qi2)/steps;

            step3=(qf3-qi3)/steps;

            step4=(qf4-qi4)/steps;




            for (i=0;i<=steps;i++)

            {

                  q1=qi1+step1*i;

                  q2=qi2+step2*i;

                  q3=qi3+step3*i;

                  q4=qi4+step4*i;


                  x=-( cos(q1)*cos(q2)*cos(q3)-cos(q1)*sin(q2)*sin(q3)
)*L4*cos(q4+PI/2)-( cos(q1)*sin(q2)*sin(q3)-cos(q1)*sin(q2)*cos(q3)
)*L4*sin(q4+PI/2)+cos(q1)*cos(q2)*L3*cos(q3)-
cos(q1)*sin(q2)*L3*sin(q3)+L2*cos(q2)*cos(q1);
```

```
                  y= ( sin(q1)*cos(q2)*cos(q3)-sin(q1)*sin(q2)*sin(q3)
)*L4*cos(q4+PI/2)-( sin(q1)*sin(q2)*sin(q3)-sin(q1)*sin(q2)*cos(q3) )*L4*sin(q4 +
PI/2)+sin(q1)*cos(q2)*L3*cos(q3)-sin(q1)*sin(q2)*L3*sin(q3)+L2*sin(q2)*sin(q1);



                  z= ( sin(q2)*cos(q3)+cos(q2)*sin(q3) )*L4*cos(q4+PI/2)-(
sin(q3)*sin(q2)+cos(q2)*cos(q3)
)*L4*sin(q4+PI/2)+sin(q2)*L3*cos(q3)+cos(q2)*L3*sin(q3)+L2*sin(q2)+L1;



                  printf("#%d q1: %.1f q2: %.1f q3: %.1f q4: %.1f=> x: %.2f y: %.2f z:
%.2f \n",i,q1deg+step1deg*i,q2deg+step2deg*i,q3deg+step3deg*i,q4deg+step4deg*i,x,y,z);



            }



      }



exit:

      EXITFUNC

      {

            printf("Thanks for using this compiler, see you soon\n");

            exit(0);

      };

delay:

      DELAYFUNC OPAR INTNUM CPAR SEMICOLON '\n'

      {

            int sleepTime=$3;

            printf("Sleeping for %d ms...",sleepTime);

            usleep(sleepTime*1000);

      };

help:

      HELPFUNC

      {

            printf("Funciones soportadas: \n\n\tmovq(q1,q2,q3,q4):\n\t\tDados los 4
```

```
angulos de las articulaciones, en grados, indica la posición X Y Z del robot\n");

            printf("\n\tmovq2q(qi1,qi2,qi3,qi4,qf1,qf2,qf3,qf4,pasos):\n\t\tDados 4
angulos iniciales, 4 angulos finales y la cantidad de pasos, calcula la trayectoria del
extremo final del robot\n");

            printf("\n\tdelay(ms):\n\t\tDetiene la ejecucion del programa durante un
tiempo expresado en ms\n");

            printf("\n\texit:\n\t\tSale del compilador\n");

      };

%%
```

6. CONCLUSIÓN

Tal como dijimos al inicio del trabajo, la robótica es una disciplina que es transversal a diversas disciplinas de la ingeniería. Combina cinemática, dinámica, electrónica, mecánica, e informática.

En principio podríamos pensar que algunos de estos aspectos exceden a nuestra formación en electrónica. Sin embargo a lo largo de la cursada y especialmente durante la realización de los trabajos prácticos, se pone de manifiesto que nuestra formación es esencialmente la de ingenieros. Y aquellos aspectos que parecían lejos de nuestro alcance, no lo estuvieron.

Además nos permitimos personalizar el trabajo. Decidimos utilizar para resolver parte de las consignas herramientas de código abierto. Esta decisión tiene que ver con que creemos que son estas las herramientas que deberíamos utilizar en un marco académico, y esperamos haber hecho aunque sea, una mínima contribución a la cátedra en este sentido.