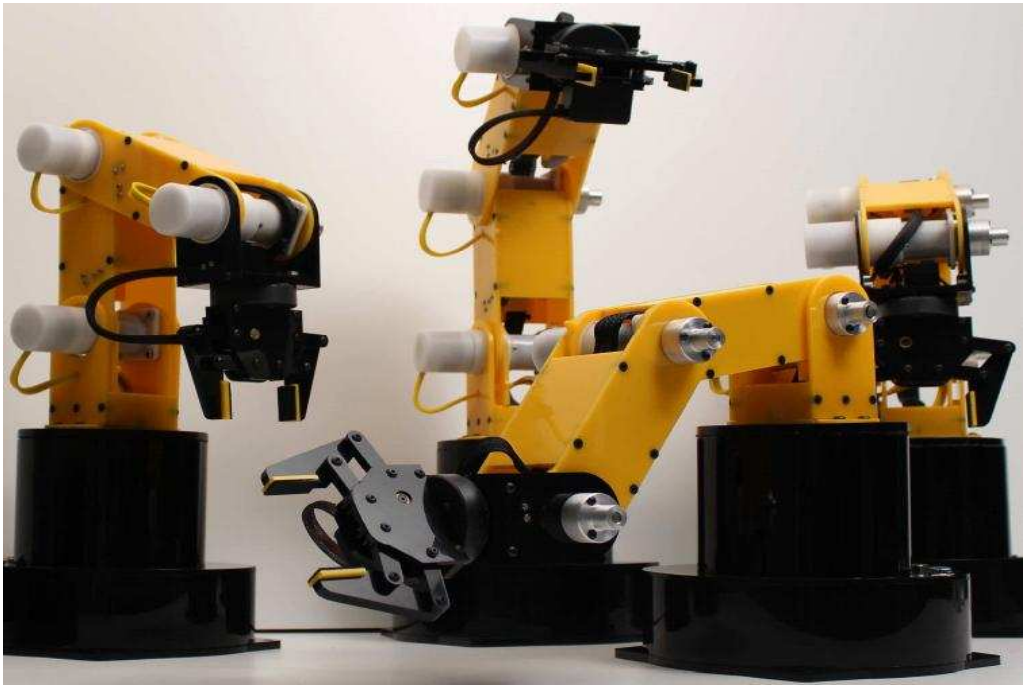
ROBOTICA
TESIS FINAL

Universidad Tecnológica Nacional

Facultad Regional Buenos Aires



Integrantes: Lee JoonJae,
Sebastian Papandrea.

Objetivos:

Introducción al M5

Análisis cinemático del M5

- ✓ Matriz homogénea cinemática directa del M5 en Matlab
- ✓ Desarrollo e implementación en CodeWarriror DSP56800/E
- ✓ Comparación entre DSP y Matlab

Análisis dinámico del M5

- ✓ Análisis dinámico de la estructura mecánica del M5 con Matlab
- ✓ Implementación en código VHDL
- ✓ Implementación PWM_FPGA:
- ✓ Implementación DRIVER MOSFET:
- ✓ Resultado de simulación en QuartusII

Desarrollo del compilador para el M5

- ✓ Introducción al compilador
- ✓ Introducción al Flex y Bison
- ✓ El Makefiles y su código
- ✓ El compilador y su código

Conclusiones finales.

1 - Introducción al M5:

El robot M5 es un brazo robótico de 5 grados de libertad con gripper, desarrollado especialmente para el uso educativo, de investigación y semi-industrial.
El M5 es una poderosa herramienta para aprender robótica, los conceptos electrónicos, programación y mecánica.

Este brazo robótico tiene características tales como:

- 5 grados de libertad + gipper
- Capacidad de carga de 1kg (con brazo extendido)
- Repetibilidad de +/- 0,5mm
- Alcance maximo de 750mm
- Velocidad maxima 25º/seg
- Actuadotes motorreductores de CC
- Transmisión directa
- Peso total 4kg

Rango de ejes:

- J1 –rotación de base +/-180º
- J2 –rotación del hombro +/-180º
- J3 –rotación del codo +/-180º
- J4 –rotación de la muñeca +/-180º
- J5 –rotación de la muñeca +/-180º

2 - Análisis cinemático del M5:

2.1 - Introducción teórica:

La mecánica es la parte de la física que estudia el movimiento, esta se subdivide en:

- ***Estática:*** Estudia las fuerzas en equilibrio mecánico.
- ***Cinemática:*** Estudia el movimiento sin tener en cuenta las causas que lo producen.
- ***Dinámica:*** Estudia los movimientos y las consecuencias que lo producen.

De estas tres formas de estudiar el movimiento en el presente trabajo práctico lo haremos a través de la cinemática, de la cual se introducen a continuación los conceptos teóricos.

2.1.1 Cinemática:

La cinemática es la parte de la mecánica que estudia las leyes del movimiento de los cuerpos sin tener en cuenta las causas que lo producen, limitándose al estudio de la trayectoria en función del tiempo.

La cinemática se subdivide en:

- ***Cinemática Inversa:*** Determina la configuración que debe adoptar el robot para alcanzar la posición y orientación del extremo
- ***Modelo Diferencial:*** Relaciona las velocidades de las articulaciones y las del extremo del robot.
- ***Cinemática Directa:*** **Determina la posición y orientación del extremo final del robot respecto al sistema de coordenadas de referencia, una vez conocidos los ángulos de las articulaciones y los parámetros geométricos de los elementos del robot.**

Entre la cinemática directa e inversa podemos fijar la siguiente relación

2.1.2 - Modelo cinemático

- *Método basado en relaciones geométricas*

  No es sistemático

  Es válido para robots de pocos grados de libertad

- *Método basado en matrices de transformación homogéneas*

  Es sistemático

  Es válido para robots con muchos grados de libertad

Para el desarrollo del estudio del movimiento del M5 escogimos hacerlo por cinemática directa, dicho método tiene las ventajas de: valer para robots de muchos grados de libertad, es sistemático y se basa en matrices de transformación homogéneas. Estos tres ítems serán descriptos a continuación.

2.1.3 - Cinemática directa x matriz de transf homogénea

Por medio de este método el problema cinemática directo se reduce a encontrar una matriz homogénea de transformación "T" que relacione la posición y orientación del extremo del robot respecto del sistema de referencias situado en la base del mismo. Esta matriz "T" será función de las coordenadas articulares.

Un robot de n grados de libertad está formado por n eslabones unidos por n articulaciones, de forma que cada par articulación-eslabón constituye un grado de libertad. De esta forma podemos notar que el M5 tiene 5 grados de libertad.

A cada eslabón se le puede asociar un sistema de referencias solidario a él y, utilizando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen al robot.

La matriz que representa la posición y orientación relativa entre los sistemas asociados a dos eslabones consecutivos del robot suele denominarse matriz $^{i-1}A_{i,}$

Así, en nuestro M5 la posición y orientación del sistema del quinto eslabón del robot respecto a la base del mismo donde se encuentra el sistema de coordenadas de referencias pude verse como:

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

Cundo se consideran todos los grados de libertad mediante cada una de las matrices, la matriz resultante se la denomina matriz de transformación T.

Para describir la relación que existe entre dos elementos contiguos hicimos uso de la representación de Denavit-Hanternber (D-H), este método matricial permite establecer de manera sistemática un sistema de coordenadas ligado a cada eslabón de una cadena articulada, pudiéndose determinar a continuación las ecuaciones cinemáticas de la cadena completa.

Según D-H, escogiendo adecuadamente las coordenadas asociadas a cada eslabón, será posible pasar de un eslabón al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas de cada eslabón.

Estas 4 transformaciones básicas permiten relacionar el sistema de referencias del elemento i con el sistema del elemento i-1.

El paso del sistema S_{i-1} al S_i mediante las 4 transformaciones básicas está garantizado solo si los sistemas han sido definidos de acuerdo a las siguientes reglas básicas:

1 – Rotación alrededor del eje z_{i-1} un ángulo θ_i

2 – Traslación a lo largo de Z_{i-1} una distancia d_i ; vector d_i $(0,0,d_i)$

3 - Traslación a lo largo de X_{i-1} una distancia a_i ; vector d_i $(0,0,a_i)$

4 – Rotación alrededor del eje X_{i-1} un ángulo α_i

Donde θ_i - α_i – a_i – d_i son los parámetros D-H del eslabón.

Siguiendo los pasos antes mencionados la matriz homogénea para el M5 queda de la siguiente forma:

$$^{i-1}A_i = T(Z, \theta_i) \,.\, T(0,0,d_i) \,.\, T(a_i,,0,0) \,.\, T(X, \alpha_i) =$$

$$= \begin{vmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La matriz definida anteriormente junto con la definición de los 4 parámetros de D-H conforman el siguiente algoritmo para la resolución del problema cinemática directo.

A continuación definiremos el Algoritmo de D-H para la obtención del modelo cinemática directo.

2.1.4 - Algoritmo de Denavit-Hartenberg

D-H 1. 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.

D-H 2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en n.

D-H 3. 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.

D-H 4. Para i de 0 *a* n-1 situar el eje **z**i sobre el eje de la articulación i + 1.

D-H 5. Situar el origen del sistema de la base {S0} en cualquier punto del eje **z**0 .Los ejes **x**0 e **y**0 se situaran de modo que formen un sistema dextrógiro con **z**0.

D-H 6. Para i de 1 a n-1, situar el sistema { Si }(solidario al eslabón i) en la intersección del eje **z**i con la línea normal común a **z**i-1 y **z**i .Si ambos ejes se cortasen se situaría {Si} en el punto de corte. Si fuesen paralelos { Si } se situaría en la articulación i + 1.

D-H 7. Situar **x**i en la línea normal común a **z**i-1 y **z**i .

D-H 8. Situar **y**i de modo que forme un sistema dextrógiro con **x**i y **z**i.

D-H 9. Situar el sistema { Sn } en el extremo del robot de modo que **z**n , coincida con la dirección de **z**n-1 y **x**n sea normal a **z**n-1 y **z**n.

D-H 10. Obtener θi como el ángulo que hay que girar en torno a **z**i-1 para que **x**i-1 y **x**i , queden paralelos.

D-H 11. Obtener di, como la distancia, medida a lo largo de **z**i-1,que habría que desplazar { Si-1 } para que **x**i y **x**i-1 quedasen alineados.

DH 12. Obtener ai como la distancia medida a lo largo de **x**i (que ahora coincidiría con **x**i-1)que habría que desplazar el nuevo { Si-1 } para que su origen coincidiese con { Si }.

DH 13. Obtener αi como el ángulo que habría que girar entorno a **x**i, (que ahora coincidiría con **x**i-1),para que el nuevo { Si-1 } coincidiese totalmente con { Si }.

DH 14. Obtener las matrices de transformación i-1Ai definidas anteriormente.

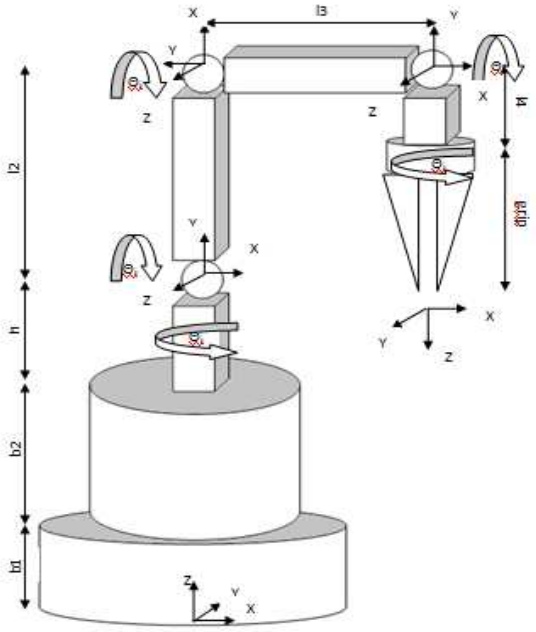
DH 15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot **T** = 0A1, 1A2 .....n-1An

DH 16. 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.

2.1.5 - Parámetros de D-H (θ_i - α_i – a_i – d_i):

- θ_i - Es el ángulo que forman los ejes X_{i-1} y X_i, medido en un plano perpendicular al eje Z_{i-1}, utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.
- d_i - Es la distancia a lo largo del eje Z_{i-1}. Es un parámetro variable en articulaciones prismáticas.
- a_i - Es la distancia a los largo del eje X_i. Es un parámetro variable en articulaciones giratorias.
- α_i - Es el ángulo de separación del eje Z_{i-1} y el eje Z_i utilizando la regla de la mano derecha.

2.2 - Modelo cinemática directo del M-5:



ARTICULACION	Θ	D	a	α
1	q1	b1+b2+l1	0	90
2	q2-90	0	l2	0
3	q3-90	0	l3	0
4	q4-90	0	l4	0
5	-90	0	0	90
6	q5	grip	0	0

2.2.1 - Matriz homogénea cinemática directa del M5 en Matlab:

```
syms b1 b2 l1 l2 l3 l4 q1 q2 q3 q4 q5 grip;
syms x0 y0 z0;
pi1=sym('pi');
A01=MDH(q1,b1+b2+l1,0,pi1/2);
A12=MDH(q2-(pi1)/2,0,l2,0);
A23=MDH(q3-(pi1)/2,0,l3,0);
A34=MDH(q4-(pi1)/2,0,l4,0);
A45=MDH(-(pi1)/2,0,0,pi1/2);
A56=MDH(q5,grip,0,0);
A46=A45*A56;
T = A01*A12*A23*A34*A46;
xyz=simplify(T*[x0;y0;z0;1]);

A=simple(xyz);
x=A(1);y=A(2); z=A(3);
```

2.2.2 - Desarrollo e implementación en CodeWarriror DSP 56800/E

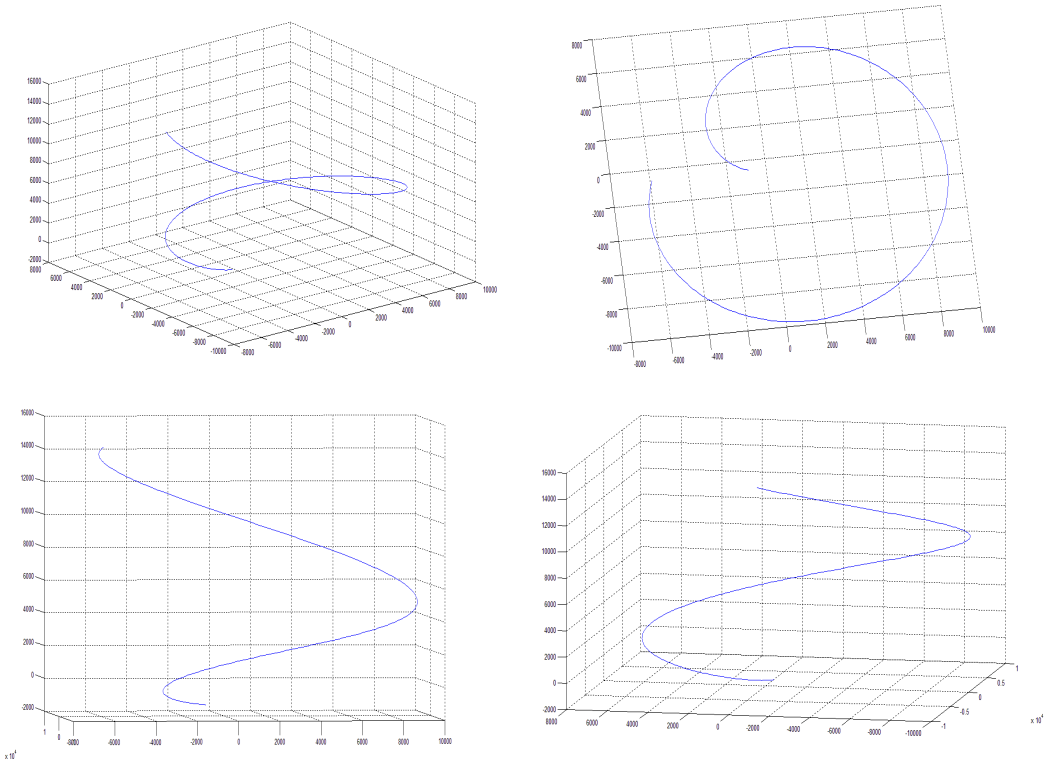
```
/* MODULE TPN1 */
#include "Cpu.h"
#include "Events.h"
#include "TFR1.h"
#include "MFR1.h"
#include "MEM1.h"
#include "XFR1.h"
#include "AFR1.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "stdio.h"

//*****************************************************************************
//            DEFINE
//*****************************************************************************
#define MXRAD 361 //100
#define PULSE2RAD 32767/MXRAD   // 32767/100 impulsos // #define PULSE2RAD 450
//*****************************************************************************
//            GLOBAL VARIABLE
//*****************************************************************************
```

```
Frac16 Cos1,Cos2,Cos3,Cos4,Cos5,Sin1,Sin2,Sin3,Sin4,Sin5,CosA,CosB,CosC,SinB,SinC;
Frac16 b1,b2,l1,l2,l3,l4,q1,q2,q3,q4,q5,grip;
Frac16 x, y, z, x0, y0, z0,ZA,ZB,ZC;
int i,j,k;

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

  /* Write your code here */
        b1=(0x10000/100)*10;b2=0x10000*5/100;
        l1=(0x10000/100)*10;l2=0x10000*10/100;l3=0x10000*10/100;l4=0x10000*10/100;grip=0x10000*2/100;
        x0=(0x10000/100)*1;y0=(0x10000/100)*1;z0=(0x10000/100)*1;
        printf ("₩nx₩ty₩tz");
        printf ("₩n₩n!!!!!!!!!! VARIANDO q4<q3<q1 y q5=q1 !!!!!!!!!!!!₩n₩n");

        for(q1=0,q2=0,q3=0,q4=0,q5=0;(q1<0xFFFF-200);q1 += 200,q3 += 50,q4 += 50,q5 += 200)
        {
                ZA=add(q2,add(q3,q4));
                ZB=add(ZA,q5);
                ZC=sub(ZA,q5);
                Cos1 = tfr16CosPIx (q1);
                Cos2 = tfr16CosPIx (q2);
                Cos3 = tfr16CosPIx (q3);
                Cos4 = tfr16CosPIx (q4);
                Cos5 = tfr16CosPIx (q5);
                Sin1 = tfr16SinPIx (q1);
                Sin2 = tfr16SinPIx (q2);
                Sin3 = tfr16SinPIx (q3);
                Sin4 = tfr16SinPIx (q4);
                Sin5 = tfr16SinPIx (q5);
                CosA = tfr16CosPIx (ZA);
                CosB = tfr16CosPIx (ZB);
                CosC = tfr16CosPIx (ZC);
                SinB = tfr16SinPIx (ZB);
                SinC = tfr16SinPIx (ZC);

                x= add(add(mult(l2,mult(Cos1,Sin2)),add(mult(y0,mult(Cos5,Sin1)),mult(x0,mult(Sin1,Sin5)))),
add(add(negate(mult(l3,mult(Cos1,mult(Cos2,Cos3)))),mult(l3,mult(Cos1,mult(Sin2,Sin3)))),
add(add(mult(grip,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))),mult(grip,mult(mult(Cos1,Cos2),mult(Cos4,Sin3)))),
add(add(mult(grip,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))),negate(mult(l4,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))))),
add(negate(add(mult(l4,mult(mult(Cos1,Cos2),mult(Cos4,Sin3))),mult(l4,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))))),
add(add(mult(z0,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))),mult(z0,mult(mult(Cos1,Cos2),mult(Cos4,Sin3)))),
add(add(mult(z0,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))),negate(mult(grip,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(l4,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))),negate(mult(z0,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))))),
```

```
add(negate(add(mult(mult(y0,Cos1),mult(mult(Cos2,Cos3),mult(Cos4,Sin5))),mult(mult(x0,Cos1),mult(mult(Cos2,Cos5),mult(Sin3,Sin4))))),
add(negate(add(mult(x0,mult(Cos1,mult(mult(Cos3,Cos5),mult(Sin2,Sin4))))),mult(x0,mult(Cos1,mult(mult(Cos4,Cos5),mult(Sin2,Sin3)))))),
add(add(mult(y0,mult(Cos1,mult(mult(Cos2,Sin3),mult(Sin4,Sin5))))),mult(y0,mult(Cos1,mult(mult(Cos3,Sin2),mult(Sin4,Sin5))))),
add(mult(y0,mult(Cos1,mult(mult(Cos4,Sin2),mult(Sin3,Sin5))))),mult(x0,mult(Cos1,mult(mult(Cos2,Cos3),mult(Cos4,Cos5))))))))))))));

                    y=add(add(mult(l2,mult(Sin1,Sin2)),negate(add(mult(x0,mult(Cos1,Sin5)),mult(y0,mult(Cos1,Cos5))))),
add(add(negate(mult(l3,mult(Cos2,mult(Cos3,Sin1))))),mult(l3,mult(Sin1,mult(Sin2,Sin3))))),
add(add(mult(grip,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))),mult(grip,mult(mult(Cos2,Cos4),mult(Sin1,Sin3))))),
add(add(mult(grip,mult(mult(Cos3,Cos4),mult(Sin1,Sin2))),negate(mult(l4,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))))),
add(negate(add(mult(l4,mult(mult(Cos2,Cos4),mult(Sin1,Sin3))),mult(l4,mult(mult(Cos3,Cos4),mult(Sin1,Sin2))))),
add(add(mult(z0,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))),mult(z0,mult(mult(Cos2,Cos4),mult(Sin1,Sin3))))),
add(add(mult(z0,mult(mult(Sin3,Sin4),mult(Sin1,Sin2))),negate(mult(grip,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(l4,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))),negate(mult(z0,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(x0,mult(Cos2,mult(mult(Cos3,Cos4),mult(Cos5,Sin1))))),negate(mult(y0,mult(Cos2,mult(mult(Cos3,Cos4),mult(Sin1,Sin5)))))),
add(negate(add(mult(mult(x0,Cos2),mult(mult(Cos5,Sin1),mult(Sin3,Sin4))),mult(mult(x0,Cos3),mult(mult(Cos5,Sin1),mult(Sin2,Sin4))))),
add(add(negate(mult(mult(x0,Cos4),mult(mult(Cos5,Sin1),mult(Sin2,Sin3))))),mult(mult(y0,Cos2),mult(mult(Sin1,Sin3),mult(Sin4,Sin5))))),
add(mult(mult(y0,Cos3),mult(mult(Sin1,Sin2),mult(Sin4,Sin5))),mult(mult(y0,Cos4),mult(mult(Sin1,Sin2),mult(Sin3,Sin5))))))))))))));

                    z=add(add(b1,add(b2,add(l1,negate(add(mult(l3,tfr16SinPIx (add(q2,q3))),mult(l2,Cos2))))))),
add(mult(add(mult(y0,CosB),add(mult(x0,SinB),add(mult(x0,SinC),negate(mult(y0,CosC))))),FRAC16(0.5)),
mult(negate(add(grip,add(negate(l4),z0))),CosA)));

                    printf ("₩n%d₩t%d₩t%d;", x, y, z);
          }
}
```

2.2.3 - Resultado de la simulación:

xyz=['tabla generada por el codewarrior'];plot3(xyz(:,1,1),xyz(:,2,1),xyz(:,3,1));grid;



2.2.4 - Verificación de código C en Matlab

```
b1=5.85;
b2=9.6;
l1=7;
l2=12.7;
l3=12.7;
l4=4.4;
grip=10;
x=zeros(30000,1);y=zeros(30000,1);z=zeros(30000,1);
x0=0;y0=0;z0=0;
q1=0;q2=0;q3=0;q4=0;q5=0;

for i=1:30000
q2=q2+50;
ZA=q2+q3+q4;
ZB=ZA+q5;
ZC=ZA-q5;

x(i)= l2*cos(q1)*sin(q2) + y0*cos(q5)*sin(q1) + x0*sin(q1)*sin(q5)  ...
       - l3*cos(q1)*cos(q2)*cos(q3) + l3*cos(q1)*sin(q2)*sin(q3)  ...
```

```
      + grip*cos(q1)*cos(q2)*cos(q3)*sin(q4) +
      grip*cos(q1)*cos(q2)*cos(q4)*sin(q3)   ...
      + grip*cos(q1)*cos(q3)*cos(q4)*sin(q2) -
      l4*cos(q1)*cos(q2)*cos(q3)*sin(q4)  ...
      - l4*cos(q1)*cos(q2)*cos(q4)*sin(q3) -
      l4*cos(q1)*cos(q3)*cos(q4)*sin(q2)  ...
      + z0*cos(q1)*cos(q2)*cos(q3)*sin(q4) +
      z0*cos(q1)*cos(q2)*cos(q4)*sin(q3)  ...
      + z0*cos(q1)*cos(q3)*cos(q4)*sin(q2) -
      grip*cos(q1)*sin(q2)*sin(q3)*sin(q4)  ...
      + l4*cos(q1)*sin(q2)*sin(q3)*sin(q4) -
      z0*cos(q1)*sin(q2)*sin(q3)*sin(q4)  ...
      - y0*cos(q1)*cos(q2)*cos(q3)*cos(q4)*sin(q5) -
      x0*cos(q1)*cos(q2)*cos(q5)*sin(q3)*sin(q4)  ...
      - x0*cos(q1)*cos(q3)*cos(q5)*sin(q2)*sin(q4) -
      x0*cos(q1)*cos(q4)*cos(q5)*sin(q2)*sin(q3)  ...
      + y0*cos(q1)*cos(q2)*sin(q3)*sin(q4)*sin(q5) +
      y0*cos(q1)*cos(q3)*sin(q2)*sin(q4)*sin(q5)  ...
      + y0*cos(q1)*cos(q4)*sin(q2)*sin(q3)*sin(q5) +
      x0*cos(q1)*cos(q2)*cos(q3)*cos(q4)*cos(q5);

y(i)=l2*sin(q1)*sin(q2) - x0*cos(q1)*sin(q5) - y0*cos(q1)*cos(q5)  ...
      - l3*cos(q2)*cos(q3)*sin(q1) + l3*sin(q1)*sin(q2)*sin(q3)  ...
      + grip*cos(q2)*cos(q3)*sin(q1)*sin(q4) +
      grip*cos(q2)*cos(q4)*sin(q1)*sin(q3)  ...
      + grip*cos(q3)*cos(q4)*sin(q1)*sin(q2) -
      l4*cos(q2)*cos(q3)*sin(q1)*sin(q4)  ...
      - l4*cos(q2)*cos(q4)*sin(q1)*sin(q3) -
      l4*cos(q3)*cos(q4)*sin(q1)*sin(q2)  ...
      + z0*cos(q2)*cos(q3)*sin(q1)*sin(q4) +
      z0*cos(q2)*cos(q4)*sin(q1)*sin(q3)  ...
      + z0*cos(q3)*cos(q4)*sin(q1)*sin(q2) -
      grip*sin(q1)*sin(q2)*sin(q3)*sin(q4) ...
      + l4*sin(q1)*sin(q2)*sin(q3)*sin(q4) -
      z0*sin(q1)*sin(q2)*sin(q3)*sin(q4)  ...
      + x0*cos(q2)*cos(q3)*cos(q4)*cos(q5)*sin(q1) -
      y0*cos(q2)*cos(q3)*cos(q4)*sin(q1)*sin(q5)  ...
      - x0*cos(q2)*cos(q5)*sin(q1)*sin(q3)*sin(q4) -
      x0*cos(q3)*cos(q5)*sin(q1)*sin(q2)*sin(q4)  ...
      - x0*cos(q4)*cos(q5)*sin(q1)*sin(q2)*sin(q3) +
      y0*cos(q2)*sin(q1)*sin(q3)*sin(q4)*sin(q5)  ...
      + y0*cos(q3)*sin(q1)*sin(q2)*sin(q4)*sin(q5) +
      y0*cos(q4)*sin(q1)*sin(q2)*sin(q3)*sin(q5) ;

z(i)=b1 + b2 + l1 - l3*sin(q2 + q3) - l2*cos(q2) ...
      + (y0*cos(ZB) + x0*sin(ZB) + x0*sin(ZC) - y0*cos(ZC))*0.5  ...
      -(grip - l4 + z0)*cos(ZA) ;
end
plot3(x,y,z);grid;
```

3- Dinámica

3.1 - Introducción teórica:

La mecánica es la parte de la física que estudia el movimiento, esta se subdivide en:

Estática: Trata sobre las fuerzas en equilibrio mecánico.

Cinemática: Estudia el movimiento sin tener en cuenta las causas que lo producen, este fue el estudio realizado en el TP1 donde estudiamos el movimiento del robot M5.

Dinámica: Estudia el movimiento y las causas que lo producen, este es el estudio que realizaremos en los siguientes apartados.

Dinámica del Robot

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

El modelo dinámico aplicado a un robot, en nuestro caso al M5, tiene por objetivo conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo. Esta relación se obtiene mediante el denominado modelo dinámico, que relaciona matemáticamente:

- La localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo y sus derivadas: velocidad y aceleración.
- Las fuerzas y pares aplicados en las articulaciones o en el extremo del robot.
- Los parámetros dimensionales del robot: longitud, masa, inercias de sus elementos, etc.

La obtención del modelo dinámico para mecanismos de uno o dos grados de libertad no es excesivamente complejo, pero a medida que el número de grados de libertad aumenta, el planteamiento y obtención del modelo dinámico se dificulta enormemente.

Por este motivo no siempre es posible obtener un modelo dinámico expresado mediante una serie de ecuaciones, normalmente del tipo diferencial de 2do orden, cuya integración permita conocer que movimiento surge al aplicar unas fuerzas o que fuerzas hay que aplicar para obtener un movimiento determinado.

El modelo dinámico debe ser resuelto de manera iterativa mediante la utilización de un procedimiento numérico.

La dificultad de la obtención del modelo dinámico de un robot es uno de los aspectos más complejos de la robótica, lo que ha llevado a ser obviado en numerosas ocasiones.

El diseño y la evaluación del control dinámico del robot, es evidentemente de gran importancia, pues de la calidad del control dinámico del robot depende la precisión y la velocidad de sus movimientos.

Es importante notar que el modelo dinámico completo debe incluir no solo la dinámica de sus elementos, tales como, barras o elementos sino también la dinámica propia de sus sistemas de transmisión, la de los actuadores y la de sus equipos electrónicos de mando. Estos elementos incorporan al modelo dinámico nuevas inercias, rozamientos, saturaciones de los circuitos electrónicos, etc. aumentando así su complejidad.

Aplicaciones de este tipo pueden encontrarse en la robótica especial o en robots de grandes dimensiones, entre otras.

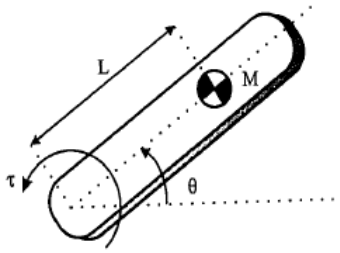
La gran complejidad a la hora de obtener el modelo dinámico del robot, ha motivado que se realicen ciertas simplificaciones, de manera que así pueda ser utilizado en el diseño del controlador.

Para hallar el modelo dinámico del M5 lo hemos considerado a este como un cuerpo rígido, no contemplando los sistemas de transmisión, actuadores, etc.

3.2 - Modelo dinámico de un robot rígido

En este caso se analizar una articulación, el equilibrio de fuerzas-pares dará como resultado la ecuación:

$$\tau = I\frac{d^2\theta}{dt^2} + M\,g\,L\cos\theta = M\,L^2\ddot{\theta} + M\,g\,L\cos\theta$$



En donde suponemos que toda la masa se encuentra concentrada en el centro de gravedad del elemento.

Para un par motor "τ" determinado, la integración de la ecuación dará lugar a la expresión θ(t) y con sus derivadas se conocerá la evolución de la coordenada articular junto con velocidad y aceleración (método dinámico directo)

Si se pretende que θ(t)evolucione según una determinada función del tiempo, sustituyendo en la formula anterior se podrá obtener el par τ(t) que es necesario aplicar (método dinámico inverso)

Si el robot tuviese que ejercer alguna fuerza en su extremo, al manipular una carga bastaría con incluir esta condición en la formula anterior y proceder del mismo modo.

3.3 - Métodos para hallar el modelo dinámico

- Metodo de Lagrange-Euler
- Metodo de Newton-Euler
- Metodo Variables de estado
- Metodo de Kane

Para nuestro análisis nos basaremos en el Método de Newton – Euler recurrente y para esto utilizaremos la herramienta ToolKit HEMERO y el toolbooks de Peter Crokepara Matlab.

3.3.1 - Breve introducción al Método de Neuton-Euler

Formulación de Newton-Euler utiliza las ecuaciones de equilibrio de fuerzas y torques

$\Sigma F = mv$

$\Sigma T = I\omega\cdot + \omega \times (I\omega)$

Un adecuado desarrollo de estas ecuaciones conduce a una formulación recursiva en la que se obtienen la posición, velocidad y aceleración del eslabón i referido a la base del robot a partir de los correspondientes valores de (posición, velocidad y aceleración) del eslabón i-1 y del movimiento relativo de la articulación i. De este modo, partiendo del eslabón 1 se llega al eslabón n. Con estos datos se procede a obtener las fuerzas y torques actuantes sobre el eslabón i referidos a la base del robot a partir de los correspondientes valores del eslabón i + 1, recorriéndose de esta forma todos los eslabones desde el eslabón n al eslabón 1.

El algoritmo se basa en operaciones vectoriales (con productos escalares y vectoriales entre magnitudes vectoriales, y productos de matrices con vectores) siendo más eficiente en comparación con las operaciones matriciales asociadas a la formulación Lagrangiana.
De hecho, el orden de complejidad computacional de la formulación recursiva de Newton-Euler es O(n)lo que indica que depende directamente del número de grados de libertad. Si lo comparamos con la formulación Lagrangiana, en esta, el orden de complejidad computacional es O(n^4). Es decir, el número de operaciones a realizar crece con la potencia 4 del número de grados de libertad.

Datos de cada eslabón

Antes de poder aplicar los algoritmos necesitamos los siguientes datos de cada eslabón del robot:

- Masa
- Centro de Masa o Gravedad
- Inercias
- Gravedad

Para ello utilizamos un programa de dibujo CAD, llamado T-FLEX el cual ofrece la posibilidad de dibujar elementos en 3D. Además tiene características muy importantes ya que se pueden calcular

los parámetros esenciales de cada pieza (centro de masa, inercias, peso, etc), gracias a que se puede especificar también el material con la que están hechas.

El software tiene 2 posibilidades para graficar en 3-D, la primera es dibujar sobre un plano y luego mediante instrucciones expandir el dibujo a uno de 3-D. La otra es directamente hacer un dibujo virtual de un espacio con los 3 planos ortogonales dibujados.

El robot M5 consta de 6 piezas esenciales:

- 1 base
- 1° eslabón
- 3 eslabones siguientes (son distintos al primero)
- 1 gripper

Nosotros no tuvimos en cuenta el gripper para los cálculos del robot por lo que nos quedan 5 piezas, de las cuales dibujar a la perfección es muy difícil y trabajoso. Como ejemplo los eslabones no son una única pieza sino que constan de 4 piezas, 1 por cada lado y son huecos. Al querer dibujar algo similar no era posible calcular los parámetros esenciales

Además debemos mencionar que para realizar las piezas nos basamos en el trabajo de un compañero llamado Daniel Abaca, que nos facilitó sus piezas y nosotros hicimos pequeños cambios, ya que nuestras eslabones son todos sólidos. Esto provoca que la dinámica no sea perfecta, sumado que la base también es un bloque sólido y, en la realidad para conocer la masa del elemento que rota en la base habría que desarmar esa parte del robot.

Piezas realizadas en T-FLEX (facilitadas por Daniel Abaca)

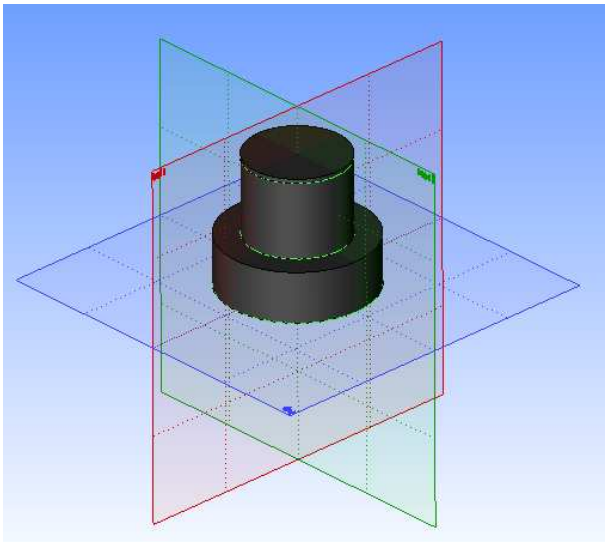


Imagen de la base del robot

Los datos de esta pieza son los siguientes:

Mass=1.22145kg
Xmass=0
Ymass=0
Zmass=105mm
Inerciax= 15390,3
Inerciay= 15390,3
Inerciaz= 2198,61
Inercia-yz=0
Inercia-zx=0
Inercia-xy=0

Los datos de esta pieza son los siguientes:

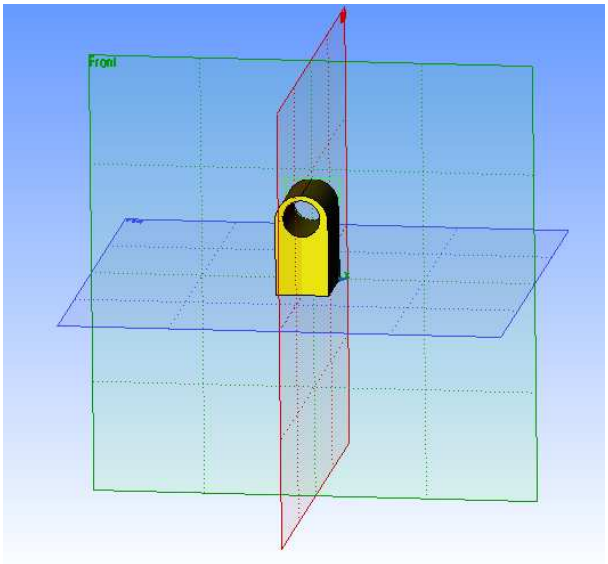


Imagen del primer eslabón del robot

Mass=0,2891kg
Xmass=0
Ymass= 33,75 mm
Zmass= 37,89mm
Inerciax=1026,5
Inerciay=653,73
Inerciaz=505,49
Inercia-yz=0
Inercia-zx=0

Inercia-xy=0

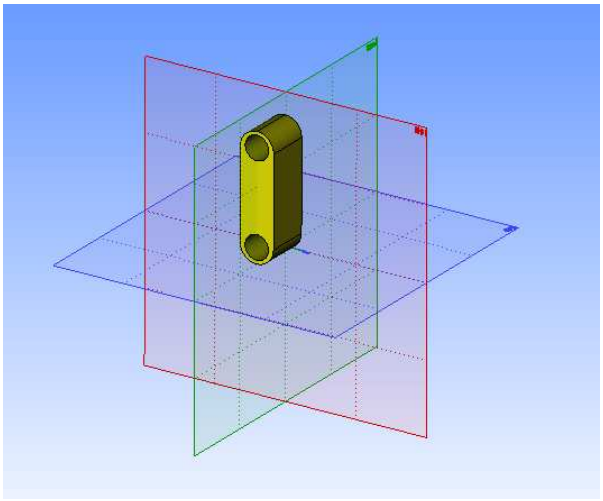


Imagen de los eslabones 2 3 y 4

Mass=0,5022kg
Xmass=-34,5 mm
Ymass= -0,0088 mm
Zmass= 90
Inerciax=5071,27
Inerciay=5724,21
Inerciaz=885,433
Inercia-yz=0
Inercia-zx=0
Inercia-xy=0,3

3.4 - Análisis dinámico de la estructura mecánica del M5 con Matlab

```
%%Dinamica del robot M5.
clear all;
clc;
close all;

syms t;

%% Iniciamos los datos del Robot
% Longitudes de los eslabones en metro
b1=(96+59+66+7)/1000;   % distancia entre el suelo y el primer motor
b2=(112.5+14)/1000;
b3=(112.5+14)/1000;
b4=(112.5+14)/1000;

% Denavit Hartenberg.
% L =LINK([alpha A theta D sigma])
L{1}=link([pi/2 0 0 b1],'standard');  %desde la base al primer munion
L{2}=link([0 0 pi/2 0],'standard');   %rota en z (por necesidad)
L{3}=link([0 b2 0 0],'standard');     %traslado en x (articulación giro enz)
L{4}=link([0 b3 -pi/2 0],'standard');%giro en z para correr x y
L{5}=link([0 b4 -pi/2 0],'standard');
```

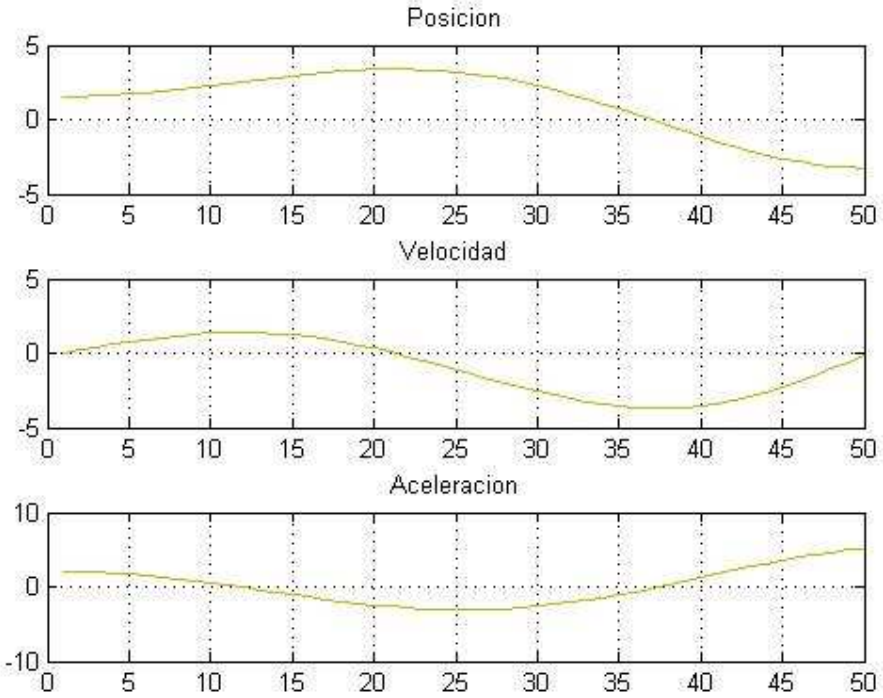
```
L{6}=link([0 0 pi/2 0],'standard');

% Masas de los eslabones en kg normalizado
L{1}.m = 0.8133;
L{2}.m = 0;
L{3}.m = 0.8133;
L{4}.m = 0.8133;
L{5}.m = 0.8133;
L{6}.m = 0;

% Coordenadas de los centros de masa de cada eslabon en metros.

L{1}.r = [0 0 0.105];
L{2}.r = [0 0 0];
L{3}.r = [0 0.03375 0.0379];
L{4}.r = [-0.0345 0 0.09];
L{5}.r = [-0.0345 0 0.09];
L{6}.r = [0 0 0];

% 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];
L{5}.I = [0 0 0 0.3 0 0];
L{6}.I = [0 0 0 0 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;
L{5}.Jm = 0;
L{6}.Jm = 0;

% Coeficiente de reducción del actuador
L{1}.G = 1;
L{2}.G = 0;
L{3}.G = 1;
L{4}.G = 1;
L{5}.G = 1;
L{6}.G = 0;

% Rozamiento viscoso del actuador
L{1}.B = 0;
L{2}.B = 0;
L{3}.B = 0;
L{4}.B = 0;
L{5}.B = 0;
L{6}.B = 0;

% Rozamiento de Coulomb en la dirección positiva y negativa del actuador
L{1}.Tc = [ 0 0 ];
L{2}.Tc = [ 0 0 ];
L{3}.Tc = [ 0 0 ];
L{4}.Tc = [ 0 0 ];
L{5}.Tc = [ 0 0 ];
```

```
L{6}.Tc = [ 0 0 ];

% Definicion del robot.
% Constructor de objetos del Robot
% function r = robot(L, a1, a2, a3).
Robot5_6 = robot(L, 'Polar RD', 'UTN_Robotics', 'robot cilindrico 2 GLD');
Robot5_6.name = 'Polar RD';
Robot5_6.manuf = 'UTN_Robotics';

% Se define la gravedad.
grav = [ 0 0 -9.8];

%% Determina la trayectoria
% Se define la posicion.
for i=1:6
sq(i,1)= sin(t) * t + pi/2;
end
% Obtenemos la velocidad y aceleracion.
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=1:6
    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

%% Dibujamos la trayectoria.
figure(1);
subplot(3,1,1);
plot(q);
title('Posicion');
grid;
subplot(3,1,2);
plot(qd);
grid
title('Velocidad');
subplot(3,1,3);
grid;
plot(qdd);
title('Aceleracion');
grid;

%% Calculo de matriz de la 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);

%% Graficamos la gravedad, coriolis, torque
```

```
j=2;
for i=1:1:5
figure(j);
   j=j+1;
subplot(4,1,1);
plot(Tau(:,i));
title('Torque');
grid
subplot(4,1,2);
plot(TauI(:,i));
title('Inercia');
grid
subplot(4,1,3);
plot(TauC(:,i));
title('Coriolis');
grid
subplot(4,1,4);
plot(TauG(:,i));
title('PAR Gravedad');
grid
end
```

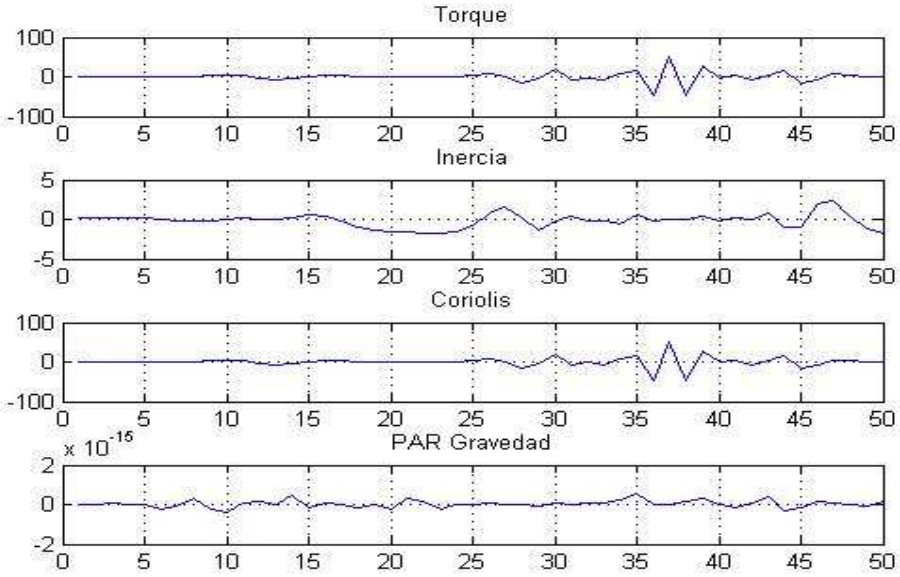
3.4.1 - Resultados de la simulación dinámica del M5 en Matlab

Propusimos una trayectoria sinusoidal y un tiempo para realizarla con esto obtuvimos la velocidad y aceleración que se muestran en la figura.

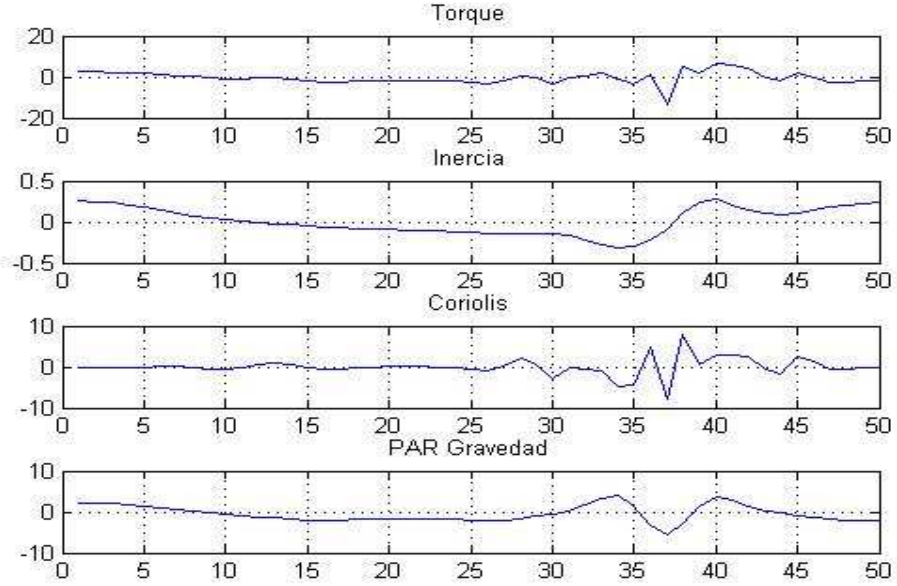


Partiendo de la Posición, velocidad y Aceleración y teniendo en cuanta la estructura mecánica del M5 obtuvimos el torque, inercia, coriolis y par gravedad para cada una de los eslabones.

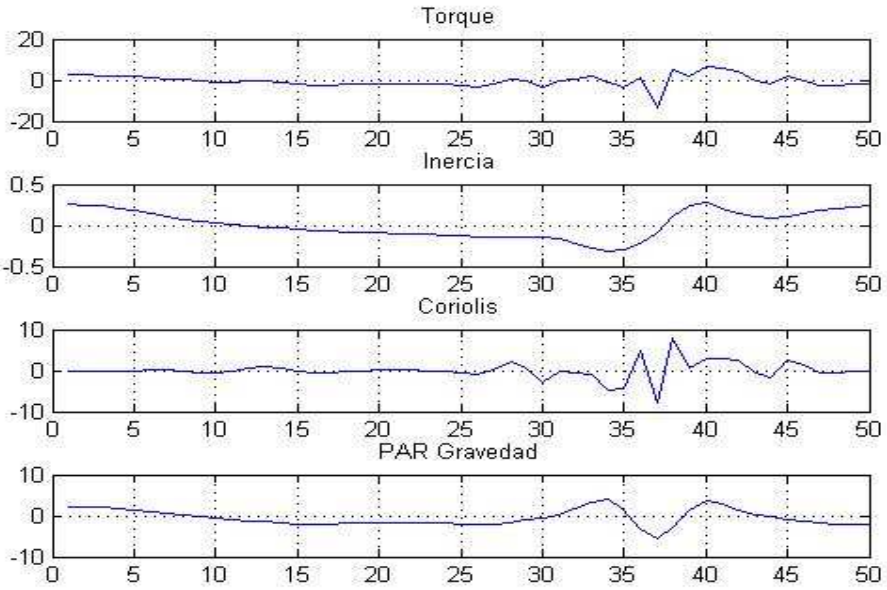
Base



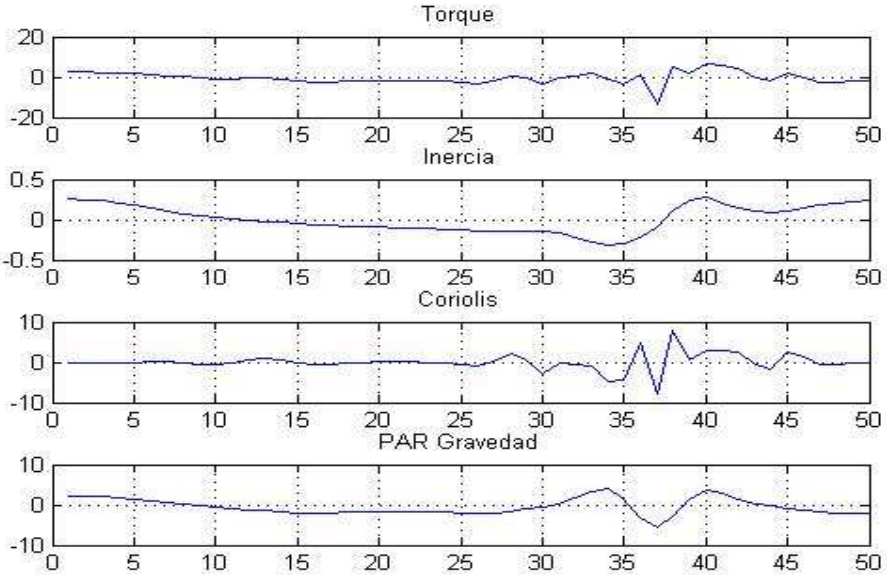
Primer eslabón



Segundo eslabón

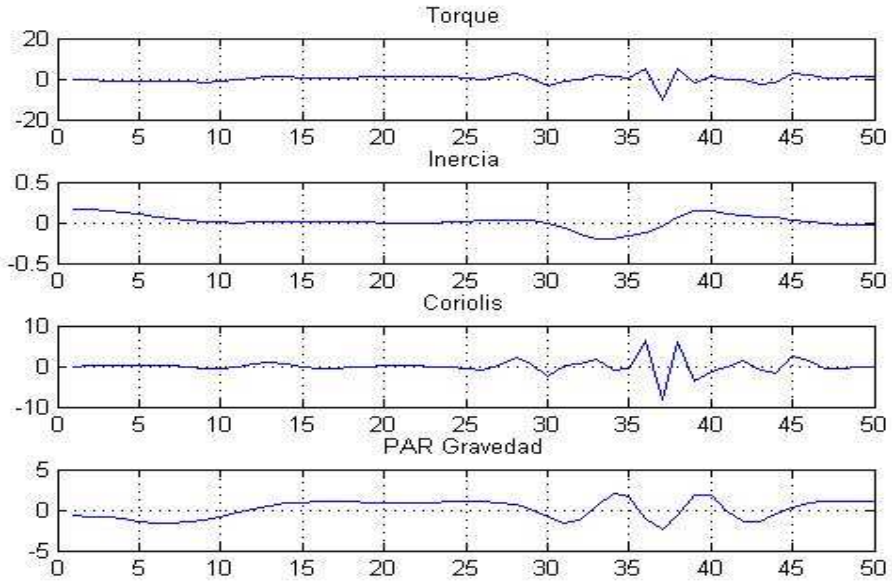


Tercer eslabón

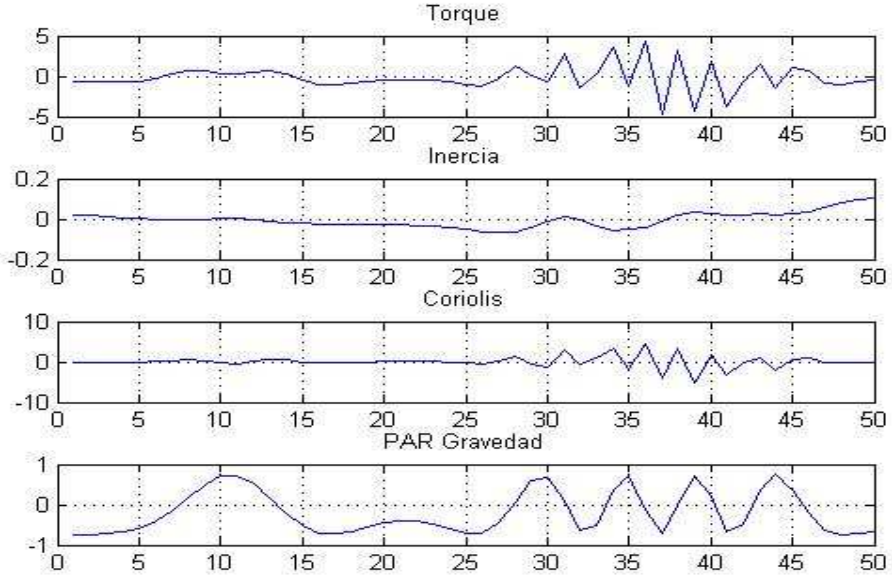


Cuarto eslabón (eslabón agregado para cumplir con las condiciones de D-H,

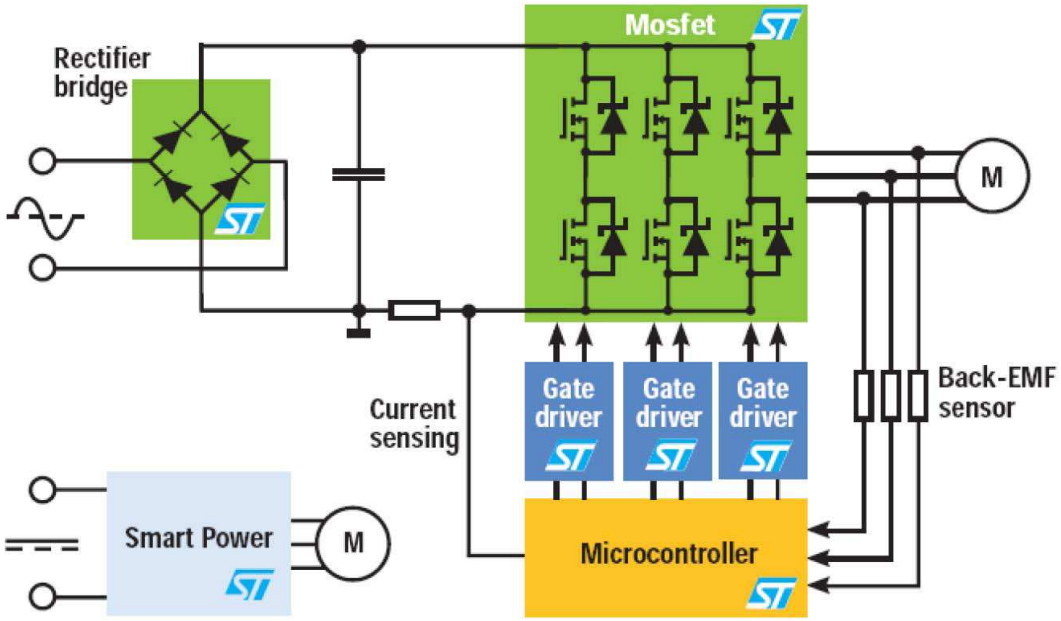
notar que las curvas son idénticas a las del tercer eslabón)



Quinto eslabón



3.5 - Implementación en código VHDL



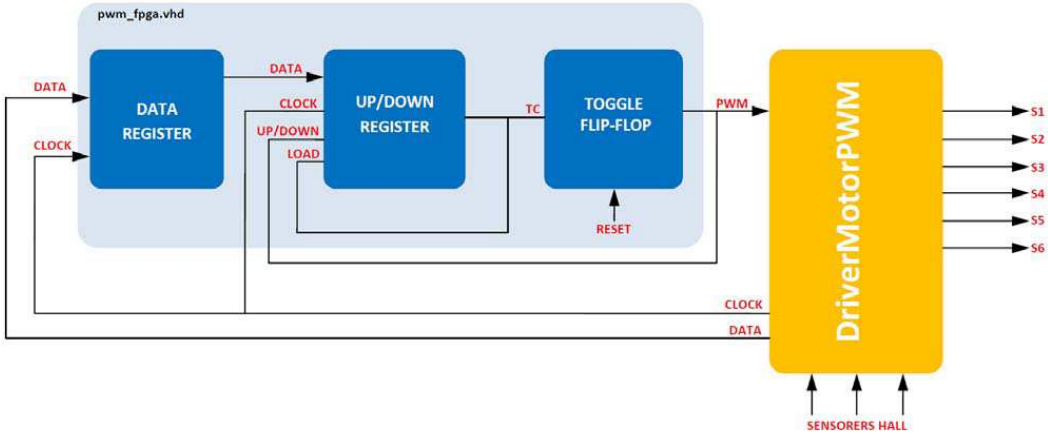


Tabla de Estados

Estado	S1	S2	S3	S4	S5	S6
0	1	0	0	PWM	0	0
1	1	0	0	0	0	PWM
2	0	0	1	0	0	PWM
3	0	PWM	1	0	0	0
4	0	PWM	0	0	1	0
5	0	0	0	PWM	1	0

3.6 - Implementación PWM_FPGA:

```
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                  :instd_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

PROCESS(clock,reg_out,reset)
                  BEGIN
                        IF (reset ='1') THEN
```

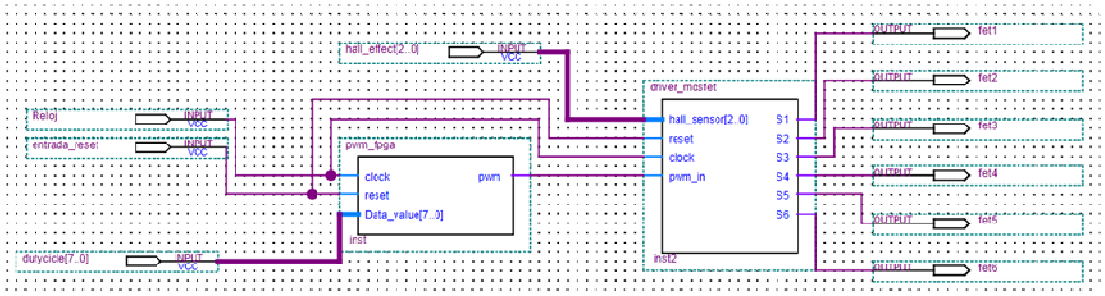
```
                        reg_out<="00000000";
                        ELSIF (rising_edge(clock)) THEN
                                reg_out<= data_value;
                        END IF;
                END PROCESS;
user_pakge library

 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;

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;
        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;
```

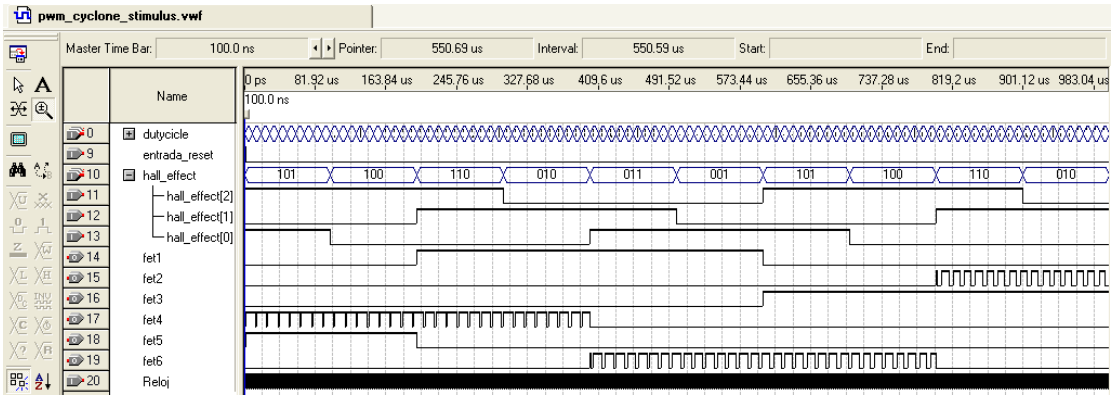
3.7 - Implementación DRIVER MOSFET:

```
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 :instd_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;
```



3.8 - Resultado de simulación de QuartusII



4 - EL COMPILADOR

4.1 - INTRODUCCIÓN AL COMPILADOR

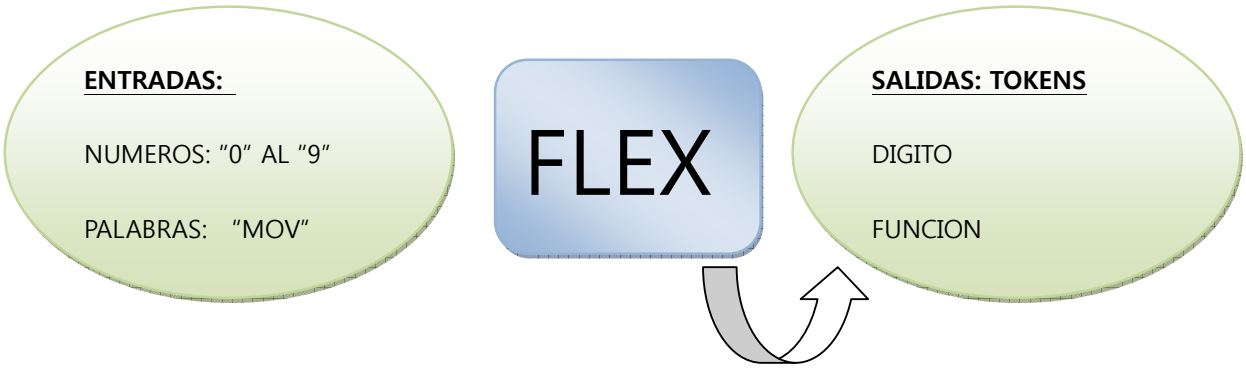
En los apartados anteriores resolvimos tanto el aspecto Cinemática como Dinámico del M5, ahora necesitamos poder manipular el M5 y para ello es necesario programarlo, el inconveniente que surge es que el lenguaje de programación del robot suele ser poco amigable e intuitivo. Para solucionar este inconveniente dedicaremos este último apartado donde el objetivo será el desarrollar un Compilador el cual tiene la función de "unir" el lenguaje de programación del robot con el lenguaje usuario, esto es de extrema utilidad dado que los manipuladores robóticos suelen ser reprogramados frecuentemente y para ello resulta mucho más simple e intuitivo utilizar una lenguaje de usuario y no el lenguaje de programación del robot.

A un compilador tenemos que ingresarle un "Lenguaje propietario" y el compilador nos devolverá un "Código Objeto". En nuestro caso la salida del compilador será la información en pantalla, con esta información se puede generar el código objeto para la FPGA y así generar el código objeto para el M5.

Para desarrollar nuestro compilador haremos uso de dos programas, Felx y Bison, elegimos estos programas ya que son muy difundidos, constan de buena documentación y son programados en C. A continuación explicaremos que función cumple cada uno de ellos para el desarrollo del compilador.

4.2 - Flex:

El Flex (nuestro scanner) es un programa que nos permitirá recibir una serie de palabras "texto" y a través de expresiones regulares las convierte en símbolos o TOKENS, este programa analiza la sintaxis, la hace coincidir con una estructura predefinida y así genera una seria de comandos. En resumen este programa genera el código parser (analizador sintáctico).



En síntesis lo que vamos a hacer con el FLEX es crear los Tokens a partir del texto.

ARCHIVOS FLEX

Las expresiones regulares, Tokens y los archivos se compilan de la siguiente forma:

flex nombrearchivo.l , la extensión .l es proveniente del formato original LEX

La salida del flex es de extensión .c y otro archivo asociado .h.

4.3 - Bison:

El Bison (nuestro parser) toma archivos de extensión .y proveniente del formato original YACC y genera otro archivo ".c" y su archivo asociado ".h" que luego compilaremos con el GCC.

ARCHIVOS BISON

Los archivos de Bison tienen la siguiente forma:

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

Los archivos Bison se generan de la siguiente forma:
En líneas de comendo se tipea "bison -d nombrearchivo.y" lo que genera un archivo .c y su archivo .h asociado.

4.4 - MAKEFILE

El MAKEFILES es un archivo que se utiliza con el comando make de Linux, este comando lo utilizaremos para generar nuestro compilador.
Para compilarlo, nos situamos en la carpeta donde están los archivos y escribimos "make" de esta forma tendremos los archivos compilados según las directivas del makefiles.

```
jae@jae:~/Escritorio/Tesis Robotica$ make
flex *.l
bison -d *.y
gcc -lm -o robotica *.c
jae@jae:~/Escritorio/Tesis Robotica$
```

Codigo del Make:

```
.PHONY: clean
all: flex bison source
flex:	*.l
	flex *.l
bison:	*.y
	bison -d *.y
source:
	gcc -lm -o robotica *.c
clean:
	rm -f *.c *.h *tab*
```

4.5 - COMPILADOR

Nuestro compilador contiene 5 funciones:

moverfase(), *mover2fase()*, *delay()*, *ayuda* y *salir*

moverfase(a1,a2,a3,a4);

Calcula la posición del actuador final en un sistema de coordenadas cuyo origen se encuentra en la base del robot. Para calcular la posición, esta función toma los valores de las cuatro articulaciones del robot (sin tener en cuenta la última articulación y el Grip).

En la captura se muestra un ejemplo de cómo ejecutar la función y la forma en

que esta vuelca los resultados en la pantalla, recordemos que el destino final de estos datos debería ser utilizado para generar el código objeto para la FPGA y así lograr el movimiento del M5.

```
moverfase(10,10,10,10);
 x       y       z
251.19  24.13   92.79
```

mover2fase(ai1,ai2,ai3,ai4,af1,af2,af3,ai4,pasos);

Esta función calcula los puntos por los que pasará el actuador al barrer desde los valores iniciales a los finales y la cantidad de pasos necesarios para ello. Para calcular estos puntos toma dos juegos de valores de las articulaciones y la cantidad de pasos.
La siguiente imagen es la captura donde se ve como se instancia dicha función y como esta muestra los resultados en pantalla.

```
mover2fase(10,10,10,10,30,30,30,30,10);
         q1      q2      q3      q4      -          x       y       z
#0      10.0    10.0    10.0    10.0    -       251.19  24.13   92.79
#1      12.0    12.0    12.0    12.0    -       247.43  29.02   104.39
#2      14.0    14.0    14.0    14.0    -       242.41  33.76   115.59
#3      16.0    16.0    16.0    16.0    -       236.19  38.29   126.36
#4      18.0    18.0    18.0    18.0    -       228.80  42.57   136.68
#5      20.0    20.0    20.0    20.0    -       220.30  46.55   146.53
#6      22.0    22.0    22.0    22.0    -       210.78  50.18   155.88
#7      24.0    24.0    24.0    24.0    -       200.32  53.43   164.72
#8      26.0    26.0    26.0    26.0    -       189.02  56.27   173.05
#9      28.0    28.0    28.0    28.0    -       176.98  58.70   180.85
#10     30.0    30.0    30.0    30.0    -       164.32  60.69   188.12
```

delay(ms);

Esta función genera una demora de tiempo en milisegundos.
También para esta función mostramos como ejecutarla y su información asociada en pantalla.

```
delay(100);

Espera 100 ms...
Fin de espera
```

ayuda;

Esta función se ejecuta con el comando "ayuda" y tiene por finalidad brindar asistencia al usuario, a continuación se detalla su implementación y contenido.

```
jae@jae:~/Escritorio/Tesis Robotica$ ./robotica
ayuda


Posibles funciones:

        moverfase(a1,a2,a3,a4):
                Ingresando 4 angulos de las articulaciones, calcula la posición X Y Z del robot

        mover2fase(ai1,ai2,ai3,ai4,af1,af2,af3,af4,pasos):
                Ingresando 4 angulos iniciales, 4 angulos finales y la cantidad de pasos,
                calcula la trayectoria del extremo final del robot

        delay(ms):
                Retiene la ejecucion del programa durante un lapso de tiempo expresado en ms

        salir:
                Termina el compilador
```

salir;

Esta función nos permite salir del compilador, a continuación mostramos como ejecutarla y su información en pantalla.

```
salir

Finalizacion del compilador, Gracias

jae@jae:~/Escritorio/Tesis Robotica$
```

Captura de pantalla del proceso completo

Este pequeño apartado tiene la finalidad de mostrar la ejecución del proceso completo y co mo se can generando los resultados en la pantalla.

```
jae@jae:~/Escritorio/Tesis Robotica$ make
flex *.l
bison -d *.y
gcc -lm -o robotica *.c
jae@jae:~/Escritorio/Tesis Robotica$
```

```
jae@jae:~/Escritorio/Tesis Robotica$ ./robotica
ayuda


Posibles funciones:

        moverfase(a1,a2,a3,a4):
                Ingresando 4 angulos de las articulaciones, calcula la posición X Y Z del robot

        mover2fase(ai1,ai2,ai3,ai4,af1,af2,af3,af4,pasos):
                Ingresando 4 angulos iniciales, 4 angulos finales y la cantidad de pasos,
                calcula la trayectoria del extremo final del robot

        delay(ms):
                Retiene la ejecucion del programa durante un lapso de tiempo expresado en ms

        salir:
                Termina el compilador

moverfase(10,10,10,10);
 x       y       z
251.19  24.13   92.79

mover2fase(10,10,10,10,30,30,30,30,10);
         q1      q2      q3      q4      -         x       y       z
#0       10.0    10.0    10.0    10.0    -       251.19  24.13   92.79
#1       12.0    12.0    12.0    12.0    -       247.43  29.02   104.39
#2       14.0    14.0    14.0    14.0    -       242.41  33.76   115.59
#3       16.0    16.0    16.0    16.0    -       236.19  38.29   126.36
#4       18.0    18.0    18.0    18.0    -       228.80  42.57   136.68
#5       20.0    20.0    20.0    20.0    -       220.30  46.55   146.53
#6       22.0    22.0    22.0    22.0    -       210.78  50.18   155.88
#7       24.0    24.0    24.0    24.0    -       200.32  53.43   164.72
#8       26.0    26.0    26.0    26.0    -       189.02  56.27   173.05
#9       28.0    28.0    28.0    28.0    -       176.98  58.70   180.85
#10      30.0    30.0    30.0    30.0    -       164.32  60.69   188.12

delay(100);

Espera 100 ms...
Fin de espera

salir

Finalizacion del compilador, Gracias

jae@jae:~/Escritorio/Tesis Robotica$
```

Código para generación del parser y el scanner

Compilador.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

float q1deg,q2deg,q3deg,q4deg;

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 MOVFaseFUNC MOV2FaseFUNC DELAYFUNC AYUDAFUNC SALIRFUNC OPARENTESIS
CPARENTESIS SEMICOLON COMA

%%
commands: /* empty */
```

```
| commands command
;
command: '₩n'
|
moverfase
|
mover2fase
|
delay
|
ayuda
|
salir
;


moverfase:
MOVFaseFUNC OPARENTESIS INTNUM COMA INTNUM COMA INTNUM COMA INTNUM
CPARENTESIS 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)+L
        2*sin(q2)+L1;
        printf(" x ₩t y ₩t z ₩n");
        printf("%.2f₩t%.2f₩t%.2f₩n₩n",x,y,z);
};

mover2fase:
MOV2FaseFUNC OPARENTESIS INTNUM COMA INTNUM COMA INTNUM COMA INTNUM COMA
INTNUM COMA INTNUM COMA INTNUM COMA INTNUM COMA INTNUM CPARENTESIS
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;
	printf("₩t q1 ₩t q2 ₩t q3 ₩t q4 ₩t-₩t   x ₩t   y ₩t   z ₩n");
	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₩t%.1f₩t%.1f₩t%.1f₩t%.1f₩t-
		₩t%.2f₩t%.2f₩t%.2f₩n",i,q1deg+step1deg*i,q2deg+step2deg*i,q3deg+step3deg*
		i,q4deg+step4deg*i,x,y,z);
	}
	printf("₩n");
}

salir:
SALIRFUNC
{
	printf("₩nFinalizacion del compilador, Gracias₩n₩n");
	exit(0);
};

delay:
DELAYFUNC OPARENTESIS INTNUM CPARENTESIS SEMICOLON '₩n'
{
	int sleepTime=$3;
	printf("₩nEspera %d ms...",sleepTime);
```

```
        usleep(sleepTime*1000);
        printf("₩nFin de espera₩n₩n");
};

ayuda:
AYUDAFUNC
{
        printf("₩n₩nPosibles funciones: ₩n₩n₩tmoverfase(a1,a2,a3,a4):₩n₩t₩tIngresando 4 angulos de las articulaciones, calcula la posición X Y Z del robot₩n");
        printf("₩n₩tmover2fase(ai1,ai2,ai3,ai4,af1,af2,af3,af4,pasos):₩n₩t₩tIngresando 4 angulos iniciales, 4 angulos finales y la cantidad de pasos,₩n₩t₩tcalcula la trayectoria del extremo final del robot₩n");
        printf("₩n₩tdelay(ms):₩n₩t₩tRetiene la ejecucion del programa durante un lapso de tiempo expresado en ms₩n");
        printf("₩n₩tsalir:₩n₩t₩tTermina el compilador₩n₩n");
};
%%
```

Compilador.l

```
%{
#include <stdio.h>
#include "compilador.tab.h"
extern YYSTYPE yylval;
%}
digit [0-9]
%%
-?{digit}+ {
        yylval.entero=atoi(yytext);
        return(INTNUM);
}

-?{digit}+"."{digit}* {
        yylval.flotante=atof(yytext);
        return(FLOATNUM);
};

moverfase return MOVFaseFUNC;
```

```
mover2fase return MOV2FaseFUNC;
delay return DELAYFUNC;
ayuda return AYUDAFUNC;
salir return SALIRFUNC;
"(" return OPARENTESIS;
")" return CPARENTESIS;
";" return SEMICOLON;
"," return COMA;
"₩n" { return (yytext[0]);}
[ ₩t]+ /* ignore whitespace */;
```

Conclusiones finales:

Del análisis cinemática pudimos verificar que por medio del método Denavit-Hartenberg expresado en Matlab y luego codificada para el DSP56800/E en CodeWarrior que llegamos a los movimientos que esperábamos tener.
Los problemas que tuvimos además de entender bien el alcance de nuestro objetivo era la codificación que se debía hacer en C para el DSP56800/EdeCodeWarrior. Ya que al realizar los cálculos en Matlab (double) tenía que codificar teniendo en cuenta el punto fijo del DSP. Sin que haya desborde en los cálculos y redimensionando las medidas.
También, los problemas que tuvimos que enfrentar son las ecuaciones, que al ser muy largas, cometíamos mucho error. Teníamos la opción de multiplicar matricialmente por medio de una función de CodeWarrior, pero ya habíamos avanzado mucho con el método de resolver la ecuación para cada eje. Por eso optamos resolver de esta manera.
El movimiento que estamos haciendo es, girar la base 360º por medio de la 'q1' y también girar el grid'q5' que no aporta en los cambios de la posición en el plano xyz. La 'q3' y 'q4' se incrementa con menor paso que la base. Este movimiento nos debería dar la sensación de un espiral que incrementa el diámetro y luego empieza a achicarse.
Este movimiento se verifica por medio del grafico en Matlab.

Del análisis dinámico realizado obtuvimos los resultados de distintos componentes de velocidad, aceleración, torque, coriolis, inercia y gravedad partiendo de una trayectoria sinusoidal. Graficando las mismas podemos observar que las correspondientes al tercer y cuarto eslabón son idénticas, esto es así porque el cuarto eslabón es fue agregado para cumplir con las condiciones de D-H.
Respecto al torque podemos decir que a lo largo de una trayectoria, en nuestro caso sinusoidal, vemos que hay excesos de esfuerzos del motor por las pendientes abruptas del torque. Esto puede ser por errores en los cálculos de

masas y fuerzas que no se están teniendo en cuenta. Otra causa de estas pendientes abruptas puede ser la elección de una trayectoria no adecuada. Pudiendo ensayarse con otras trayectorias que no sobre exijan al motor.
La implementación de PWM se hizo sobre Quartus en VHDL, donde se logro el control del puente trifásico. Esos trenes de pulsos pasan por un driver para controlar las conmutaciones del FETs. Estos FETs se conmutan alimentados con cada fase del motor, es ahí donde hay que tener en cuenta que en los cambios de conmutación debe tener un tiempo muerto entre la conmutación de cada FET para evitar un cortocircuito.
Como última etapa de este trabajo desarrollamos un compilador para poder manipular nuestro robot, para esto decidimos apoyar nuestro desarrollo en dos herramientas de uso libre tales como FLEX y BISON los cuales corren bajo LINUX siguiendo de esta forma con la línea de herramientas libres, a nuestro compilador lo datamos con solo un grupo de funciones básicas, las que nos permitieron investigar, aprender y luego plasmar en forma práctica los temas básicos que forman parte de la implementación de un compilador.
Hoy, habiendo transcurrido tanto los conceptos teóricos y prácticos de este informe, y mirando hacia atrás, recordamos con gozo los obstáculos superados y tomamos noción de la diversidad de especialidades con la que se nutre la robótica permitiendo así realizar un trabajo multidisciplinario, en el marco de las incumbencias de un Ingeniero Electrónico.

Desde ya muchas gracias, esperamos que este informe sea una buena base para los colegas que comiencen a incursionar en este maravilloso mundo y que aquí puedan encontrar el puntapié inicial de muchos temas.

Autores : **Lee JoonJae,**
Sebastián Papandrea

Índice