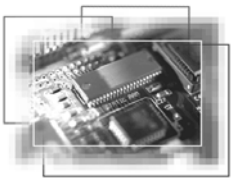


UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
INGENIERIA ELECTRÓNICA

TRABAJO PRÁCTICO #2

Análisis dinámico + PWM Control en VHDL

ROBÓTICA

INTEGRANTES:

PEREZ, Julián Gabriel [LEG : 1203307]
DE ANGELIS, Fernando [LEG: 1190842]

Profesor: **GIANNETTA,** Hernán
Jefe de TP**: GRANZELLA,** Damián
Fecha de Entrega: 12 -07-2010
Curso: R5055
Año: 2010

TRABAJO PRÁCTICO N°2

Análisis dinámico + PWM Control en VHDL

OBJETIVO

- Introducción sobre dinámica del robot.
- Análisis dinámico de la estructura mecánica del "One Legged Robot".
- Implementación en código VHDL.
- Resultados de la simulación.
- Conclusiones finales.

INTRODUCCIÓN SOBRE LA DINÁMICA DEL ROBOT

En el trabajo práctico anterior se ha considerado la geometría de los robots, así como el movimiento sin analizar las fuerzas que lo producen. Como se sabe, las velocidades lineales y angulares vienen dadas por las fuerzas y pares que se aplican a la estructura mecánica y dependen también de las magnitudes de las masas y su distribución. Las relaciones involucradas constituyen el modelo dinámico del manipulador.

Este modelo, establece la relación matemática entre:

- 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.

La obtención de este modelo para mecanismos de uno o dos grados de libertad no es excesivamente compleja, pero a medida que el número de grados de libertad aumenta, el planteamiento y obtención del modelo dinámico se complica enormemente. Por este motivo no siempre es posible obtener un modelo dinámico expresado de una forma cerrada, esto es, mediante una serie de ecuaciones, normalmente de tipo diferencial de segundo orden, cuya integración permita conocer que movimiento surge al aplicar unas fuerzas o qué fuerzas hay que aplicar para obtener un movimiento determinado. El modelo dinámico debe ser resuelto entonces de manera iterativa mediante la utilización de un procedimiento numérico.

Es por esto que en este trabajo práctico se hará hincapié en la utilización de herramientas informáticas, tales como ToolKit HEMERO para Matlab. Este calcula el modelo dinámico completo de forma analítica empleando para ello el método de Newton-Euler recurrente.

A modo introductorio diremos que la formulación de Newton-Euler utiliza las ecuaciones de equilibrio de fuerzas y torques.

$$\sum F = m\dot{v}$$

$$\sum T = I\dot{\omega} + \omega \times (I\omega)$$

Un adecuado desarrollo de estas ecuaciones conduce a una formulación recursiva en la que se obtienen la posición, velocidad y aceleración del eslabón i referidos 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.

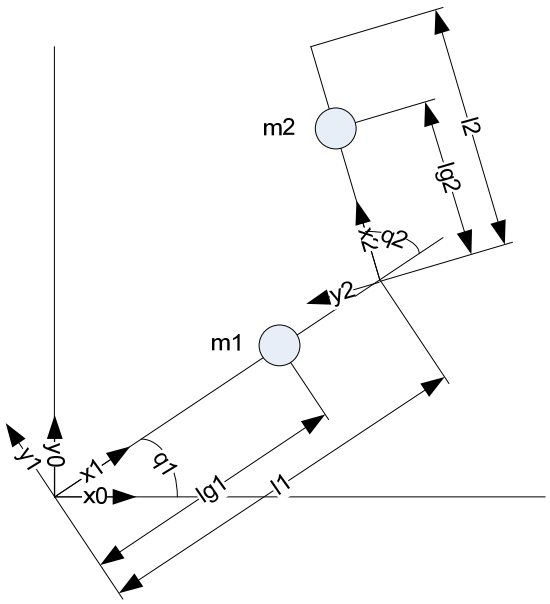
El algoritmo se basa en operaciones vectoriales (con productos escalares y vectoriales entre magnitudes vectoriales, y productos de matrices con vectores) siendo más eficiente en comparación con las operaciones matriciales asociadas a la formulación Lagrangiana.

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.

ANÁLISIS DINAMICO DE LA ESTRUCTURA MECÁNICA "ONE LEGGED ROBOT"

Para la realización del trabajo práctico 2, decidimos basarnos en la utilización del ToolKit denominado HEMERO (HErramienta Matlab-simulink para la Enseñanza de la RObótica).
Esta herramienta, puede ser empleada tanto para los manipuladores robóticas como para los robots móviles y está compuesta por funciones MATLAB y por bloques de Simulink que se han definido para facilitar la simulaciones de los modelos cinemáticos y dinámicos.
Para el estudio del robot de una sola pierna (basado en el paper "Study on one-legged robot jumping") utilizamos el siguiente modelo.

MODELO DE TRABAJO



La asignación de los marcos de referencia se muestra en la figura. Los ejes z0, z1 y z2 son paralelos y en la misma dirección que los ejes de las tres articulaciones apuntando hacia fuera (salientes de la hoja). Por consiguiente, los parámetros di y los αi son todos nulos.
En la siguiente tabla se indican los parámetros de Denavit-Hartenberg resultado de la asignación de los sistemas de referencia que se muestran en la figura.

Articulación	θi	di	ai	αi
1	q1	0	0	0
2	q2	0	l1	0
3	q3	0	l2	0

Cada vez que se quiera utilizar una función de HEMERO relacionada con la cinemática se deberán introducir en una matriz los parámetros de Denavit-Hartenberg del manipulador, de acuerdo con la notación de Craig (1986). El modo de introducir dicha información en esa matriz es el siguiente:
- Habrá una fila por cada enlace que tenga el manipulador.
- Cada fila tendrá el siguiente formato: [alpha(i-1) a(i-1) theta(i) d(i) sigma(i)]
donde:
- alpha(i-1), a(i-1), theta(i), d(i) son los parámetros de D-H según Craig (1986).
- sigma(i) indicará el tipo de articulación (será 0 si es de rotación y un número distinto de cero si por el contrario es prismática).
Así pues para un robot con n enlaces, la matriz sería de dimensiones nx5.
Todos los ángulos deberán ser introducidos en radianes. Las longitudes a(i-1) y d(i) podrán ser expresadas en cualquier unidad y sólo habrá que tener cuidado de recordar que las transformaciones homogéneas y los Jacobianos que se calculen aparecerán en esas mismas unidades.
Quedando entonces la matriz dh en Matlab:

Articulación	α(i-1)	a(i-1)	θ(i)	d(i)	Σ(i)
1	0	0	q1	0	0
2	0	l1	q2	0	0
3	0	l2	q3	0	0

Además, cada vez que se quiera utilizar una función de HEMERO relacionada con la dinámica será necesario introducir en una matriz los parámetros de D-H del manipulador, junto con ciertos parámetros dinámicos. El modo de introducir esta información en dicha matriz (a la que se denominará genéricamente DYN) es el siguiente:
Habrá una fila por cada enlace que tenga el manipulador y cada fila tendrá el siguiente formato:

```
1      alpha(i-1)     Parámetros de Denavit-Hartenberg
2      a(i-1)
3      theta(i)
4      d(i)
5      sigma(i)       Tipo de articulación, 0 si es de rotación y 1 si es prismática
6      masa           Masa del enlace i
7      rx             Centro de masas del enlace respecto al cuadro de referencia de
dicho enlace
8      ry
9      rz
10     Ixx     Elementos del tensor de inercia referido al centro de masas del enlace
11     Iyy
12     Izz
13     Ixy
14     Iyz
15     Ixz
16     Jm      Inercia de la armadura
17     G       Velocidad de la articulación / velocidad del enlace
18     B       Fricción viscosa, referida al motor
19     Tc+     Fricción de Coulomb (rotación positiva), referida al motor
20     Tc-     Fricción de Coulomb (rotación negativa), referida al motor
```

Así pues para un robot con n enlaces, la matriz DYN tendría dimensiones nx20. Todos los ángulos deberán ser introducidos en radianes. El resto de parámetros de la matriz podrán tener las unidades que se deseen, siempre que se sea coherente en el uso de dichas unidades. Es decir que si se introducen las masas en Kg y los centros de masas en metros, al escribir el tensor de inercia se deberá expresar en Kg m2.
La función principal para el análisis dinámico será:

`rne(dyn, q, qd, qdd)`

Que calcula el modelo dinámico completo de forma analítica empleando para ello el método de Newton-Euler recurrente, tal como se describe en el capítulo 5 del libro de Olleros
Adicionalmente, para la utilización de la misma, es necesario pasarle un vector con la aceleración de la gravedad que sufre el manipulador, así como los valores de las variables articulares y de las velocidades y aceleraciones articulares. La función rne devuelve como resultado los pares ejercidos en cada articulación.
Para la gráfica del robot, se utilizó otra función del toolbox:

`plotbot(dh, q)`

Esta función construye una representación gráfica del robot, a partir de los parámetros cinemáticos contenidos en DH y de los valores de las variables articulares (Q) que se le pasen.

Suposiciones de cálculo

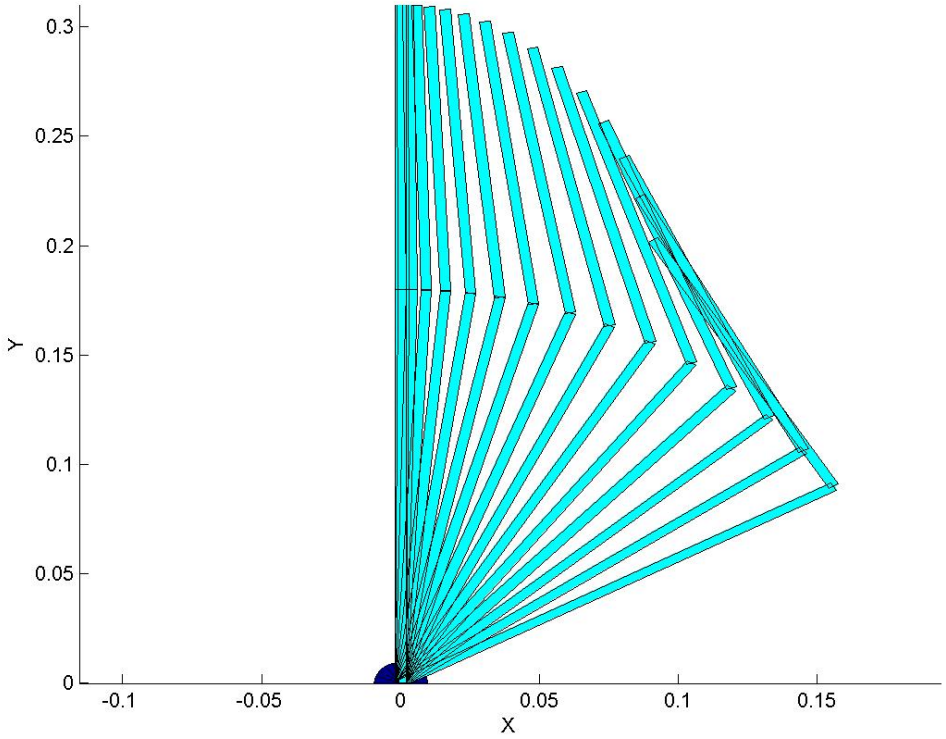
Para comenzar a plantear el análisis, supusimos un movimiento similar a el de una pierna humana, en el cual, desde una posición flexionada se comienza a enderezar la pierna rápidamente para lograr impulsar el cuerpo hacia arriba.

Con esto en mente se diseñó una función que comience en un ángulo (qix) y termine en otro ángulo (qfx), haciéndolo en un tiempo determinado (ttransx), y según una función seno (ya que es lo más simple y según creímos, similar al movimiento de una pierna).

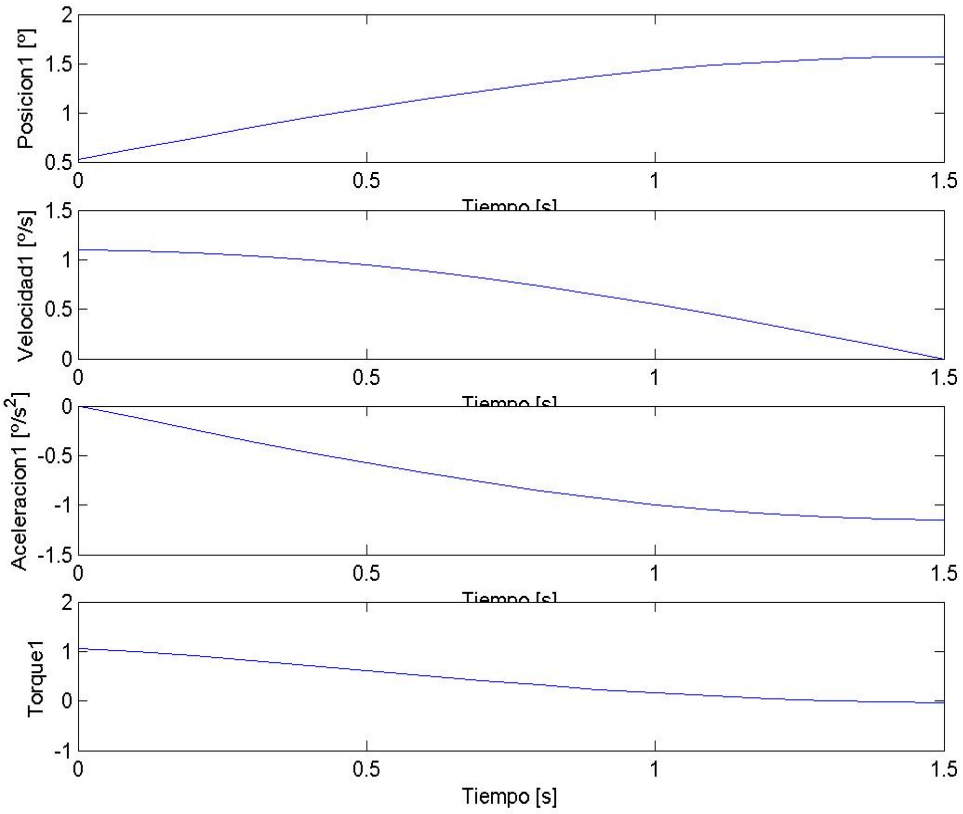
Esta misma función se implementa para ambos motores, pasando a las funciones del toolbox HEMERO los datos de posición, velocidad y aceleración angular. Obteniéndose como resultado el torque de cada motor.

El proceso se repite desde un tiempo inicial (t1) a un tiempo final (t2), de a pasos determinados (pasot), pudiendo graficarse cada una de los valores intermedios.

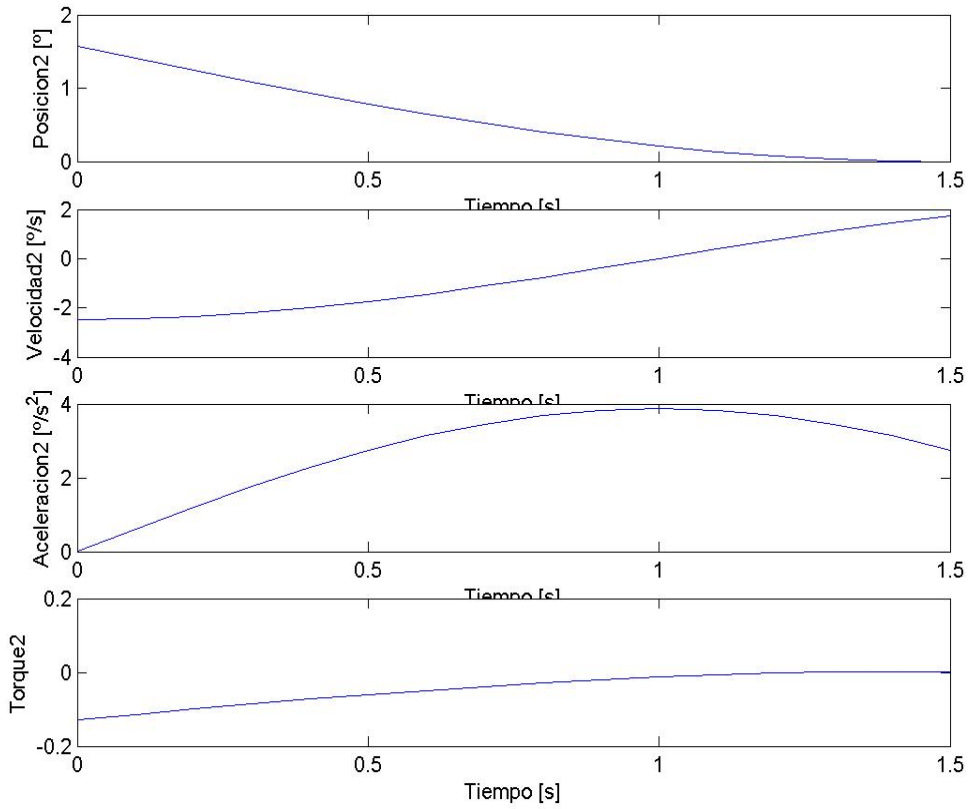
SIMULACIÓN DE LA PIERNA EN MOVIMIENTO



GRÁFICA DE POSICIÓN, VELOCIDAD, ACELERACION ANGULAR Y TORQUE DEL MOTOR 1 (HEMERO)

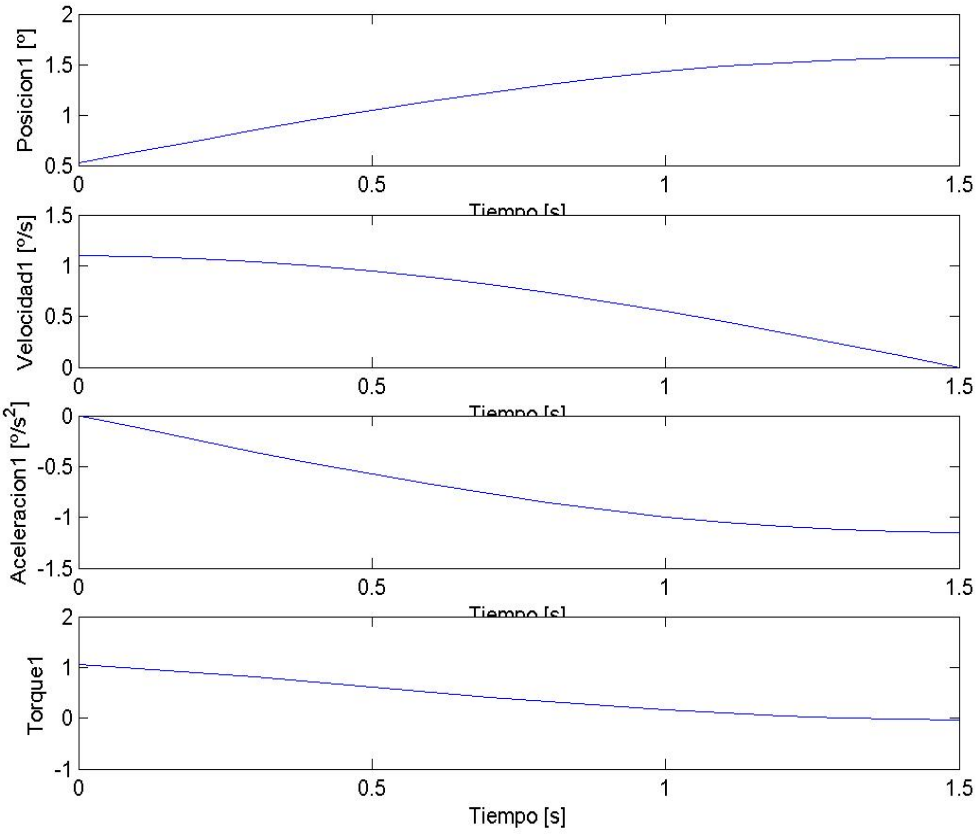


GRÁFICA DE POSICIÓN, VELOCIDAD, ACELERACION ANGULAR Y TORQUE DEL MOTOR 2 (HEMERO)

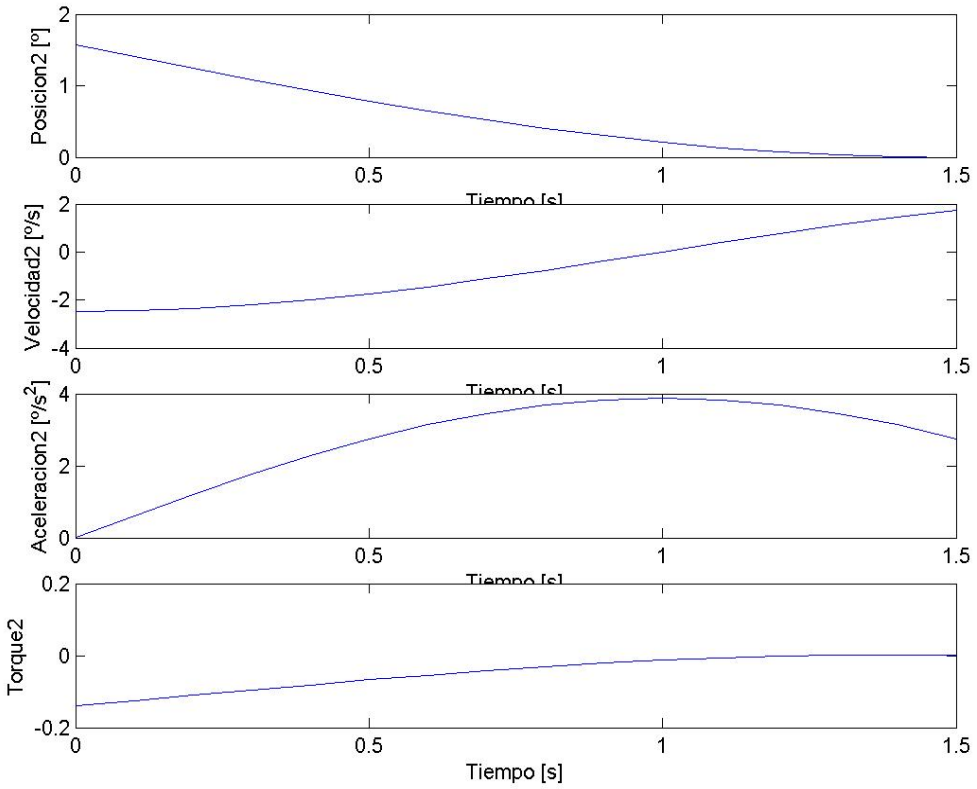


En una segunda parte del estudio, decidimos implementar las matrices del paper “Study on one-legged robot jumping” pasándole los mismos datos de posición, velocidad y aceleración angular. De esta forma esperamos conseguir contrastar los resultados de torque de ambos motores para validar, por un lado, las matrices resultado del análisis del paper y, por el otro, el ToolKit HEMERO.
Los resultados son los siguientes.

GRÁFICA DE POSICIÓN, VELOCIDAD, ACELERACION ANGULAR Y TORQUE DEL MOTOR 1 (PAPER)

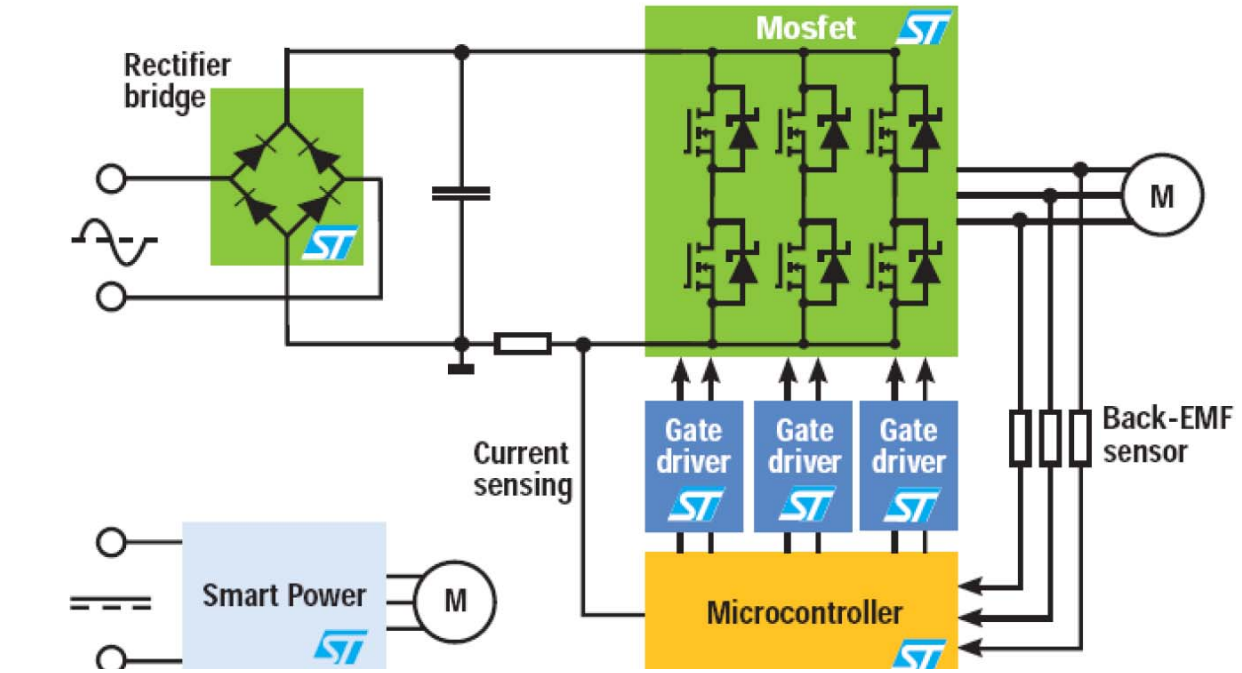


GRÁFICA DE POSICIÓN, VELOCIDAD, ACELERACION ANGULAR Y TORQUE DEL MOTOR 2 (PAPER)



IMPLEMENTACIÓN EN CODIGO VHDL

ESQUEMA DEL CONTROL



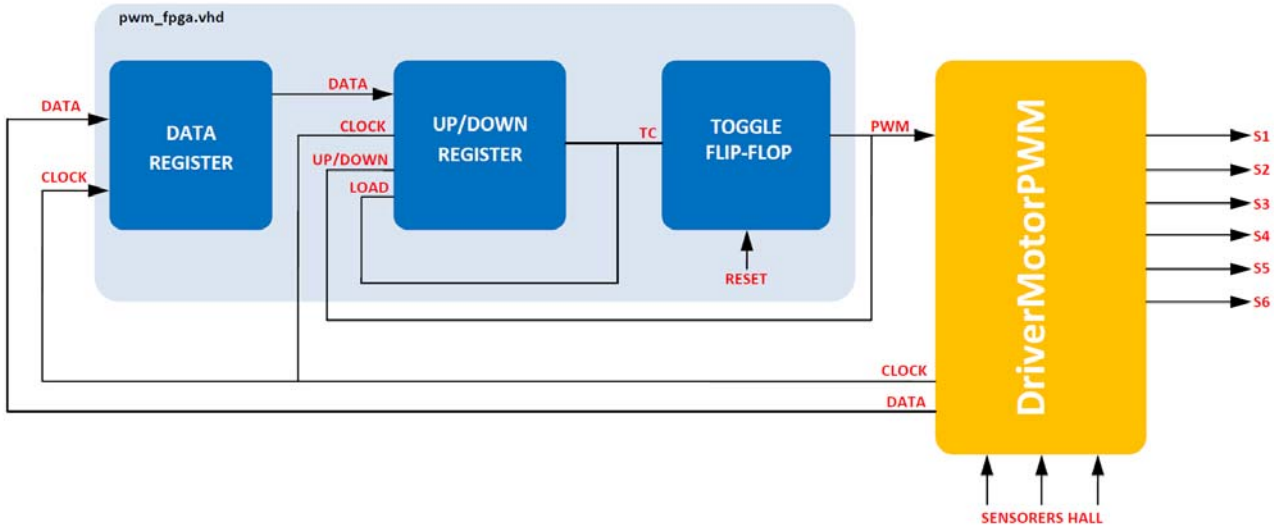


TABLA DE ESTADOS

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

CÓDIGO EN VHDL DE DRIVER MOTOR PWM

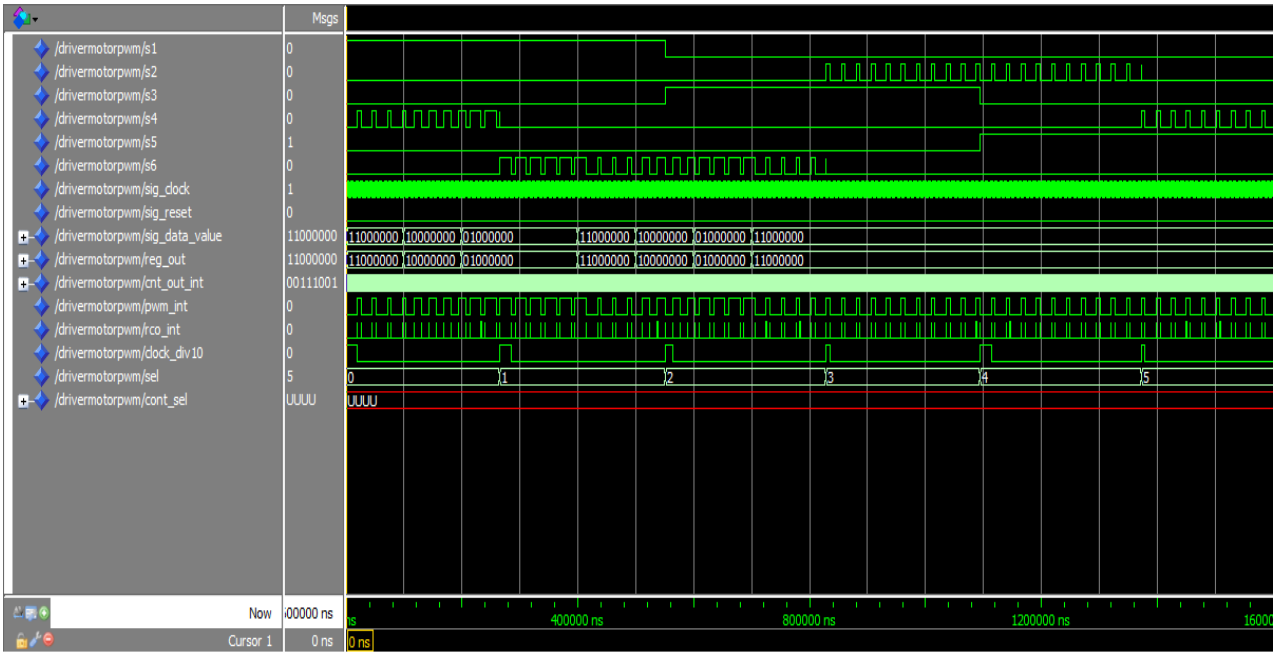
```
library IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all ;
USE    work.user_pkg.all;

ENTITY DriverMotorPWM 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
     );

END DriverMotorPWM;

ARCHITECTURE arch_DriverMotorPWM OF DriverMotorPWM IS

--SEÑALES INTERNAS
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;
shared variable ENDSIM: boolean:=false;
constant clk_period:TIME:=100ns;

SIGNAL reg_out            : std_logic_vector(7 downto 0);
SIGNAL cnt_out_int        : std_logic_vector(7 downto 0);
SIGNAL pwm_int, rco_int, clock_div10 : STD_LOGIC;
SIGNAL sel                : integer range 0 to 5;
SIGNAL cont_sel           : std_logic_vector(3 downto 0);

BEGIN

--GENERACION DE LA SEÑAL DE CLOCK
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;

--GENERACION DE 3 VELOCIDADES DIFERENTES PARA EL CONTROL PWM
stimulus_process: PROCESS
  BEGIN
           sig_reset <= '1';
           wait for 100 ns;
           sig_reset <= '0';
           sig_data_value <= "11000000";
           wait for 100 us;
           sig_data_value <= "10000000";
           wait for 100 us;
           sig_data_value <= "01000000";
           wait for 100 us;
           sig_data_value <= "01000000";
           wait for 100 us;
           sig_data_value <= "11000000";
           wait for 100 us;
           sig_data_value <= "10000000";
           wait for 100 us;
           sig_data_value <= "01000000";
```

```
            wait for 100 us;
            sig_data_value <= "11000000";
            wait for 100 us;
       wait;

 END PROCESS stimulus_process;

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

 PROCESS(sig_clock,reg_out,sig_reset)
             BEGIN
                   IF (sig_reset ='1') THEN
                                reg_out <="00000000";
                   ELSIF (rising_edge(sig_clock)) THEN
                                reg_out <= sig_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_pakage library

 PROCESS (sig_clock, cnt_out_int, rco_int, reg_out)

       BEGIN
        IF (rco_int = '1') THEN
            cnt_out_int <= reg_out;
        ELSIF  rising_edge(sig_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, sig_clock, sig_reset)
       BEGIN
      IF (sig_reset ='1') THEN
             rco_int <='1';
          ELSIF rising_edge(sig_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 (sig_clock, rco_int, sig_reset)
       BEGIN
           IF (sig_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;


--GENERACION DE UNA SEÑAL DE FRECUENCIA 10 VECES MENOR AL CLOCK
Gen_clk10: PROCESS (sig_reset, rco_int) IS
VARIABLE cuenta: INTEGER range 0 to 20;

BEGIN
	IF (sig_reset = '1') THEN
		cuenta:= 0;
		clock_div10 <= '1';
	ELSE
		IF rising_edge(rco_int) THEN
			IF (cuenta = 20) THEN
		cuenta:= 0;
			clock_div10 <= '1';
			ELSE
			cuenta:= cuenta + 1;
			clock_div10 <= '0';
			END IF;
		END IF;
	END IF;

END PROCESS;


GenVarState: PROCESS (sig_reset, clock_div10) IS
VARIABLE cuenta: INTEGER range 0 to 5;
BEGIN
	IF (sig_reset = '1') THEN
		cuenta:= 0;
	ELSE
		IF rising_edge (clock_div10) THEN
			IF (cuenta = 5) THEN
			cuenta:= 0;
			ELSE
			cuenta:= cuenta + 1;
			END IF;
		END IF;
	END IF;

sel <= cuenta;

END PROCESS;

--CONTROL DE LOS DEL SERVOMOTOR
Cntrl_PWM_puenteH: PROCESS (sig_reset, sel, pwm_int) IS
BEGIN
	IF (sig_reset = '1') THEN
		S1<= '0'; S2<= '0'; S3<= '0'; S4<= '0'; S5<= '0'; S6<= '0';
	ELSE
		CASE sel IS
			WHEN 0 =>			--ESTADO #1
				S1<= '1';
				S2<= '0';
				S3<= '0';
				S4<= pwm_int;
				S5<= '0';
				S6<= '0';

			WHEN 1 =>			--ESTADO #2
				S1<= '1';
				S2<= '0';
				S3<= '0';
				S4<= '0';
				S5<= '0';
				S6<= pwm_int;

			WHEN 2 =>			--ESTADO #3
				S1<= '0';
				S2<= '0';
```

```
                S3<= '1';
                S4<= '0';
                S5<= '0';
                S6<= pwm_int;

            WHEN 3 =>                   --ESTADO #4
                S1<= '0';
                S2<= pwm_int;
                S3<= '1';
                S4<= '0';
                S5<= '0';
                S6<= '0';

            WHEN 4 =>                   --ESTADO #5
                S1<= '0';
                S2<= pwm_int;
                S3<= '0';
                S4<= '0';
                S5<= '1';
                S6<= '0';

            WHEN 5 =>                   --ESTADO #6
                S1<= '0';
                S2<= '0';
                S3<= '0';
                S4<= pwm_int;
                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_DriverMotorPWM;
```

RESULTADO DE LA SIMULACIÓN EN MODELSIM



ANEXO 1 – CÓDIGO PROGRMA SIMULACIÓN EN MATLAB (TP2-HEMERO.M)

```
disp('TP2 Utilizando ToolBox Hemero')

clear all

m1=0.465;
m2=0.450;
m3=0.010;
l1=0.18;
l2=0.13;
lg1=(0.7037*l1);
lg2=(0.444*l2);
lg3=0.01;
i1=(m1*(lg1^2));
i2=(m2*(lg2^2));
g=9.8;

syms x1 x2

qi1=deg2rad(30);                    %angulo inicial
qf1=deg2rad(90);                    %angulo final
ttrans1=1;                          %tiempo de transicion entre angulo
inicial y final
dospif1=abs((qf1-qi1)/ttrans1);
qi2=deg2rad(90);                    %angulo inicial
qf2=deg2rad(0);                     %angulo final
ttrans2=1;                          %tiempo de transicion entre angulo
inicial y final
dospif2=abs((qf2-qi2)/ttrans2);

t1=f1(qi1, (qf1-qi1), dospif1, x1);     %Funcion posicion motor1
td1=inline(diff(t1,x1));                %Funcion velocidad motor1 (deribada de
la posicion)
tdd1=inline(diff(t1,x1,2));              %Funcion aceleracion motor1 (deribada
de la velocidad)
t2=f2(qi2, (qf2-qi2), dospif2, x2);     %Funcion posicion motor2
td2=inline(diff(t2,x2));                %Funcion velocidad motor2 (deribada de
la posicion)
tdd2=inline(diff(t2,x2,2));             %Funcion aceleracion motor2 (deribada
de la velocidad)

figure(5);
hold on;

t1=0;                           %tiempo inicial
t2=12;                          %tiempo final
pasot=0.1;

indice=0;

for i=t1:pasot:t2

indice=indice+1;

q1=feval('f1',qi1, (qf1-qi1), dospif1, i);
q2=feval('f2',qi2, (qf2-qi2), dospif1, i);
q3=0;

dh=[   0   0   q1  0   0;
       0   l1  q2  0   0;
       0   l2  q3  0   0];

dyn=[0  0  q1  0  0  m1 lg1  0  0  i1  0  0  0  0  0  i1  1  0  0  0;
     0  l1 q2  0  0  m2 lg2  0  0  i2  0  0  0  0  0  i2  1  0  0  0;
     0  l2 q3  0  0  m3 lg3  0  0  0   0  0  0  0  0  0   1  0  0  0];
```

```
q = [q1 q2 q3];
qd = [feval(td1,i) feval(td2,i) q3];
qdd = [feval(tdd1,i) feval(tdd2,i) q3];
grav = [0 -g 0];

plotbot(dh, q, 'd');

tau = rne(dyn, q, qd, qdd, grav);

tau1(indice)=tau(1);
tau2(indice)=tau(2);
pos1(indice)=q(1);
pos2(indice)=q(2);
vel1(indice)=qd(1);
vel2(indice)=qd(2);
ace1(indice)=qdd(1);
ace2(indice)=qdd(2);
index(indice)=i;

end

figure(1);
grid on;

subplot(4,1,1);
plot(index, pos1);              %grafica posicion motor1
xlabel('Tiempo [s]');
ylabel('Posicion1 [º]');

subplot(4,1,2);
plot(index, vel1);              %grafica velocidad motor1
xlabel('Tiempo [s]');
ylabel('Velocidad1 [º/s]');

subplot(4,1,3);
plot(index, ace1);              %grafica aceleracion motor1
xlabel('Tiempo [s]');
ylabel('Aceleracion1 [º/s^2]');

subplot(4,1,4);
plot(index, tau1);              %grafica torque motor1
xlabel('Tiempo [s]');
ylabel('Torque1');

figure(2);
grid on;

subplot(4,1,1);
plot(index, pos2);              %grafica posicion motor2
xlabel('Tiempo [s]');
ylabel('Posicion2 [º]');

subplot(4,1,2);
plot(index, vel2);              %grafica velocidad motor2
xlabel('Tiempo [s]');
ylabel('Velocidad2 [º/s]');

subplot(4,1,3);
plot(index, ace2);              %grafica aceleracion motor2
xlabel('Tiempo [s]');
ylabel('Aceleracion2 [º/s^2]');

subplot(4,1,4);
plot(index, tau2);              %grafica torque motor2
xlabel('Tiempo [s]');
ylabel('Torque2');
```

ANEXO 2 – CÓDIGO PROGRMA SIMULACIÓN EN MATLAB (TP2-PAPER.M)

```
disp('TP2 Utilizando Matriz paper "One Legged Robot"')

clear all

m1=0.465;
m2=0.450;
l1=0.18;
l2=0.13;
lg1=(0.7037*l1);
lg2=(0.444*l2);
i1=m1*(lg1^2);
i2=m2*(lg2^2);
g=9.8;

syms x1 x2

qi1=deg2rad(30);                    %angulo inicial
qf1=deg2rad(90);                    %angulo final
ttrans1=1;                          %tiempo de transicion entre angulo
inicial y final
dospif1=abs((qf1-qi1)/ttrans1);
qi2=deg2rad(90);                    %angulo inicial
qf2=deg2rad(0);                     %angulo final
ttrans2=1;                          %tiempo de transicion entre angulo
inicial y final
dospif2=abs((qf2-qi2)/ttrans2);

t1=f1(qi1, (qf1-qi1), dospif1, x1);     %Funcion posicion motor1
td1=inline(diff(t1,x1));                %Funcion velocidad motor1 (deribada de
la posicion)
tdd1=inline(diff(t1,x1,2));             %Funcion aceleracion motor1 (deribada
de la velocidad)
t2=f2(qi2, (qf2-qi2), dospif2, x2);     %Funcion posicion motor2
td2=inline(diff(t2,x2));                %Funcion velocidad motor2 (deribada de
la posicion)
tdd2=inline(diff(t2,x2,2));             %Funcion aceleracion motor2 (deribada
de la velocidad)

t1=0;                           %tiempo inicial
t2=12;                          %tiempo final
pasot=0.1;

indice=0;

for i=t1:pasot:t2

indice=indice+1;

q1=feval('f1',qi1, (qf1-qi1), dospif1, i);
q2=feval('f2',qi2, (qf2-qi2), dospif1, i);
qd1=feval(td1,i);
qd2=feval(td2,i);
qdd1=feval(tdd1,i);
qdd2=feval(tdd2,i);

q = [q1; q2];
qd = [qd1; qd2];
qdd = [qdd1; qdd2];

s1=sin(q1);
s2=sin(q2);
s12=sin(q1+q2);
c1=cos(q1);
c2=cos(q2);
c12=cos(q1+q2);

M=[(m1*(lg1^2)+i1+m2*((l1^2)+(lg2^2)+2*l1*lg2*c2)+i2)
(m2*((lg2^2)+(l1*lg2*c2))+i2); (m2*((lg2^2)+l1*lg2*c2)+i2) (m2*(lg2^2)+i2)];
```

```
C=[(-2*m2*l1*lg2*s2*qd1*qd2-m2*l1*lg2*s2*(qd2^2));
(m2*((lg2^2)+l1*lg2*qd1*qd2))];
G=[((m1*g*lg1*c1)+m2*g*(l1*c1+lg2*c12)); (m2*g*lg2*c12)];
;J=[(-l1*s1-l2*s12) (-l2*s12); (l1*c1+l2*c12) (l2*c12)];

tau=M*qdd+C+G;

tau1(indice)=tau(1);
tau2(indice)=tau(2);
pos1(indice)=q(1);
pos2(indice)=q(2);
vel1(indice)=qd(1);
vel2(indice)=qd(2);
ace1(indice)=qdd(1);
ace2(indice)=qdd(2);
index(indice)=i;

end

figure(3);
grid on;

subplot(4,1,1);
plot(index, pos1);              %grafica posicion motor1
xlabel('Tiempo [s]');
ylabel('Posicion1 [º]');

subplot(4,1,2);
plot(index, vel1);              %grafica velocidad motor1
xlabel('Tiempo [s]');
ylabel('Velocidad1 [º/s]');

subplot(4,1,3);
plot(index, ace1);              %grafica aceleracion motor1
xlabel('Tiempo [s]');
ylabel('Aceleracion1 [º/s^2]');

subplot(4,1,4);
plot(index, tau1);              %grafica torque motor1
xlabel('Tiempo [s]');
ylabel('Torque1');

figure(4);
grid on;

subplot(4,1,1);
plot(index, pos2);              %grafica posicion motor2
xlabel('Tiempo [s]');
ylabel('Posicion2 [º]');

subplot(4,1,2);
plot(index, vel2);              %grafica velocidad motor2
xlabel('Tiempo [s]');
ylabel('Velocidad2 [º/s]');

subplot(4,1,3);
plot(index, ace2);              %grafica aceleracion motor2
xlabel('Tiempo [s]');
ylabel('Aceleracion2 [º/s^2]');

subplot(4,1,4);
plot(index, tau2);              %grafica torque motor2
xlabel('Tiempo [s]');
ylabel('Torque2');
```

CONCLUSIONES

Gracias a la utilización del ToolKit HEMERO y las herramientas de simulación del Matlab se obtuvieron una graficas finales en las cuales puede observarse con claridad la evolución del torque de los motores en función de la posición, velocidad y aceleración de las articulaciones.
En estas se revela que el máximo torque para el motor 1 es de aproximadamente 1Kg.m, y para el motor 2 de aproximadamente 0,2Kg.m.
Se evidencia también, como es de esperarse, que el máximo torque se de en el comienzo del movimiento, ya que debe vencer la inercia y es la posición del recorrido fijado donde la gravedad tiene mayor influencia.
Debe tenerse en cuenta que los torques obtenidos son los finales, o sea, los necesarios en la articulación en si. De manera que con la utilización de unidades de reducción, el torque necesario seria inferior, dependiendo de esta. En estas aplicaciones, donde los requerimientos de velocidad final no son importantes, esto se hace más factible ya que se intercambia velocidad por torque.
De la comparación de ambos métodos, puede notarse que los torques son iguales. Lo que nos lleva a pensar que ambos métodos son correctos y la utilización del ToolKit HEMERO es la adecuada. Esto nos brinda cierto grado de confianza en la utilización del ToolKit HEMERO, el cual nos permite agilizar los resultados y obtener una simulación que nos brinda una visión general del asunto.

En lo que respecta a la simulación en VHDL, a pesar de no tener una buena base teórica sobre la programación en este lenguaje, esta tarea no fue dificultosa, sin embargo, en lo que respecta a nuestro grupo, el uso del programa MODELSIM fue complicado, debido a ser la primera vez que se lo utilizo y a ser un software poco intuitivo para el usuario.
La simulación obtenida con el programa fue exitosa, y ayudo a comprender más el funcionamiento y las ventajas de un sistema basado en FPGA.