UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Departamento de Electrónica

Materia: Robótica

Trabajo Práctico 2

Docente: Ing. Hernán Gianetta

Ayudante de TP: Ing. Damián Granzella

Grupo N°:

Alumnos :

	Apellido y Nombre	Legajo
1	Resquín, Fernando Emilio	112316-6

Tabla de contenido

1. Introducción Teórica

El siguiente trabajo tiene como finalidad realizar un estudio de la dinámica de la plataforma móvil del robot propuesto. Se compone de una base donde va anclado el manipulador y contiene todos los sistemas electrónicos y mecánicos del mismo, y un sistema de ruedas en configuración diferencial. Este trabajo complementa el del análisis cinemático realizado anteriormente. No se realizará el estudio de dinámica de manipuladores, ya que puede obtenerse de la bibliografía y existen herramientas matemáticas que ofrecen la solución mediante cálculo numérico.

La dinámica es 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 estado de movimiento. Por lo tanto, el modelo dinámico de un robot 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 posición y orientación del cuerpo del robot en el plano y sus derivadas: velocidad y aceleración.
- Las fuerzas y pares aplicados en las ruedas y en el cuerpo del robot.
- Los parámetros dimensionales del robot, como dimensiones, masas e inercias de sus elementos.

La obtención del modelo dinámico es imprescindible para:

- Simular el robot.
- Diseñar y evaluar la estructura mecánica.
- Dimensionar los motores de las ruedas.
- Diseñar y evaluar el control dinámico.

1.1 Desarrollo Teórico del Modelo Dinámico:

Se tomó como referencia la publicación de Collazo Cuevas, Gorrostieta Hurtado, Pedraza Ortega, Geovanni Villaseñor Carrillo, Romero Torrez y González Aguirre. **{1}**.

Método Lagrangiano:

$$\tau = \frac{d}{dt} \cdot \frac{d\mathcal{L}}{d\dot{q}_n} - \frac{d\mathcal{L}}{dq_n} \quad ; \quad \mathcal{L} = \mathcal{k} - \mu$$

q_n: Coordenada de articulación

τ: Torque requerido o aplicado en la articulación "n"

$\mathcal{L}$ = Función Lagrangiana

$\mathcal{k}$ = Energía cinética

μ = Energía Potencial

En el caso de un robot con ruedas, moviéndose sobre un plano, la energía potencial es nula ($\mu = 0$), ya que $h = 0$.

$$\mathcal{L} = K_c + K_{ri} + K_{rd} \quad (10)$$

K_c = Energía cinética del cuerpo del robot

K_{ri} = Energía cinética de la rueda izquierda

K_{rd} = Energía cinética de la rueda derecha

$$\mathcal{L} = \frac{1}{2}M.v^2 + \frac{1}{2}I.\omega^2 \quad (11)$$

$\frac{1}{2}M.v^2$: Energía cinética de translación

$\frac{1}{2}I.\omega^2$: Energía cinética de rotación

M: Masa del cuerpo del robot

I: Momento de inercia del robot respecto de su eje de rotación

Luego, la energía cinética de cada rueda es:

$$K_{ri} = \frac{1}{2}\cdot M_r \cdot {v_i}^2 + \frac{1}{2}\cdot I_r \cdot {\omega_i}^2 \quad (12)$$

$$K_{rd} = \frac{1}{2}\cdot M_r \cdot {v_d}^2 + \frac{1}{2}\cdot I_r \cdot {\omega_d}^2 \ (13)$$

Reemplazando (11), (12) y (13) en (10):

$$\mathcal{L} = \frac{1}{2}M.v^2 + \frac{1}{2}I.\omega^2 + \frac{1}{2}\cdot M_r \cdot {v_i}^2 + \frac{1}{2}\cdot I_r \cdot {\omega_i}^2 + \frac{1}{2}\cdot M_r \cdot {v_d}^2 + \frac{1}{2}\cdot I_r \cdot {\omega_d}^2$$

$$\mathcal{L} = \frac{1}{2}M.v^2 + \frac{1}{2}I.\omega^2 + \frac{1}{2}\cdot M_r \cdot ({v_i}^2 + {v_d}^2) + \frac{1}{2}\cdot I_r \cdot ({\omega_i}^2 + {\omega_d}^2)$$

$$\mathcal{L} = \frac{1}{2}M.v^2 + \frac{1}{2}I.\dot{\varphi}^2 + \frac{1}{2}\cdot M_r \cdot ({v_i}^2 + {v_d}^2) + \frac{1}{2}\cdot I_r \cdot \left({\dot{\theta}_i}^2 + {\dot{\theta}_d}^2\right) \quad (14)$$

Sabiendo que:

$\dot{\varphi} = \frac{v_d - v_i}{2.b}$ (15) $\qquad v = \frac{1}{2}\cdot(v_i + v_d)$ (16)

$v_i = \dot{\theta}_i \cdot r$ (17) $\qquad v_d = \dot{\theta}_d \cdot r$ (18)

Reemplazando (15), (16) y (17) y (18) en (14):

$$\mathcal{L} = \frac{1}{8}M.(\dot{\theta}_i \cdot r + \dot{\theta}_d \cdot r)^2 + \frac{I}{8.b^2}\cdot\left(\dot{\theta}_d \cdot r - \dot{\theta}_i \cdot r\right)^2 + \frac{M_r.r^2}{2}\cdot\left({\dot{\theta}_i}^2 + {\dot{\theta}_d}^2\right) + \frac{I_r}{2}\cdot\left({\dot{\theta}_i}^2 + {\dot{\theta}_d}^2\right)$$

$$\mathcal{L} = \frac{1}{8}M.\left(\dot{\theta_i}^2 \cdot r^2 + 2.\dot{\theta_i}.\dot{\theta_d}.r^2 + \dot{\theta_d}^2 \cdot r^2\right) + \frac{I.r^2}{8.b^2}\cdot\left(\dot{\theta_i}^2 - 2.\dot{\theta_i}.\dot{\theta_d} + \dot{\theta_d}^2\right) + \frac{M_r.r^2}{2}\cdot\left(\dot{\theta_\iota}^2 + \dot{\theta_d}^2\right) + \frac{I_r}{2}\cdot\left(\dot{\theta_\iota}^2 + \dot{\theta_d}^2\right)$$

$$\mathcal{L} = \frac{1}{8}M.r^2.\left(\dot{\theta_i}^2 + 2.\dot{\theta_i}.\dot{\theta_d} + \dot{\theta_d}^2\right) + \frac{I.r^2}{8.b^2}\cdot\left(\dot{\theta_i}^2 - 2.\dot{\theta_i}.\dot{\theta_d} + \dot{\theta_d}^2\right) + \frac{M_r.r^2}{2}\cdot\left(\dot{\theta_\iota}^2 + \dot{\theta_d}^2\right) + \frac{I_r}{2}\cdot\left(\dot{\theta_\iota}^2 + \dot{\theta_d}^2\right)$$

$$\mathcal{L} = \left(\frac{1}{8}M.r^2 + \frac{I.r^2}{8.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\dot{\theta_i}^2 + \left(\frac{1}{8}M.r^2 + \frac{I.r^2}{8.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\dot{\theta_d}^2 + 2.\frac{1}{8}M.r^2.\dot{\theta_i}.\dot{\theta_d} - 2.\frac{I.r^2}{8.b^2}\cdot\dot{\theta_i}.\dot{\theta_d}$$

$$\mathcal{L} = \left(\frac{1}{8}M.r^2 + \frac{I.r^2}{8.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\dot{\theta_i}^2 + \left(\frac{1}{8}M.r^2 + \frac{I.r^2}{8.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\dot{\theta_d}^2 + \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right)\cdot\dot{\theta_i}.\dot{\theta_d} \quad (19)$$

Método Lagrangiano:

$$\frac{\partial}{\partial t}\cdot\frac{\partial\mathcal{L}}{\partial\dot{\theta}_i} - \frac{\partial\mathcal{L}}{\partial\theta_i} \quad (20) \quad ; \quad \frac{\partial}{\partial t}\cdot\frac{\partial\mathcal{L}}{\partial\dot{\theta}_d} - \frac{\partial\mathcal{L}}{\partial\theta_d} \quad (21)$$

Aplicando (20) a (19):

$$\tau_i = 2\cdot\left(\frac{1}{8}M.r^2 + \frac{I.r^2}{8.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\ddot{\theta_\iota} + \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right)\cdot\ddot{\theta_d}$$

$$\tau_i = \left(\frac{1}{4}M.r^2 + \frac{I.r^2}{4.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\ddot{\theta_\iota} + \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right)\cdot\ddot{\theta_d}$$

Aplicando (21) a (19):

$$\tau_d = \left(\frac{1}{4}M.r^2 + \frac{I.r^2}{4.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\cdot\ddot{\theta_d} + \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right)\cdot\ddot{\theta_\iota}$$

1.2 Modelo Dinámico Inverso:

$$\begin{bmatrix}\tau_i\\ \tau_d\end{bmatrix} = \begin{bmatrix}\left(\frac{1}{4}M.r^2 + \frac{I.r^2}{4.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right) & \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right)\\ \left(\frac{1}{4}M.r^2 - \frac{I.r^2}{4.b^2}\right) & \left(\frac{1}{4}M.r^2 + \frac{I.r^2}{4.b^2} + \frac{M_r.r^2}{2} + \frac{I_r}{2}\right)\end{bmatrix}\cdot\begin{bmatrix}\ddot{\theta}_i\\ \ddot{\theta}_d\end{bmatrix}$$

2. Simulación en Matlab del modelo dinámico y dimensionamiento de motores:

Para poder simular el modelo dinámico, hizo falta obtener los parámetros dinámicos del robot diseñado. Dicho diseño y dimensiones fue definido en el trabajo anterior, utilizando el software T-FLEX, en su versión estudiantil. Dicho software permite crear las piezas que componen el robot, definir materiales, colores y articulaciones. A su vez, posee una función de cálculo de parámetros dinámicos, básicamente masas e inercias.
Se utilizó este CAD para modelar el robot y obtener las masas e inercias de la plataforma y las ruedas. Los valores obtenidos fueron los sgtes:

Momento de Inercia del cuerpo (Plataforma + Manipulador): Ic = (2.13 kg.m2 + 1.7 kg.m2) = 38 N.m2

Momento de Inercia de la rueda: Ir = 0.5354 kg.m2 = 5.2469 N.m2

Masa del cuerpo (Plataforma + Manipulador): Mc = (53 kg + 10 kg) = 617.4 N

Masa de la rueda: Mr = 8.212 kg = 80.48 N

A su vez, sabiendo que en el modelo dinámico obtenido en el trabajo anterior se consideró rozamiento nulo, se puede asumir que el pico de torque en la aceleración y desaceleración depende solamente de:

- Masas y momentos de inercia del sistema.
- Saltos de velocidades en la aceleración y desaceleración.
- Tiempo que insume acelerar o desacelerar.

Como consecuencia de estos tres parámetros se debe definir una velocidad máxima para nuestro Robot. Tomamos como punto de comparación un robot comercial, el 710 de la firma iRobot, cuyas especificaciones están en la web del fabricante. **{2}**

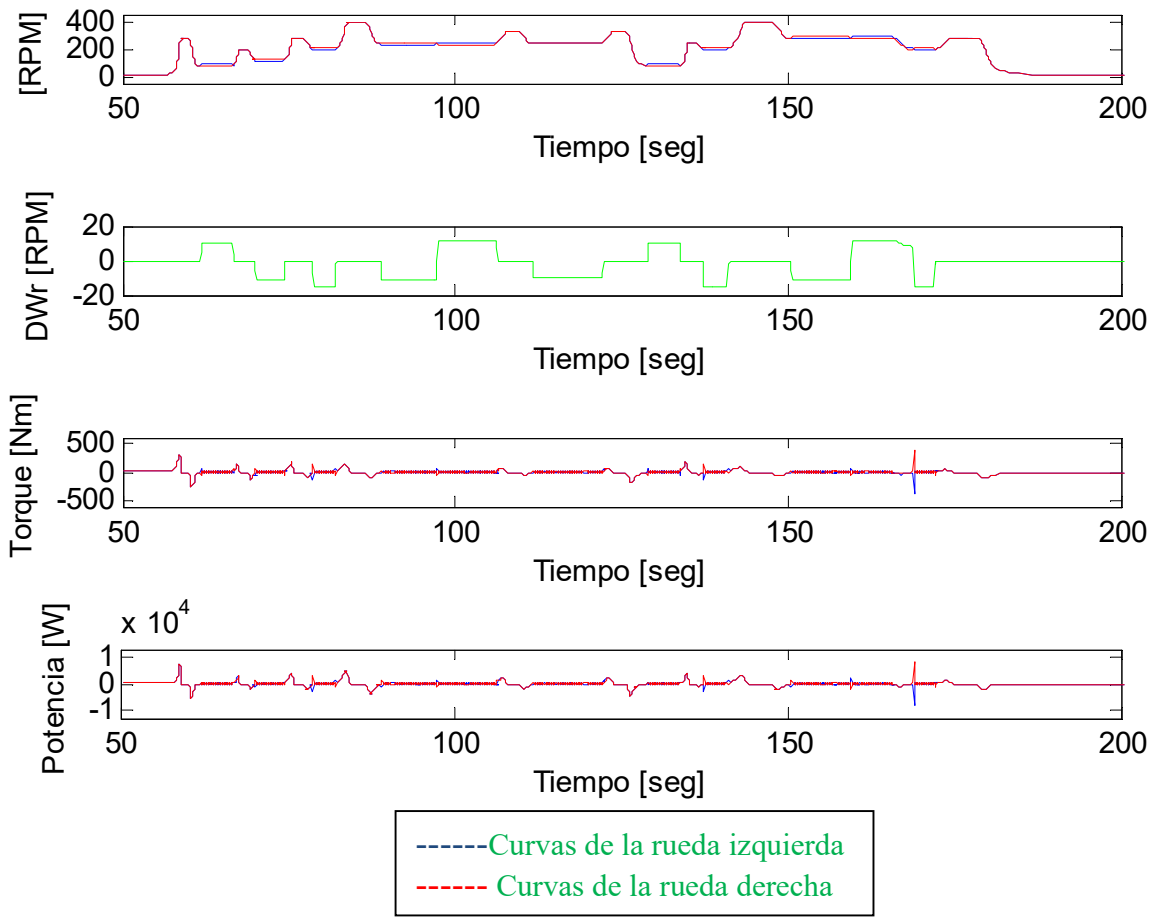
- Largo: 92 cm
- Ancho: 54 cm
- Alto: 46 cm
- Peso: 167 kg
- Velocidad máxima: 13km/h

Se consideraron las dimensiones y peso sin el brazo manipulador montado. En el caso de Jarvis, las dimensiones son menores:

- Ancho= 51 cm
- Alto=28 cm
- Largo=73 cm

El peso también es menor, por lo que se decidió que la velocidad máxima sea de 5 m/s (18 km/h).

Gráficas de RPM, Torque y potencia mecánica para cada rueda



El script de Matlab calcula el torque y la potencia máxima para cada rueda. Se observan picos de torque muy importantes pero de duración muy corta respecto de las constantes de tiempo del sistema mecánico, presentes en los instantes en que cambia el radio de curvatura. Estos valores son destacados en rojo pero también se tomó de las gráficas el valor máximo de torque, fuera de estos instantes especiales, dentro del tiempo que dura el movimiento.

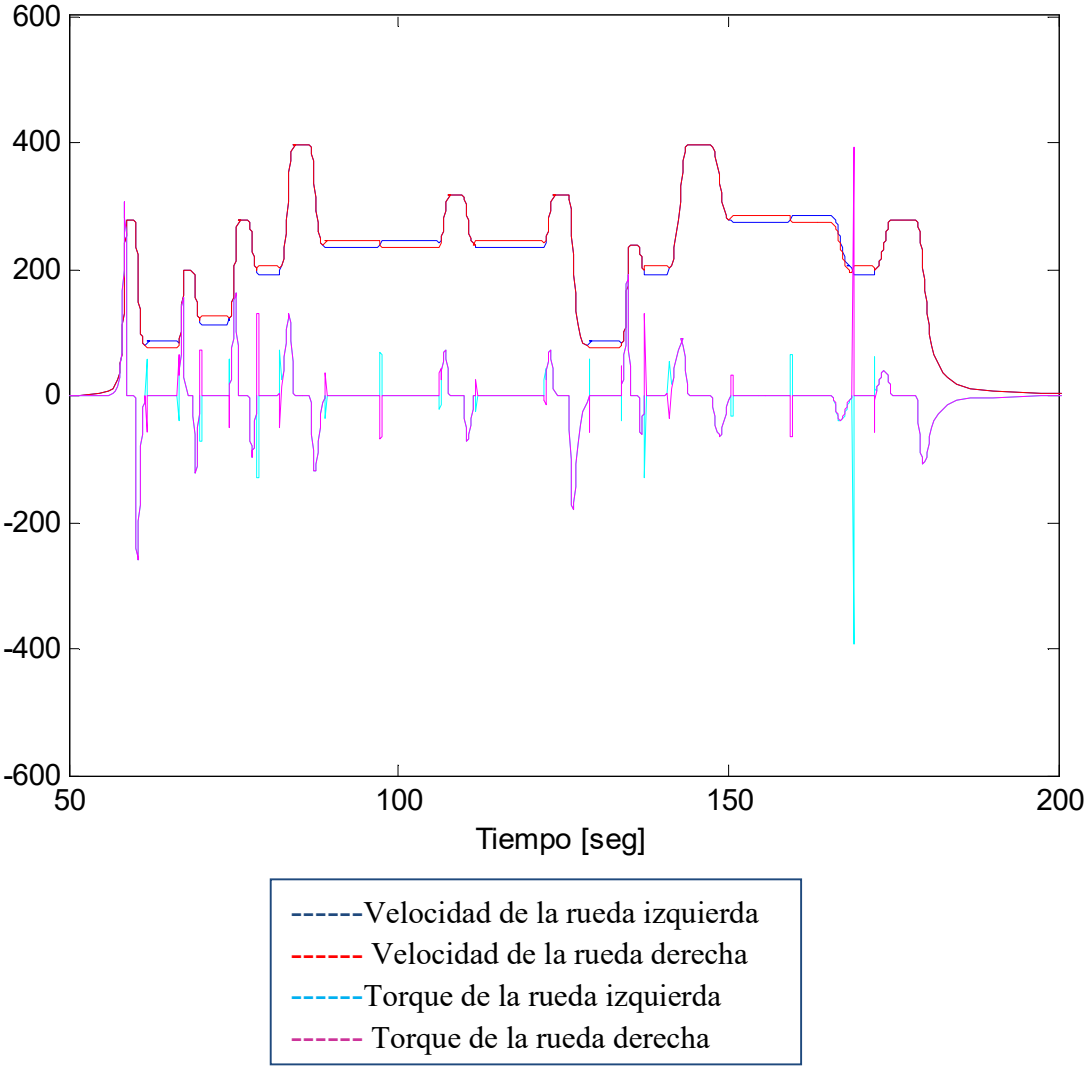
Rueda derecha:

Torque Máximo: -300 Nm ; Potencia Máxima: 7191 W; Velocidad máxima: 400 RPM
Torque Máximo: -392.302427 Nm ; Potencia Máxima: -7846.045108 W

Rueda izquierda:

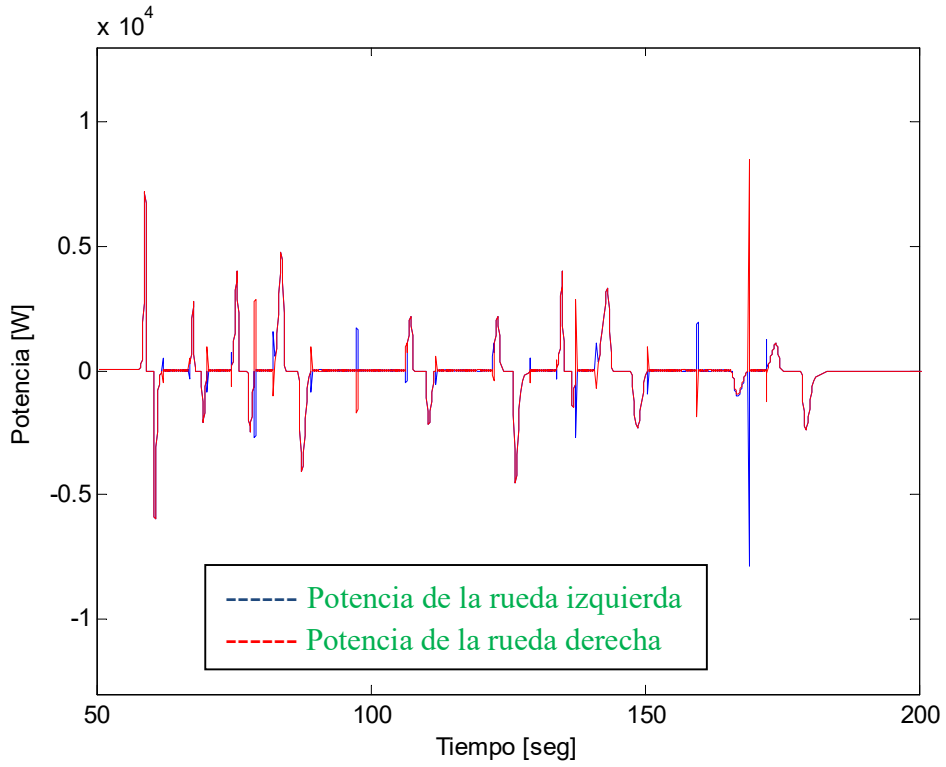
Torque Máximo: -300 Nm ; Potencia Máxima: 7191 W; Velocidad máxima: 400 RPM
Torque Máximo: 392.302427 Nm ; Potencia Máxima: 8499.889345 W

Velocidades angulares y torques de cada rueda - Comparativa temporal

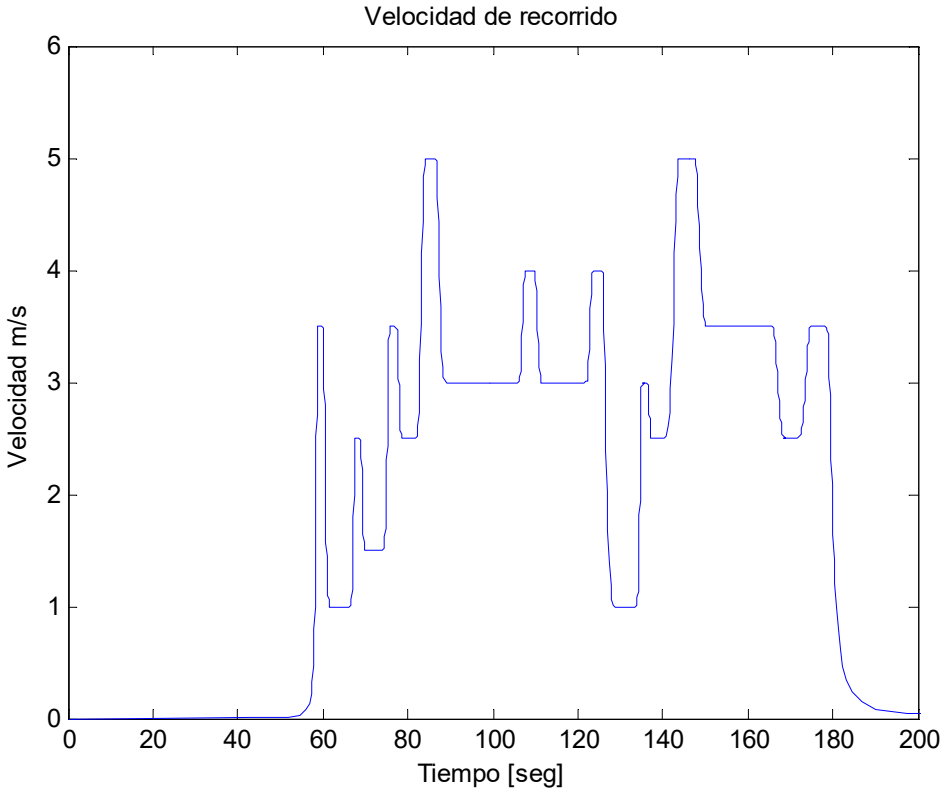


Como se observa en la sgte gráfica, los motores de las ruedas estarán entregando potencia mecánica al sistema cuando la rueda debe acelerar, pero estará tomando energía mecánica del sistema cuando la misma debe frenar. En el segundo caso, el motor se estará comportando como generador.

Potencia mecánica de cada rueda - Comparativa temporal

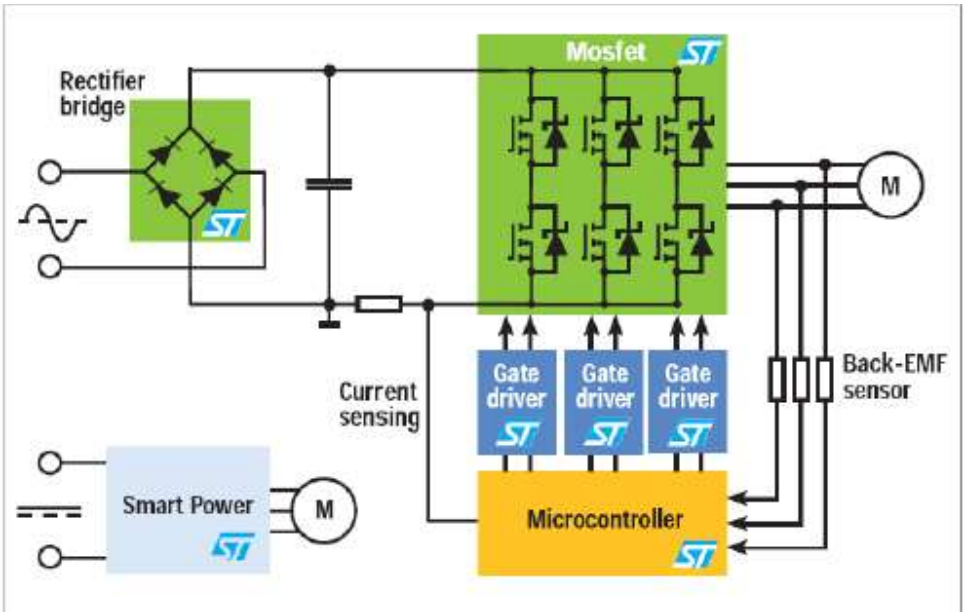


Velocidad Lineal de recorrido del robot



3. Interfaz de control de motores

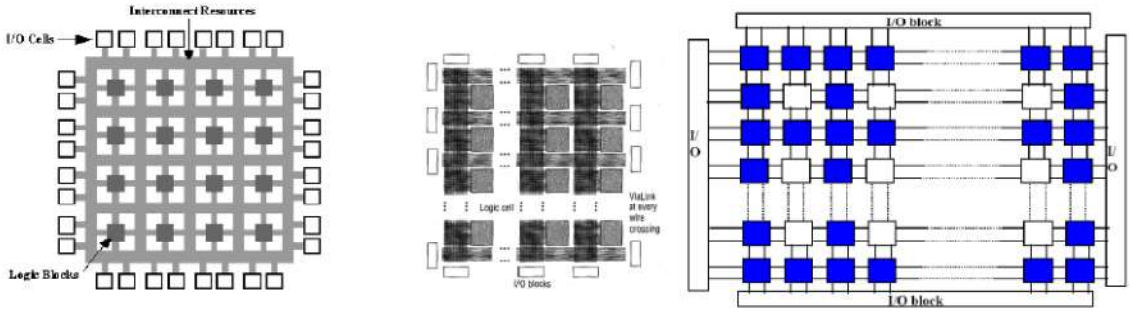
Un motor BLDC (Brushless DC) es un motor de corriente continua en el cual la conmutación de bobinados no se realiza mediante escobillas, sino que se realiza mediante un sistema electrónico externo. En cierta forma, está construido como un motor trifásico de inducción, salvo que el rotor está compuesto por un arreglo de imanes permanentes.
Para que el motor funcione es necesario energizar los bobinados en una secuencia precisa, y en sincronización con la posición del rotor. Para detectar la posición del mismo se suele utilizar un arreglo de sensores de efecto HALL. Otro método para dicha detección es a través de la fuerza contra-electromotriz que aparece en el estator.
En definitiva, se reemplazan las escobillas por un sistema electrónico a lazo cerrado.



El robot desarrollado en el presente trabajo requiere de 3 motores BLDC para el brazo manipulador y 2 motores en las ruedas de la plataforma móvil, o sea 5 motores en total.
La implementación de este sistema con una FPGA en lugar de un microcontrolador se justifica en que se está controlando cada motor por hardware, con un tiempo de respuesta mucho menor que el de un microcontrolador, y en paralelo para cada uno de los motores, por lo cual se van a lograr mejores resultados. Por otro lado, el PWM de cada motor funciona en alta frecuencia y por lo general los GPIO de un microcontrolador no permiten esto.
Una consideración a tener en cuenta al momento de programar cada PWM es evitar que durante los switchs de un Mosfet a otro, se genere un corto circuito en la fuente de alimentación. Esto se resuelve simplemente ubicando una pequeña demora del orden de 1 us entre desactivación de un PWM y activación del siguiente correspondientes al mismo motor.
La excitación de los bobinados de cada motor se realiza mediante PWM a través de una etapa de potencia MOSFET, con estructura de puente H. La velocidad y torque se ajusta variando el ciclo de actividad.

3.1 Descripción de un FPGA

El FPGA representa uno de los últimos avances en tecnología de dispositivos lógicos programables, es importante señalar que una FPGA realmente se re configura con un programa, a diferencia de lo que normalmente se conoce como sistema programado (microcontrolador, microprocesador, etc.) en donde un hardware fijo es capaz de interpretar y ejecutar un programa especificado como un conjunto de instrucciones por el programador, en las FPGA lo que se tiene es un hardware que se configura realizando conexiones físicas que son especificadas por un programa o cadena de configuración.
Las FPGA a diferencia de los PLD y PAL, es que su estructura no está compuesta por compuertas AND/OR , en su lugar contienen blocks lógicos para implementar las funciones requeridas.



La "programación" de un FPGA se realiza en un lenguaje llamado VHDL (VHSIC Hardware Description Language)

3.2 Implementación de la interfaz

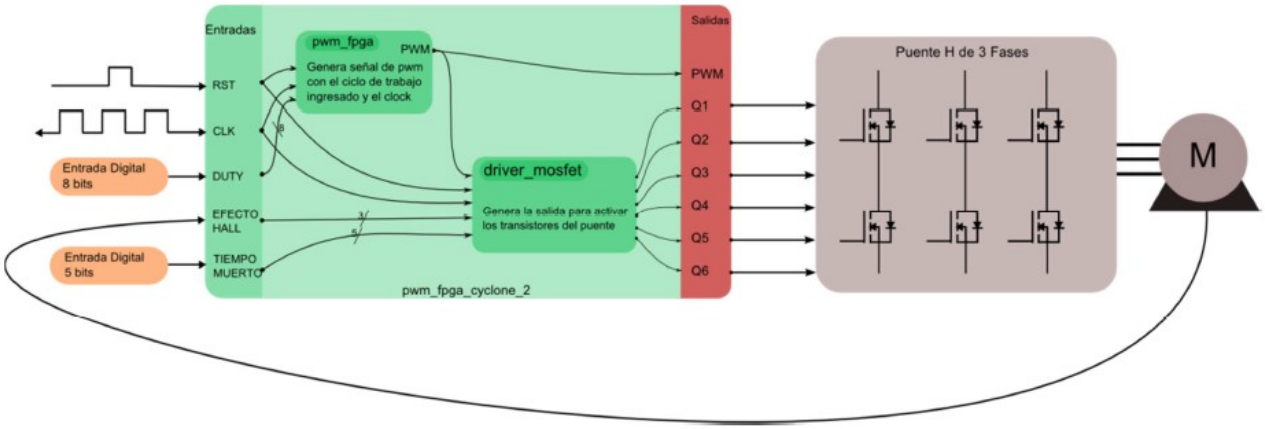
El dispositivo final a sintetizar (JarvisFPGA) consta de las siguientes entradas:

- 40 entradas por las cuales se especifica de manera independiente y en palabras de 8 bits, el ancho de pulso a generar por cada generador de PWM (como son cinco generadores, entonces 5x8=40)
- Una entrada de reset
- Una entrada de clock
- 15 entradas para los sensores hall, 3 por motor. Al ser 5 motores, 5x3=15.

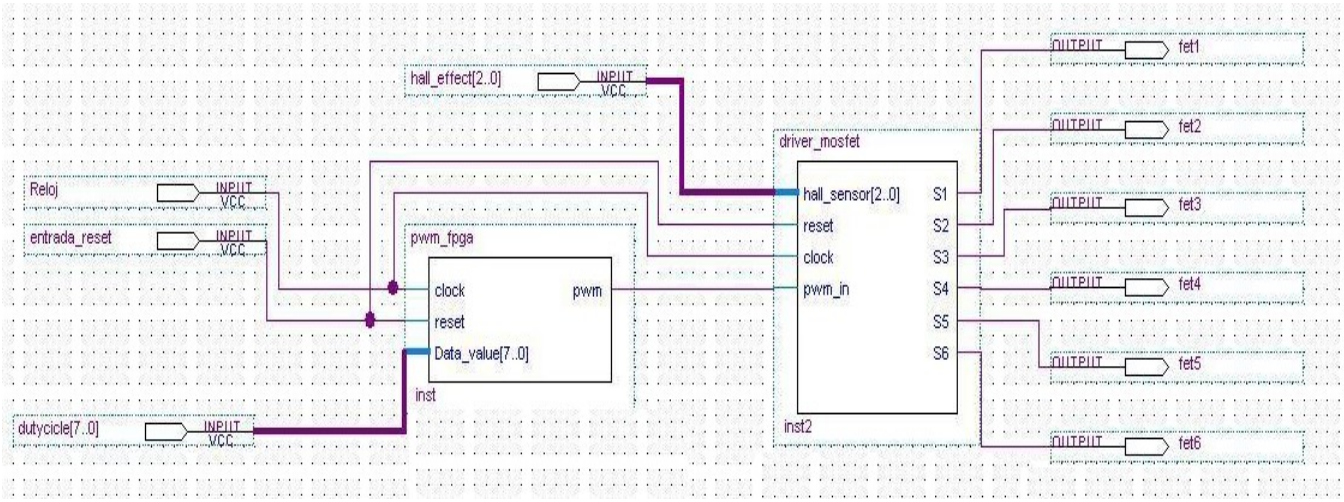
A su vez, contiene las siguientes salidas:

- 5 salidas de PWM. Cada una consiste en el output puro del generador de PWM para cada motor
- 30 salidas de manera de poder conectar seis transistores por motor (al ser cinco motores, 5x6=30). Estas salidas contienen el output de PWM conmutado entre ellas de acuerdo a lo que marcan las entradas de los sensores hall para permitir que el motor gire.

Esquema general del sistema y su codificación en VHDL



Para la implementación de dicha solución se utilizó el software "Altera 15.0.0 Web Edition", dentro de cual se incluye el aplicativo "ModelSim".
Altera es un fabricante de dispositivos, entre ellos las FPGA. Sin embargo, la librerías VHDL utilizadas son genéricas.



Cada dispositivo JarvisFPGA está compuesto por cinco controladores PWM (pwm_fpga_cyclone_2) definidos como componentes. Estos últimos contienen a su vez dos componentes cada uno:

- **pwm_fpga**
  Este bloque crea una señal en base a la frecuencia del clock y el ciclo de trabajo adoptado. Luego, esa señal es utilizada para variar la corriente media que alimenta al motor. De esta manera, se tiene un control sobre la capacidad de torque que debe realizar el motor.
- **driver_mosfet**
  Controla la secuencia de encendido de los transistores del puente H. Esta secuencia varía según la posición del motor, la cual es realimentada por los sensores de efecto Hall. La conducción de una fase se dará cuando 2 transistores estén saturados.

Para ensayar el bloque "pwm_fpga_cyclone_2" y poder simular su funcionamiento se utiliza un bloque llamado "TestbenchPWMCyclone". Este "banco de pruebas" permite la simulación de un solo driver, pero dicha simulación será válida para todos, ya que son idénticos.

Como se había dicho previamente, la excitación de los bobinados debe darse de acuerdo a la posición del rotor, lo que constituye una secuencia de conmutación, como se indica a continuación:

Sensores Efecto Hall			Transistores					
S_2	S_1	S_0	Q1	Q2	Q3	Q4	Q5	Q6
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

Secuencia PWM

3.3 Código VHDL de los componentes

TestBench_PWMCyclone

```
--*******************************************************
--          Testbench File for design pwm_fpga produced by
--                    Atmel System Designer on
--                    May 25, 2001 1:54:17 pm
--*******************************************************

LIBRARY ieee;

use ieee.std_logic_1164.all;


ENTITY TestBench_PWMCyclone IS

generic(clk_period:TIME:=100 ns;
      encoder_rate: TIME:= 250 us);

END TestBench_PWMCyclone;

ARCHITECTURE arch_test_bench OF TestBench_PWMCyclone IS

COMPONENT pwm_fpga_cyclone_2
          port
          (
                    reloj :  IN  STD_LOGIC;
                    reset :  IN  STD_LOGIC;
                    dutycicle :  IN  STD_LOGIC_VECTOR(7 downto 0);
                    hallsensor :  IN  STD_LOGIC_VECTOR(2 downto 0);
                    Sal1 :  OUT  STD_LOGIC;
                    Sal2 :  OUT  STD_LOGIC;
                    Sal3 :  OUT  STD_LOGIC;
                    Sal4 :  OUT  STD_LOGIC;
                    Sal5 :  OUT  STD_LOGIC;
                    Sal6 :  OUT  STD_LOGIC;
                    pwm1 :  OUT  STD_LOGIC
          );
END COMPONENT;
```

```
-- Internal signal declaration
SIGNAL sig_clock          : std_logic;
SIGNAL sig_reset          : std_logic;
SIGNAL reload_var         : std_logic_vector(7 downto 0);
SIGNAL sig_hall_effect    : std_logic_vector(2 downto 0);
SIGNAL sig_pwm_out        : std_logic;

SIGNAL sig_fet1: std_logic;
SIGNAL sig_fet2: std_logic;
SIGNAL sig_fet3: std_logic;
SIGNAL sig_fet4: std_logic;
SIGNAL sig_fet5: std_logic;
SIGNAL sig_fet6: std_logic;



shared variable ENDSIM: boolean:=false;

-- El cuerpo del testbench.
BEGIN

-- Instantiating top level design Componentpwm_cyclone
inst_pwm_fpga_cyclone_2 : pwm_fpga_cyclone_2
PORT MAP(
                reloj => sig_clock,
                reset      => sig_reset,
                dutycicle => reload_var,
                hallsensor => sig_hall_effect,
                Sal1 => sig_fet1,
                Sal2 => sig_fet2,
                Sal3 => sig_fet3,
                Sal4 => sig_fet4,
                Sal5 => sig_fet5,
                Sal6 => sig_fet6,
                pwm1 => sig_pwm_out
          );


-- Procesos generadores de las seÃ±ales de estÃ-mulo para la simulaciÃ³n de la respuesta del circuito lÃ³gico.
-- Generador de la seÃ±al de RELOJ
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;


-- Generador de la seÃ±al de reset inicial y de las seÃ±ales de recarga de los mÃ³dulos PWM para variar el dutycicle.
stimulus_process: PROCESS
```

```
  BEGIN

  if (endsim = false) then
    sig_reset <= '1';
              wait for clk_period;
    sig_reset <= '0';
    reload_var <= "00010000";
    wait for 100 us;
    reload_var <= "00100000";
    wait for 100 us;
              reload_var <= "00110000";
              wait for 100 us;
              reload_var <= "01000000";
              wait for 100 us;
              reload_var <= "01010000";
              wait for 100 us;
              reload_var <= "01110000";
              wait for 100 us;
              reload_var <= "10000000";
              wait for 100 us;
              reload_var <= "11100000";
              wait for 100 us;
    reload_var <= "11110000";
    wait;
  else
    wait;
  end if;

            END PROCESS stimulus_process;


-- Generador de la señales del efecto hall.
switch_process: PROCESS
  BEGIN

  if (endsim = false) then

    sig_hall_effect <= "000";
    wait for 150 us;
    sig_hall_effect <= "100";
    wait for 150 us;
    sig_hall_effect <= "110";
    wait for 150 us;
    sig_hall_effect <= "010";
    wait for 150 us;
    sig_hall_effect <= "011";
    wait for 150 us;
    sig_hall_effect <= "001";
    wait for 150 us;
    sig_hall_effect <= "101";
    wait;
  else
    wait;
  end if;

END PROCESS switch_process;
```

```
END arch_test_bench;
```

pwm_fpga_cyclone

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY pwm_fpga_cyclone_2 IS
	port
	(
		reloj :  IN  STD_LOGIC;
		reset :  IN  STD_LOGIC;
		pwm1 :  OUT  STD_LOGIC;
		dutycicle :  IN  STD_LOGIC_VECTOR(7 downto 0);
		hallsensor :  IN  STD_LOGIC_VECTOR(2 downto 0);
		Sal1 :  OUT  STD_LOGIC;
		Sal2 :  OUT  STD_LOGIC;
		Sal3 :  OUT  STD_LOGIC;
		Sal4 :  OUT  STD_LOGIC;
		Sal5 :  OUT  STD_LOGIC;
		Sal6 :  OUT  STD_LOGIC
	);
END pwm_fpga_cyclone_2;

ARCHITECTURE bdf_type OF pwm_fpga_cyclone_2 IS

component driver_mosfet
	PORT(reset : IN STD_LOGIC;
		clock : IN STD_LOGIC;
		pwm_in : IN STD_LOGIC;
		hall_sensor : IN STD_LOGIC_VECTOR(2 downto 0);
		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 component;

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;


signal	SYNTHESIZED_WIRE_23 :  STD_LOGIC;


BEGIN

pwm1 <= SYNTHESIZED_WIRE_23;
```

```
b2v_inst : driver_mosfet
PORT MAP(
    reset => reset,
                clock => reloj,
                pwm_in => SYNTHESIZED_WIRE_23,
                hall_sensor => hallsensor,
                S1 => Sal1,
                S2 => Sal2,
                S3 => Sal3,
                S4 => Sal4,
                S5 => Sal5,
                S6 => Sal6 );

b2v_inst4 : pwm_fpga
PORT MAP(clock => reloj,
                reset => reset,
                Data_value => dutycicle,
                pwm => SYNTHESIZED_WIRE_23);

END ARCHITECTURE bdf_type;
```

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              :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;
```

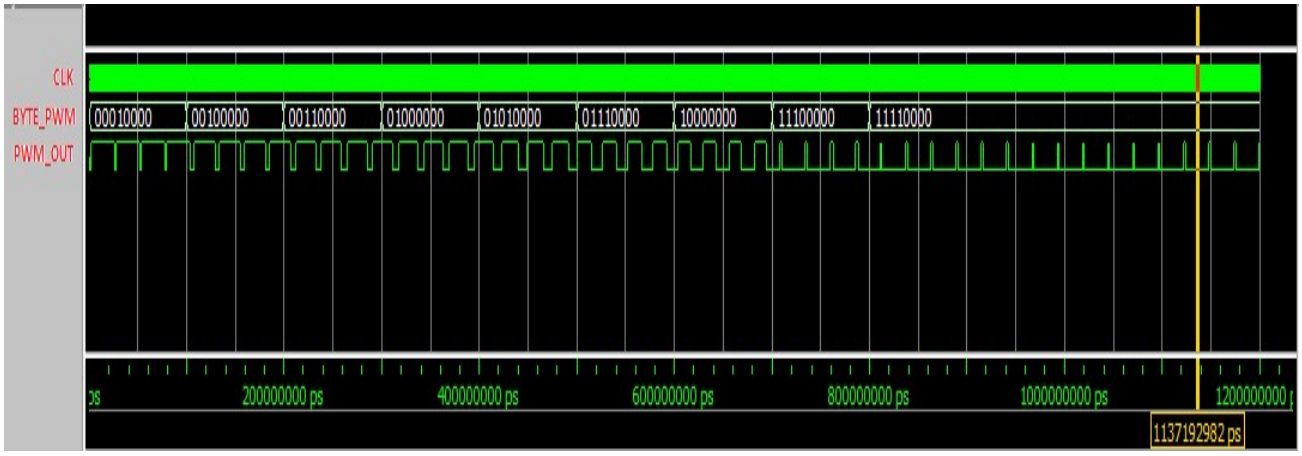
driver_mosfet

```
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 :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';
```

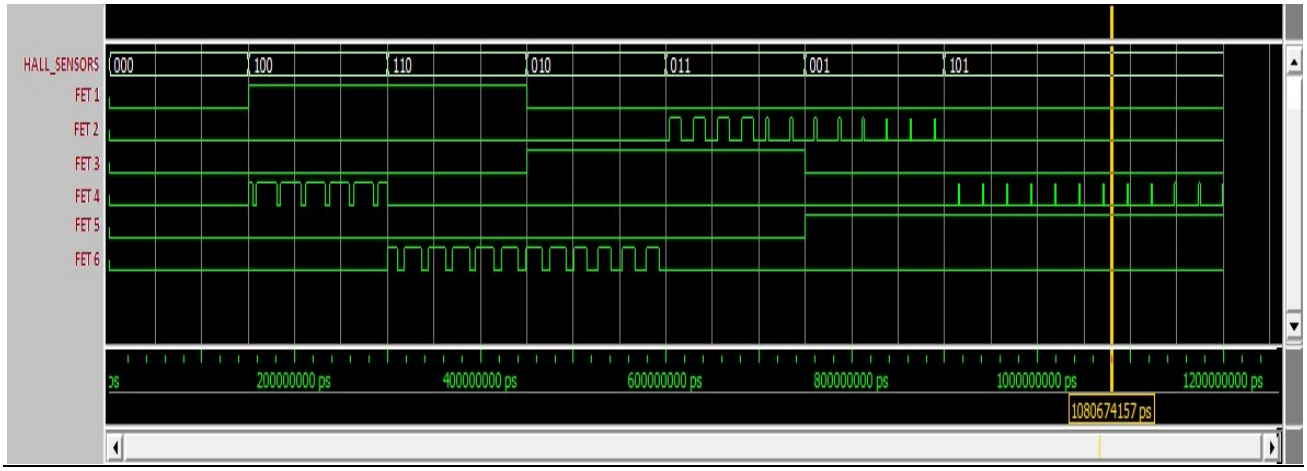
```
                WHEN "101" =>                   --Estado 6
                        s1<= '0'; s2<= '0'; s3<= '0'; s4<= pwm_in; s5<= '1'; s6<= '0';

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

END PROCESS;

END ARCHITECTURE arch_driver;
```

3.4 Resultados de la simulación

En esta imagen se observa la variación del ciclo de actividad de la salida PWM de acuerdo a la palabra de control.



En esta imagen se observa la secuencia de conmutación de los MOSFET, en concordancia con la tabla anterior



4. Conclusiones

- **En líneas generales se observa que el presente desarrollo dio los resultados esperados dentro de las restricciones propuestas. Los desarrollos matemáticos pudieron verificarse totalmente en la simulación de Matlab. En el caso de la simulación de FPGA, el modelo fué desarrollado por la cátedra, aunque se pudo verificar que la simulación coincide con lo visto en clase.**

- En la simulación del modelo dinámico en Matlab se observó que aparecen picos de torque muy angostos en los instantes en que cambia el radio de curvatura de la trayectoria. Esto ocurre al pasar de un tramo recto a uno curvo y viceversa, porque los tramos curvos están basados por simplicidad en arcos de circunferencia, que presentan radios de curvatura de valor finito y constante desde el inicio hasta el final del giro. Como consecuencia de ello, la velocidad angular de las ruedas dejan de ser coincidentes bruscamente, generando (teóricamente) una Delta de Dirac de torque ante un escalón de velocidad angular.
  Para solucionar este inconveniente se puede implementar un filtro pasabajos en el setpoint de velocidad de cada driver, para limitar su velocidad de variación. A su vez, de acuerdo a la complejidad pretendida en el sistema de control de motores, puede implementarse una función de límite de torque que impida daños en el mismo. Posiblemente a consecuencia de ello existan errores de seguimiento de la trayectoria en el momento del giro, que deberán compensarse en niveles superiores del sistema de control del robot. Este es el motivo por el cual no se considerarían estos picos para dimensionar los motoreductores asociados a las ruedas.

- El modelo dinámico planteado es una primera aproximación que no considera la parte disipativa del sistema, por una cuestión de simplicidad matemática. En la práctica los rodamientos de los ejes presentan una componente disipativa, las ruedas presentan rozamiento contra el suelo y la resistencia del aire pueden ser un factor importante a considerar. En caso de incluirlos en el modelo, el sistema consumirá mayor energía, lo que deriva en la utilización de motoreductores de mayor potencia.

- Uno puede pensar que el diseño del robot se limita a definir formas u zonas de trabajo, pero esto no es así. La parte más compleja del diseño es la que permite el movimiento.
  Si bien la electrónica y las implementaciones de control no se ven, estas requieren un cálculo y análisis incluso más extenso que la definición del robot en sí.

- Yendo al sentido electrónico del trabajo, la utilización de VHDL demuestra como una FPGA puede resolver fácilmente la proyección de sistemas electrónicas que de otra manera serian muy complicados.

5. Referencias:

1. Collazo Cuevas J, Gorrostieta Hurtado E, Pedraza Ortega J, Geovanni Villaseñor Carrillo U, Romero Torrez R y González Aguirre M. *Modelación de un Robot Móvil de Dos Ruedas con Tracción Diferencial*
   Disponible en
   http://www.mecamex.net/anterior/cong08/articulos/58.pdf

2. iRobot® 710 Kobra Specs
   Disponible en:
   http://www.irobotweb.com/~/media/Files/Robots/Defense/710/710_Specs.pdf?la=en

- Además, se tomaron ideas de los trabajos previos de 2014:
  - Aguirre F y Carlucci Rigoni T.
  - Suárez J M y Vales J P.
  - Espain F y Ferreyro L.
  - Fabbro M y Pinola A.
  - Trejo M y Ferrari M
  - Sokolowicz M y Focaraccio M.
  - Zubiaurre F y Zubiaurre S.
  - Gimeno A y Donnadio G.
  - Samudio Caceres B y Samez M