Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Trabajo Práctico Nº 2:
Dinámica del Robot Paralelo Delta

Materia: Robótica

Curso: R6051

Año: 2014

Docente: Ing. Hernán Giannetta

ATP: Ing. Damián Granzella, Ing. Lucas Barrera

Alumnos: Agustín G. Gimeno, Gustavo Donnadio

Contenido

Introducción

El presente trabajo tiene por fin principal obtener el modelo dinámico de un robot cuyo fin es conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo.

El modelo dinámico 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 los actuadores y en el extremo del robot
- Los parámetros dimensionales del robot, como longitud, masas e inercias de sus elementos

Dinámica del Robot Delta

Se proponen las siguientes hipótesis para simplificar la obtención del modelo dinámico del Robot Delta:

- La inercia de los brazos inferiores se desprecia.
- Se desprecia la fricción y elasticidad.
- La masa de los brazos inferiores se separa en dos y se ubica en las dos extremidades, 2/3 en el brazo superior y 1/3 en la plataforma inferior.

Parámetros Dinámicos

En la plataforma inferior (TCP) influyen:

- La masa de la plataforma m_n
- La masa de la carga $m_{payload}$
- Las contribuciones de las masas de cada unos de los brazos inferiores m_{fb}

$$m_{nt} = m_n + m_{payload} + 3 \times (\frac{1}{3}\, m_{fb})$$

La posición del centro de masa de cada brazo superior es calculada:

$$r_{Gb} = l_A \frac{\frac{1}{2}m_b + \frac{2}{3}m_{fb}}{m_b + \frac{2}{3}m_{fb}}$$

m_b es la masa del brazo superior y l_A es la longitud del brazo superior.

La contribución inercial de cada brazo superior es la suma de la inercia del motor I_m y la inercia del brazo superior I_{bc}.

$$I_{bi} = I_m + I_{bc}$$

$$I_{bc} = {l_A}^2 \left(\frac{m_b}{3} + \frac{2}{3}m_{fb}\right)$$

Principio del Trabajo Virtual

El trabajo Virtual en un sistema es el trabajo resultante de fuerzas virtuales actuando a través de un desplazamiento real o de fuerzas reales actuando a través de un desplazamiento virtual. Desplazamiento se refiere tanto a traslación como rotación y fuerzas tanto para fuerzas como momentos.

La igualdad de los trabajos virtuales asociados a los sistemas de coordenadas:

$$\tau^T.\delta\theta = {\tau_n}^T.\delta X_n$$

Donde τ es el torque/fuerza correspondiente al desplazamiento $\delta\theta$ de los actuadores y τ_n el torque/fuerza actuando en la plataforma móvil correspondiente al desplazamiento δX_n.

La matriz jacobiana puede ser usada para transformar el torque/fuerza actuando en la plataforma móvil al espacio de los actuadores.

$$\dot{X}_n = J.\dot{\theta}$$

$${\tau^T = \tau_n}^T.J \text{ es igual a } \tau = \tau_n.J^T$$

De acuerdo con estas hipótesis se puede reducir el robot en 4 cuerpos: la plataforma móvil y los tres brazos superiores. Luego se puede resolver la dinámica transformando la contribución de los torques/fuerzas de acuerdo al principio del trabajo virtual.

En la plataforma móvil actúan dos fuerzas, la fuerza gravitacional G_n y la fuerza inercial F_n.

$$G_n = m_{nt}(0 \quad 0 \quad -g)^T$$

$$F_n = m_{nt}.\ddot{X}_n$$

Utilizando la matriz jacobiana como previamente se describió se puede obtener:

$$\tau_n = J^T.F_n = J^T.m_{nt}.\ddot{X}_n$$

$$\tau_{Gn} = J^T.G_n = J^T.m_{nt}(0 \quad 0 \quad -g)^T$$

En los actuadores hay dos clases de torques, el torque producido por la fuerza gravitacional de cada brazo superior τ_{Gb} y el torque producido por la fuerza inercial actuando en cada brazo τ_b.

$$\tau_{Gb} = r_{Gb}G_b(cos\theta_1 \quad cos\theta_2 \quad cos\theta_3)^T$$

Donde G_b es la fuerza gravitacional actuando en el centro de masa de cada brazo superior.

$$G_b = g.\left(m_b + \frac{2}{3}m_{fb}\right)$$

La contribución de torque de cada brazo superior es:

$$\tau_b = I_b\ddot{\theta}$$

Donde I_b es la matriz inercial de los brazos:

$$I_b = \begin{bmatrix} I_{b1} & 0 & 0 \\ 0 & I_{b2} & 0 \\ 0 & 0 & I_{b3} \end{bmatrix}$$

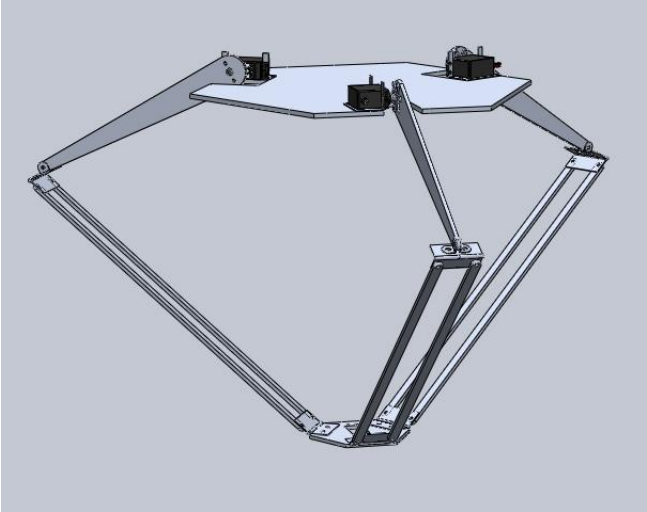
De acuerdo con el Principio de D'Alembert la contribución de todas las fuerzas inerciales debe ser igual a la contribución de las fuerzas no inerciales, con lo que se obtiene:

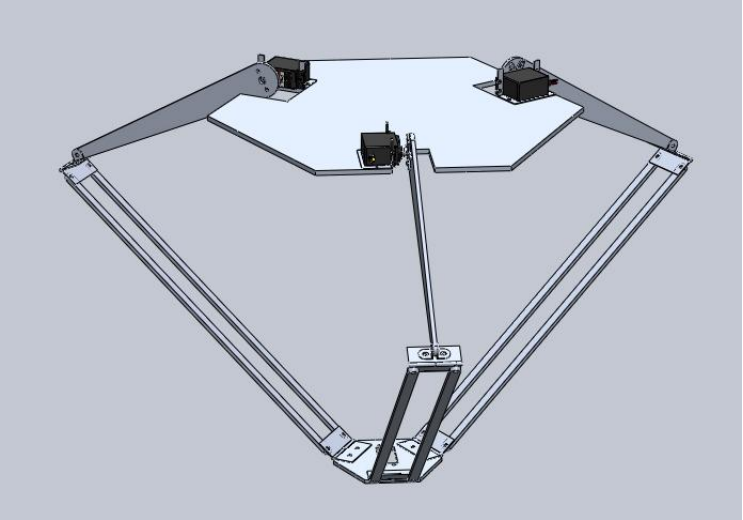
$$\tau + \tau_{Gn} + \tau_{Gb} = \tau_b + \tau_n$$

$$\tau = I_b\ddot{\theta} + J^T.m_{nt}.\ddot{X}_n - J^T.m_{nt}(0 \quad 0 \quad -g)^T - r_{Gb}G_b(cos\theta_1 \quad cos\theta_2 \quad cos\theta_3)^T$$

Modelización del Robot Delta

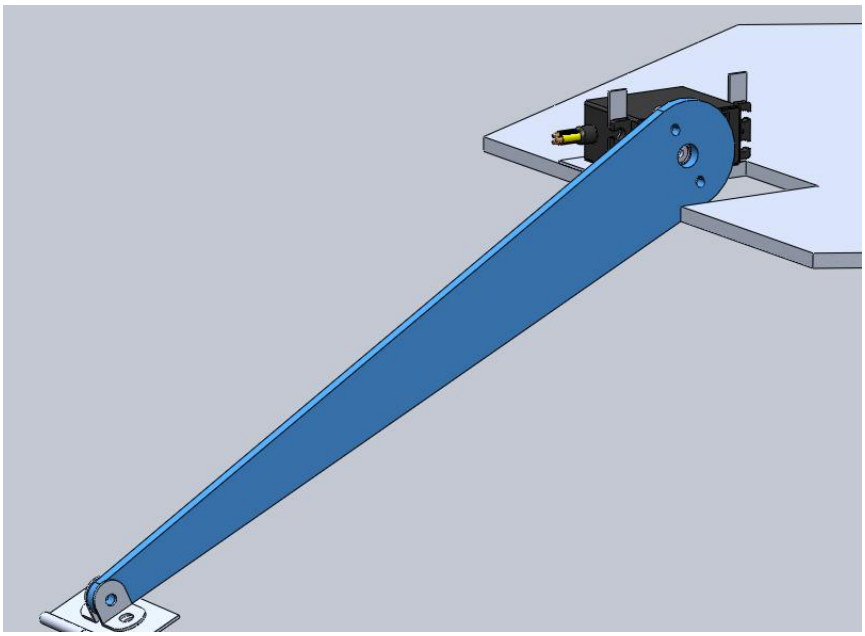
Para realizar la modelización se utilizo el software SolidWorks 2014.





A continuación se expone cada parte aislada del conjunto Robot.

Brazo Superior

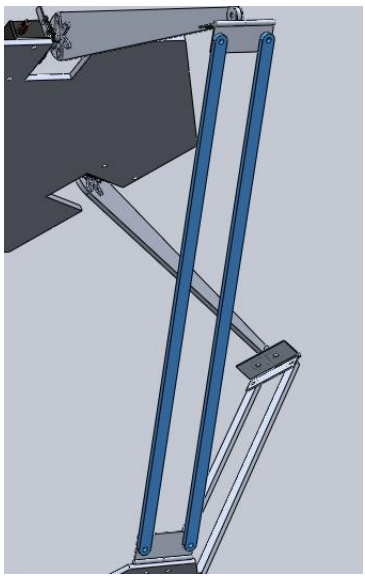


Material: Aluminio Aleación 6061

Masa: 71,75 gramos

Longitud: 250 mm

Brazo inferior

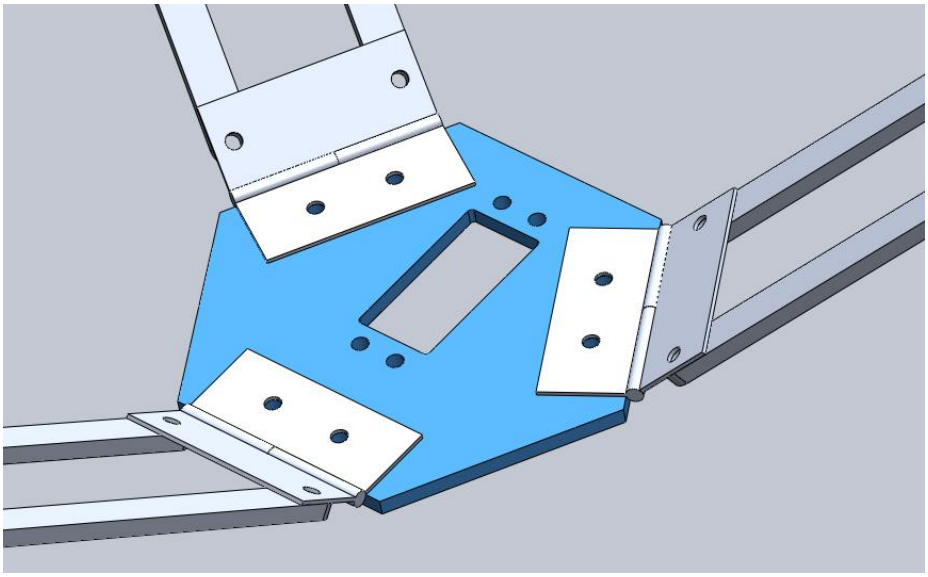


Material: Aluminio Aleación 6061

Masa: 87,55 gramos

Longitud: 400 mm

Plataforma Móvil



Material: Aluminio Aleación 6061

Masa: 83,43 gramos

Dinámica del Robot Delta en MatLab

A continuación se exponen los Scripts en MatLab sobre la dinámica del Robot Delta implementada.

```
function torques=calcular_torques(configuracion,q,qa,p,a)
  g=9.8;
  J=calcular_jacobiana(configuracion,q,p);

   Tb=((configuracion.mb)/3 + (2*configuracion.mfb)/3)*configuracion.la^2*eye(3)*qa;
   Tn=J'*(configuracion.mn+configuracion.mp+configuracion.mfb)*eye(3)*a;

   TGn=J'*(configuracion.mn+configuracion.mp+configuracion.mfb)*[0 0 -g]';
   TGb=configuracion.la*((configuracion.mb)/2+(2*configuracion.mfb)/3)*g*eye(3)*cos(q);

   torques=Tb+Tn- TGn - TGb;

end
```

Para calcular la Matriz Jacobiana utilizada en los Scripts se recurrió a un paper específico[1]:

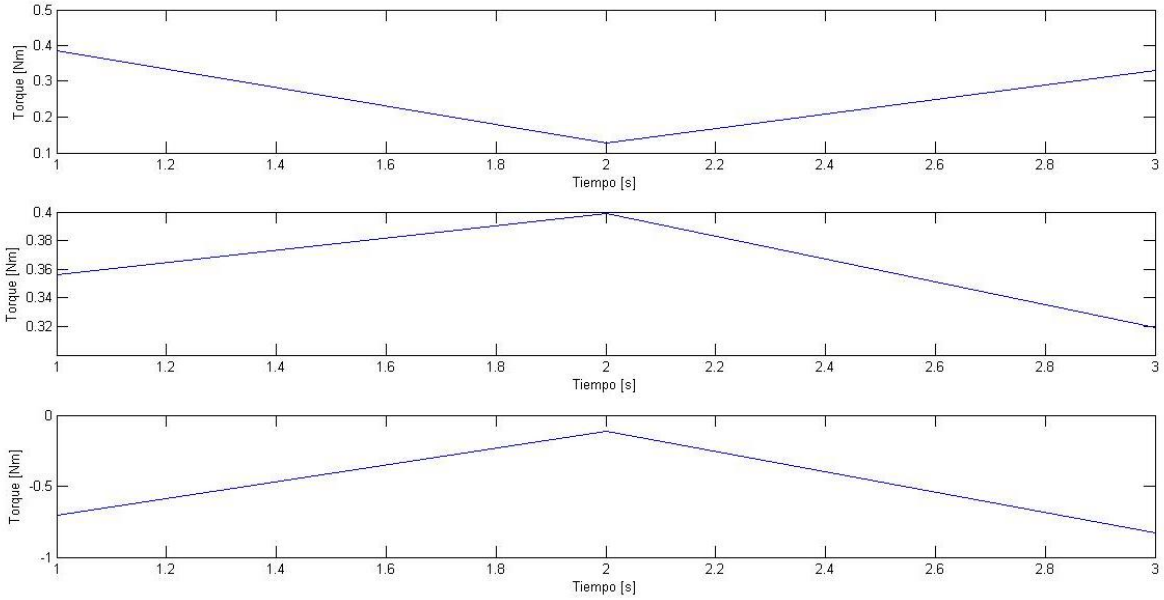
```
function jacobiana=calcular_jacobiana(configuracion,q,p)
dr=78;
A=zeros(3,3);
B=zeros(3,3);
for i=1:3
   alfa=(i-1)*(pi*120/180);

   A(i,1)=p(1)-((dr+configuracion.la*cos(q(i)))*cos(alfa));
   A(i,2)=p(2)-((dr+configuracion.la*cos(q(i)))*sin(alfa));
   A(i,3)=p(3)-(configuracion.la*sin(q(i)));

   B(i,i)=configuracion.la*((dr-(p(1)*cos(alfa))-(p(2)*sin(alfa)))*sin(q(i))+p(3)*cos(q(i)));
end
jacobiana=inv(A)*B;
```

[1] Ver Referencias y Bibliografia - RoboTenis: Optimal Design of a Parallel Robot with High Performance

La simulación en MatLab presenta los siguientes resultados, al calcular una trayectoria fijada arbitrariamente uniendo dos puntos, desde el punto (0.15, 0.15, 0.15) hasta el punto (0.19, 0.25, 0.15) cargando una masa de 0.100 kg:



Implementación del controlador de los actuadores mediante el uso de una FPGA

Módulo de generación de PWM

Este módulo genera una señal de PWM con un ciclo de trabajo especificado. La implementación realizada es genérica, por lo que permite cambiar la resolución de dicho ciclo según la necesidad de la aplicación.

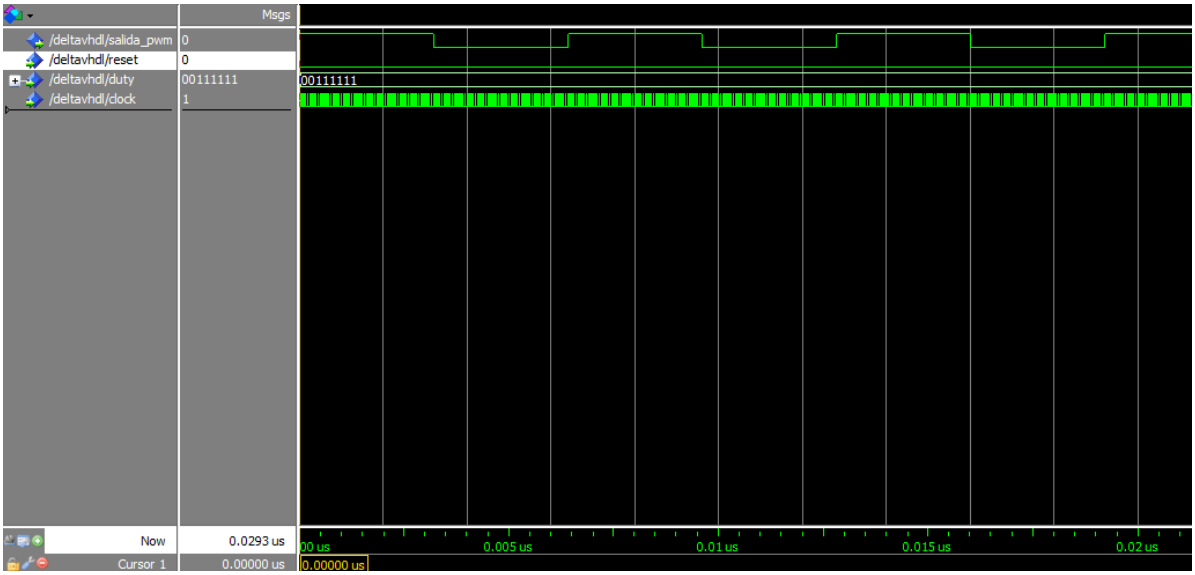
```
library IEEE;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_arith.all ;
use     ieee.numeric_std.all ;

entity generador_pwm is
       generic ( resolucion : integer :=7 );
port ( clock          :in std_logic;
       reset          :in std_logic;
       duty           :in std_logic_vector(resolucion downto 0);
       salida_pwm     :out std_logic
);
end generador_pwm;

architecture arq_generador_pwm of generador_pwm is
signal registro_contador  : integer;
begin
     process(clock,reset)
     begin
          if reset='1' then
             registro_contador<=0;
          elsif rising_edge(clock) then
             if registro_contador=((2**resolucion)-1) then
                registro_contador<=0;
             else
                registro_contador<= registro_contador+1;
             end if;
          end if;
    end process;

    process(registro_contador)
    begin
         if registro_contador>ieee.numeric_std.unsigned(duty) then
            salida_pwm<='0';
         else
             salida_pwm<='1';
         end if;
    end process;

end arq_generador_pwm;
```



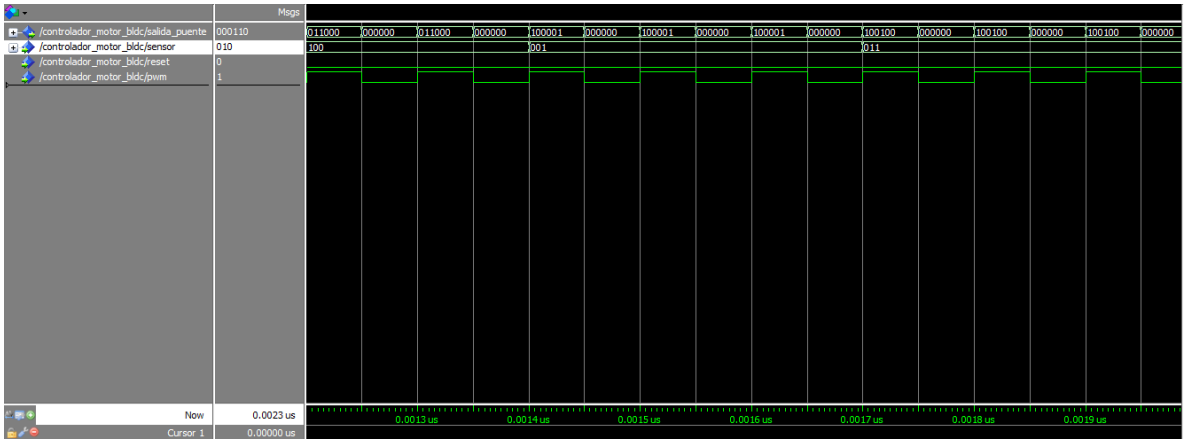
Módulo de control de motor

```
library IEEE;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_arith.all ;
use     ieee.numeric_std.all ;

entity controlador_motor_bldc is
port ( reset         :in std_logic;
       sensor        :in std_logic_vector(2 downto 0);
       pwm           :in std_logic;
       salida_puente :out std_logic_vector(5 downto 0)
);
end controlador_motor_bldc;

architecture arq_controlador_motor_bldc of controlador_motor_bldc is
begin
     process(pwm,reset,sensor)
     begin
          if reset='1' then
             salida_puente<="000000";
           else
             if pwm='1' then
                case sensor is
                  when "100"  => salida_puente<="011000";
                  when "101"  => salida_puente<="001001";
                  when "001"  => salida_puente<="100001";
                  when "011"  => salida_puente<="100100";
                  when "010"  => salida_puente<="000110";
                  when "110"  => salida_puente<="010010";
                  when others => salida_puente<="000000";
                  end case;
             else
                  salida_puente<="000000";
             end if;
          end if;
```

```
    end process;
end arq_controlador_motor_bldc;
```



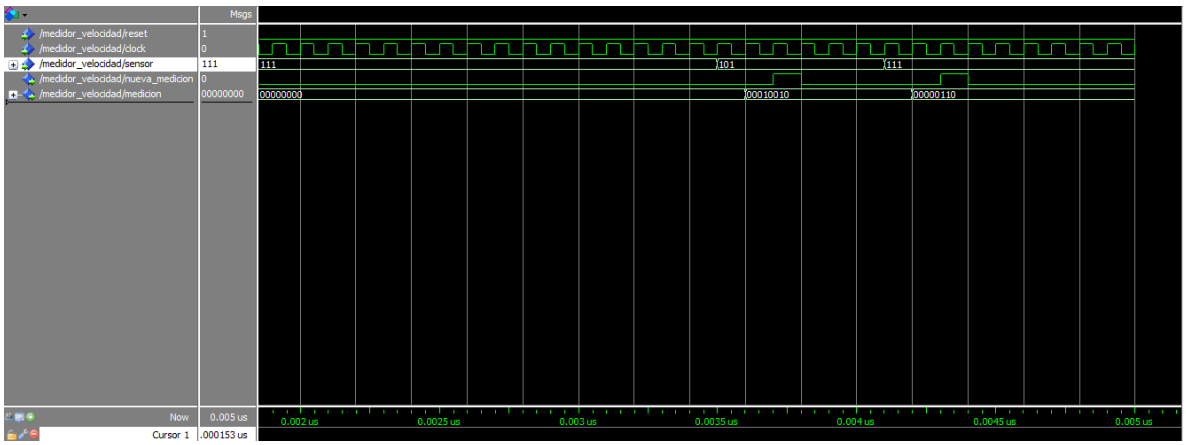
Medidor de velocidad

```
library IEEE;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_arith.all ;
use     ieee.numeric_std.all ;

entity medidor_velocidad is
       generic ( resolucion : integer :=7 );
port ( reset              :in std_logic;
       clock              :in std_logic;
       sensor             :in std_logic_vector(2 downto 0);
       nueva_medicion     :out std_logic;
       medicion           :out std_logic_vector(resolucion downto 0)
);
end medidor_velocidad;

architecture arq_medidor_velocidad of medidor_velocidad is
type estado is (iniciando,contando,cargando);
signal ultima_entrada_sensor:std_logic_vector(2 downto 0);
signal estado_medicion:estado;
signal contador:integer;
begin
     process(reset,clock)
     begin
          if reset='1' then
             nueva_medicion<='0';
             contador<=0;
             ultima_entrada_sensor<=sensor;
             estado_medicion<=contando;
           else
             if rising_edge(clock) then
                case estado_medicion is
                when iniciando=>
                     nueva_medicion<='0';
                     contador<=2;
                     ultima_entrada_sensor<=sensor;
                     estado_medicion<=contando;
```

```
                when contando=>
                    if ultima_entrada_sensor/=sensor then
                        medicion<=conv_std_logic_vector(contador+1, resolucion+1);
                        estado_medicion<=cargando;
                    else
                        contador<=contador+1;
                    end if;
                when cargando=>
                    nueva_medicion<='1';
                    estado_medicion<=iniciando;
                end case;
            end if;
        end if;
    end process;
end arq_medidor_velocidad;
```



Referencias y bibliografía

- Modeling and control of a Delta-3 robot.
  André Olsson
- robotkinematics-http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
- Design optimization, Impedance Control and Characterization of a Modified Delta Robot.
  Mehmet Alper ERGIN, Aykut Cihan SATICI, Volkan PATOGLU
- Delta robot: inverse, direct, and intermediate Jacobians.
  M. López, E. Castillo, G. García y A. Bashir.
- An Improved Approach to the Kinematics of Clavel's DELTA Robot.
  P.J. Zsombor-Murray Center for Intelligent Machines, McGill University, Montreal, Canada-November 27, 2009
- The mathematical model and direct kinematics solution analysis of Delta parallel robot.
  Jingjun Zhang -Lihong Shi Ruizhen Gao Chaoyang Lian
- TOWARDS A FULLY-PARALLEL 6 DOF ROBOT FOR HIGH-SPEED APPLICATIONS.
  F. Pierrot, A. Fournier and P. Dauchez
- A New Approach to the Design of a DELTA Robot with a Desired Workspace.
  XIN-JUN LIU-JINSONGWANG-KUN-KU OH and JONGWON KIM
- Dynamics and Control of a Novel 3-DoF Spatial Parallel Robot
  Mohsen Asgari, Mahdi Alinaghizadeh Ardestani, Mersad Asgari
- RoboTenis: Optimal Design of a Parallel Robot with High Performance
  Luis Angel Silva, J.M. Sebastian, R. Saltaren, R. Aracil and J. Sanpedro