



*Universidad
Tecnológica Nacional*

Facultad Regional Buenos Aires

Cátedra: Robótica - Plan 95A

Trabajo práctico Final

*Robot Tipo Stanford con 3
grados de libertad.*

Profesor: Ing. Hernan Giannetta.

JTP: Ing. Damián Granzella.

Integrantes: Juan Pablo Perelló. Leg (132651-0)

Nicolás Pimentel. Leg (119137-8)

Fecha de entrega: 27 / 08 / 2010.

Índice

Cinemática del Robot	Pag. 2
Cinemática directa	Pag. 4
Cinemática inversa	Pag. 10
Control Cinemático	Pag. 11
Simulación del área de trabajo	Pag. 16
Ejemplo 1	Pag. 17
Ejemplo 2	Pag. 18
Dinámica del robot	Pag. 25
Modelo dinámico (Lagrange - Euler)	Pag. 28
Modelo dinámico (Newton - Euler)	Pag. 30
Modelo dinámico (variables de estado)	Pag. 33
Torques utilizando toolbox HEMERO	Pag. 34
Selección de motores	Pag. 38
Código VHDL – PWM	Pag. 40
Código VHDL – Control del motor	Pag. 45
Resultados simulados VHDL	Pag. 53
Compilador	Pag. 58
Código fuente del Léxico	Pag. 61
Código fuente de la gramática	Pag. 64
Código fuente de la librería	Pag. 76

Introducción Teórica:

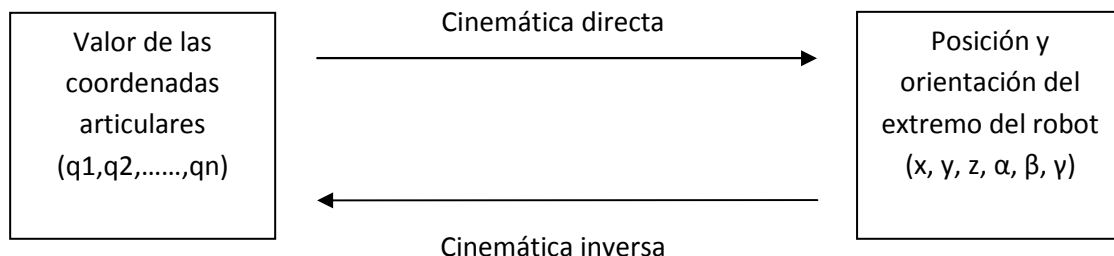
Cinemática del robot

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

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

Denavit y Hartenberg propusieron un método sistemático para describir y representar la geometría espacial de los elementos de una cadena cinemática, y en particular de un robot, con respecto a un sistema de referencia fijo. Este método utiliza una matriz de transformación homogénea para describir la relación espacial entre los elementos rígidos adyacentes, reduciéndose el problema cinemático directo a encontrar una matriz de transformación homogénea 4×4 que relacione la localización espacial del extremo del robot con respecto al sistema de coordenadas de su base.

Por otra parte, la cinemática del robot trata de encontrar las relaciones entre las velocidades del movimiento de las articulaciones y las del extremo. Esta relación viene dada por el **modelo diferencial** expresado mediante la matriz Jacobiana.



El problema cinemático directo

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

Resolución del problema cinemático directo mediante matrices de transformación homogénea

La resolución del problema cinemático directo consiste en encontrar las relaciones que permiten conocer la localización espacial del extremo del robot a partir de los valores de sus coordenadas articulares.

Así, si se han escogido coordenadas cartesianas y ángulos de Euler para representar la posición y orientación del extremo de un robot de 6 grados de libertad, la solución del problema cinemático directo vendrá dada por las relaciones:

$x=f_x(q_1,q_2,q_3,q_4,q_5,q_6)$	$\alpha=f_\alpha(q_1,q_2,q_3,q_4,q_5,q_6)$
$y=f_y(q_1,q_2,q_3,q_4,q_5,q_6)$	$\beta=f_\beta(q_1,q_2,q_3,q_4,q_5,q_6)$
$z=f_z(q_1,q_2,q_3,q_4,q_5,q_6)$	$\gamma=f_\gamma(q_1,q_2,q_3,q_4,q_5,q_6)$

De acuerdo al robot que nos toca caracterizar en este trabajo práctico final vamos a tener 3 grados de libertad, con lo que nos quedará solo x , y , z .

Para la realización del trabajo práctico nos vamos a plantear por el método sistemático basado en la utilización de las matrices de transformación homogénea.

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. A cada eslabón se le puede asociar un sistema de referencia solidario a él y , utilizando las transformaciones homogéneas, es posible representar las rotaciones y translaciones relativas entre los distintos eslabones que componen el robot. Normalmente la matriz de transformación homogénea que representa la posición y orientación relativa entre los sistemas asociados a dos eslabones consecutivos del

robot se suele denominar matriz ${}^{i-1}A_i$. Así pues, 0A_1 describe la posición y orientación del sistema de referencia solidario al primer eslabón con respecto al sistema de referencia solidario a la base, 1A_2 describe la posición y orientación del segundo eslabón respecto del primero, etc. Del mismo modo denominando 0A_k a las matrices resultantes del producto de las matrices ${}^{i-1}A_i$ con i desde 1 hasta k , se puede representar de forma total o parcial la cadena cinemática que forma el robot. Así, por ejemplo, la posición y orientación del sistema solidario con el segundo eslabón del robot respecto al sistema de coordenadas de la base se puede expresar mediante la matriz 0A_2 :

$${}^0A_2 = {}^0A_1 \cdot {}^1A_2.$$

Cuando se consideran todos los grados de libertad, a la matriz 0A_n se la suele denominar T . Así, dado un robot de seis grados de libertad, se tiene que la posición y orientación del eslabón final vendrá dada por la matriz T :

$$T = {}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6.$$

Aunque para describir la relación que existe entre dos elementos contiguos se puede hacer uso de cualquier sistema de referencia ligado a cada elemento, la forma habitual que se suele utilizar en robótica es la representación de Denavit-Hartenberg (D-H).

Estos propusieron en 1995 un método matricial que permite establecer de manera sistemática un sistema de coordenadas $\{S_i\}$ ligado a cada eslabón i de una cadena articulada, pudiéndose determinar a continuación las ecuaciones cinemáticas de la cadena completa.

Según la representación de (D-H), escogiendo adecuadamente los sistemas de coordenadas asociados a cada eslabón, será posible pasar de uno al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Estas transformaciones básicas consisten en una sucesión de rotaciones y translaciones que permiten relacionar el sistema de referencia del objeto i con el sistema del elemento $i-1$. Las transformaciones en cuestión son las siguientes:

- 1) Rotación alrededor del eje z_{i-1} un ángulo θ_i .
- 2) Translación a lo largo de z_{i-1} una distancia d_i ; vector $d_i(0,0,d_i)$.
- 3) Translación a lo largo de x_i una distancia a_i ; vector $a_i(0,0,a_i)$.
- 4) Rotación alrededor del eje x_i un ángulo α_i .

Dado que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que:

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

$${}^{i-1}A_i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \cdot \sin \theta_i & \sin \alpha_i \cdot \sin \theta_i & a_i \cdot \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cdot \cos \theta_i & -\sin \alpha_i \cdot \cos \theta_i & a_i \cdot \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [\text{expresión 1}]$$

Donde θ_i , a_i , d_i , α_i son parámetros D-H del eslabón i . De este modo, basta con identificar los parámetros θ_i , a_i , d_i , α_i para obtener las matrices A y relacionar así todos y cada uno de los eslabones del robot.

En nuestro caso, vamos a utilizar el Algoritmo de Denavit y Hartenberg para la obtención del modelo cinemático directo para la resolución del trabajo práctico.

Reglas que establece el método de D-H para la obtención del modelo cinemático directo:

D-H 1. Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se enumerará 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 ésta 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 $\{S_0\}$ en cualquier punto del eje z_0 . Los ejes x_0 e y_0 se situarán de modo que formen un sistema dextrógiro con z_0 .

D-H 6. Para i de 1 a $n-1$, situar el sistema $\{S_i\}$ (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 $\{S_i\}$ en el punto de corte. Si fuesen paralelos $\{S_i\}$ 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 $\{S_n\}$ 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 θ_i 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 d_i como la distancia, medida a lo largo de z_{i-1} , que habría que desplazar $\{S_{i-1}\}$ para que x_i y x_{i-1} quedasen alineados.

D-H 12. Obtener a_i como la distancia medida a lo largo de x_i (que ahora coincidirá con x_{i-1}) que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

D-H 13. Obtener α_i como el ángulo que habría que girar entorno a x_i (que ahora coincidiría con x_{i-1}), para que el nuevo $\{S_{i-1}\}$ coincidiese totalmente con $\{S_i\}$.

D-H 14. Obtener las matrices de transformación ${}^{i-1}A_i$.

D-H 15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot $T = {}^0A_1 \cdot {}^1A_2 \cdot \dots \cdot {}^{n-1}A_n$.

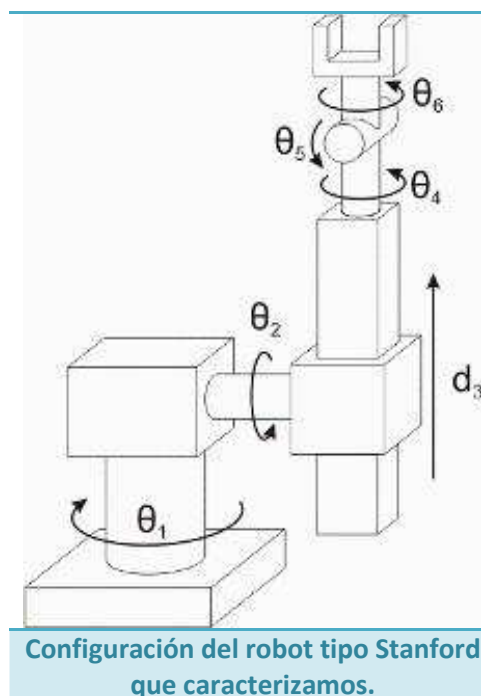
D-H 16. La matriz T define la orientación (submatriz de rotación) y posición (submatriz de translación) del extremo referido a la base en función de las n coordenadas articulares.

Los cuatro parámetros de D-H (θ_i , d_i , a_i , α_i) dependen únicamente de las características geométricas de cada eslabón y de las articulaciones que lo unen con el anterior y siguiente.

Una vez obtenidos los parámetros de D-H, el cálculo de las relaciones entre los eslabones consecutivos del robot es inmediato, ya que vienen dadas por las matrices A , que se calculan según la expresión general [expresión 1]. Las relaciones entre eslabones no consecutivos vienen dadas por las matrices T que se obtienen como producto de un conjunto de matrices A .

Obtenida la matriz T , ésta expresará la orientación y posición del extremo del robot en función de sus coordenadas articulares, con lo que quedará resuelto el problema cinemático directo.

A continuación, en base a la configuración del robot del tipo Stanford vamos a obtener los parámetros D-H con los que desarrollaremos el resto de la práctica.



Parámetros de D-H para el robot del tipo Stanford

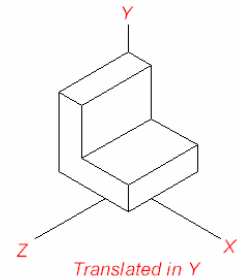
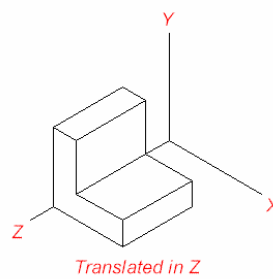
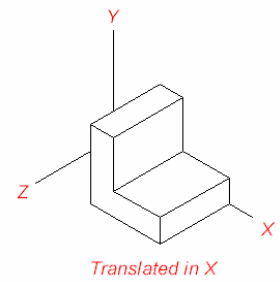
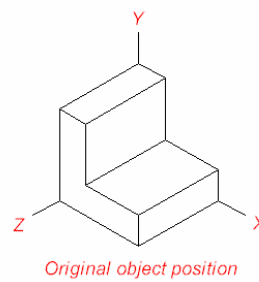
Articulación	θ	d	a	α
1	Θ_1	0	0	0
2	Θ_2	d_2	0	-90
3	0	d_3	0	90

A continuación se exponen las diferentes matrices en forma canónica que vamos a utilizar:

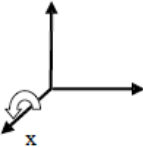
Translación en 3 dimensiones – Forma canónica -

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \end{aligned}$$

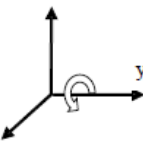
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



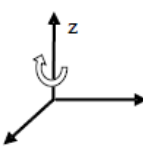
Rotación en 3 dimensiones – Forma canónica –



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Rotación en x}$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Rotación en y}$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Rotación en z}$$

Las anteriores son las transformaciones básicas con las que sacamos la “expresión 1” que vamos a utilizar para sacar las matrices correspondientes al robot .

$${}^0A_1 = \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 & 0 & 0 \\ \sin \theta_1 & \cos \theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^1A_2 = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ -\sin \theta_2 & \cos \theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -d_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3$$

$$T = \begin{bmatrix} \cos \theta_1 \cos \theta_2 & -\sin \theta_1 & 1 & \cos \theta_1 \sin \theta_2 d_3 - \sin \theta_1 d_2 \\ \sin \theta_1 \cos \theta_2 & C1 & \sin \theta_1 \sin \theta_2 & \sin \theta_1 \sin \theta_2 d_3 + \cos \theta_1 d_2 \\ -\sin \theta_2 & 0 & -\cos \theta_2 & -\cos \theta_2 d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_{x3} & o_{x3} & a_{x3} & p_{x3} \\ n_{y3} & o_{y3} & a_{y3} & p_{y3} \\ n_{z3} & o_{z3} & a_{z3} & p_{z3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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.

En las ecuaciones resultantes queda reflejado el valor de la posición (p_{x3} , p_{y3} , p_{z3}) y orientación (n , o , a) del extremo del robot en función de las coordenadas articulares (1, 2, 3).

Por lo tanto las ecuaciones resultantes describirán las coordenadas finales del extremo del robot en base a sistema de referencia ubicado en la base del mismo.

$$X = \cos \theta_1 \sin \theta_2 d_3 - \sin \theta_1 d_2$$

$$Y = \sin \theta_1 \sin \theta_2 d_3 + \cos \theta_1 d_2$$

$$Z = -\cos \theta_2 d_3$$

CINEMÁTICA INVERSA

El objetivo del problema cinemática inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot $q=[q_1, q_2, \dots, q_n]^T$ para que su extremo se posicione y oriente según una determinada localización espacial.

Así como es posible abordar el problema cinemática directo de una manera sistemática a partir de la utilización de matrices de transformación homogéneas, e independientemente de la configuración del robot, no ocurre lo mismo con el problema cinemática inverso, siendo el procedimiento de obtención de las ecuaciones fuertemente independiente de la configuración del robot.

Se han desarrollado algunos procedimientos genéricos susceptibles de ser programados, de modo que una computadora pueda, a partir del conocimiento de la cinemática del robot, obtener la cuplas de valores que posicionan y orientan su extremo. El inconveniente de estos procedimientos es que se trata de procesos numéricos iterativos cuya velocidad de convergencia e incluso su convergencia en sí no está siempre garantizada.

A la hora de resolver el problema cinemático inverso es mucho más adecuado encontrar una solución cerrada. Esto es, encontrar una relación matemática explícita de la forma:

$$q_k = f_k(x, y, z, \alpha, \beta, \gamma)$$

$$k = 1 \dots n \quad \text{Grados de Libertad}$$

Este tipo de solución presenta, entre otras, las siguientes ventajas:

1. En muchas ocasiones, el problema cinemática inverso ha de resolverse en tiempo real, por ejemplo, en el seguimiento de una determinada trayectoria. Una solución del tipo iterativo no garantiza tener la solución en el momento adecuado.

2. Al contrario de lo que ocurría en el problema cinemática directo, con cierta frecuencia la solución del problema cinemática inverso no es única. Existiendo diferentes cuplas que posicionan y orientan al extremo del robot del mismo modo. En estos casos una solución cerrada permite incluir determinadas reglas o restricciones que aseguren que la solución obtenida sea la más adecuada de entre las posibles.

No obstante, a pesar de la dificultades comentadas, la mayor parte de los robots poseen cinemáticas relativamente simples, que facilitan en cierta medida la resolución de su problema cinemático inverso. Por ejemplo, si se consideran sólo los tres primeros grados de libertad de muchos robots, éstos tienen una estructura planar, esto es, los tres primeros elementos quedan contenidos en un plano. Asimismo, en muchos robots, se da la circunstancia de que los tres grados de libertad últimos, dedicados fundamentalmente a orientar el extremo del robot, corresponden a giros sobre ejes que se cortan en un punto. De nuevo esta situación facilita el cálculo de la cupla correspondiente a la posición y orientación deseadas. Por lo tanto, para los casos citados y otros, es posible establecer ciertas pautas generales que permitan plantear y resolver el problema cinemática inverso de una manera sistemática.

Los métodos geométricos permiten generalmente obtener los valores de las primeras variables articulares, que son las que consiguen posicionar el robot. Para ello se utilizan relaciones trigonométricas y geométricas sobre los elementos del robot. Se suele recurrir a la resolución de triángulos formados por los elementos y articulaciones del robot.

Como alternativa para resolver el mismo problema se puede recurrir a manipular directamente las ecuaciones correspondientes al problema cinemático directo. Es decir, puesto que se establece la relación

$$\begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} = [t_{ij}]$$

donde los elementos t_{ij} son función de las coordenadas articulares, es posible pensar que mediante ciertas combinaciones de las 12 ecuaciones planteadas, se puedan despejar las n variables articulares q_i en función de las componentes de los vectores n , a , o y p .

Control Cinemático

El control cinemático establece cuáles son las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para lograr los objetivos fijados por el usuario, como punto destino trayectoria cartesiana del efector final del robot, tiempo invertido por el usuario, etc. Estas trayectorias se seleccionarán atendiendo a las restricciones físicas propias de los accionamientos y a ciertos criterios de calidad de trayectoria, como suavidad y precisión de la misma.

Funciones del Control Cinemático

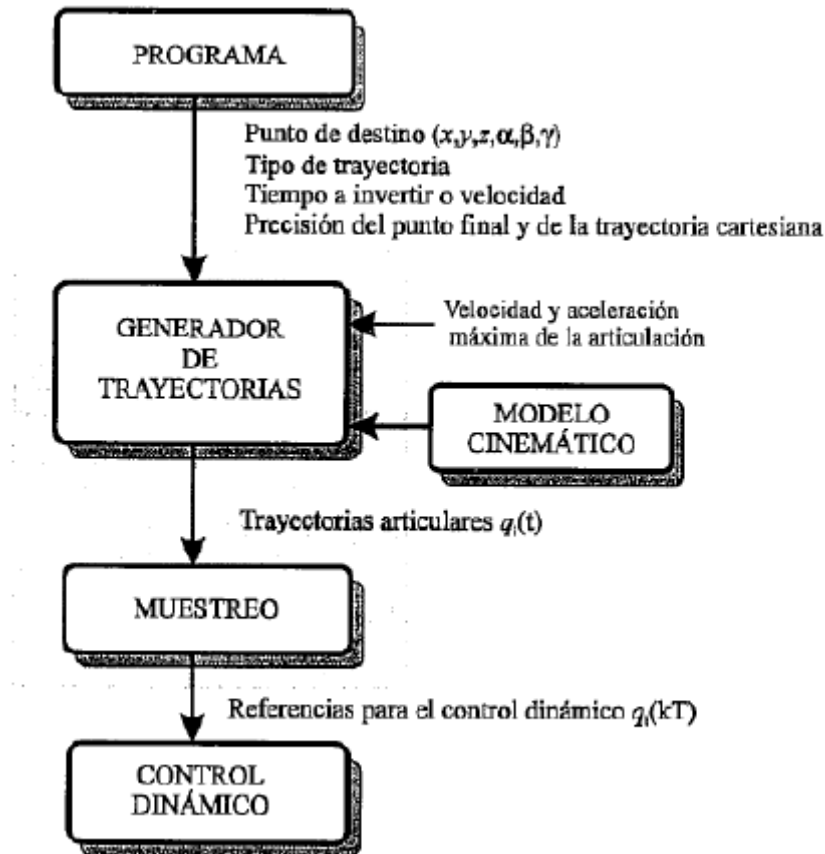
De manera general, el control cinemático deberá realizar las siguientes funciones:

1. Convertir la especificación de movimiento dada en el programa en una trayectoria analítica en espacio artesaiano.
2. Muestrear la trayectoria cartesiana obteniendo un número finito de puntos de dicha trayectoria.
3. Utilizando la transformación homogénea inversa, convertir cada uno de estos puntos en sus correspondientes coordenadas articulares. Debe tenerse en cuenta aquí la posibilidad de soluciones múltiples de la transformación homogénea inversa, así como la posibilidad de

ausencia de solución y puntos singulares, de modo que se asegure la continuidad de la trayectoria.

4. Interpolación de los puntos articulares obtenidos, generando para cada variable particular una expresión $q_i(t)$ que pase o se aproxime por ellos de modo que, siendo una trayectoria realizable por los actuadores, se transforme en una trayectoria cartesiana lo más próxima a la especificada por el usuario.

5. Muestreo de la trayectoria articular para generar referencias al control dinámico.



Como se describió anteriormente, la matriz Jacobiana establece las relaciones diferenciales entre variables articulares y cartesianas, que se puede expresar como:

$$\dot{j}(t) = J(q)\dot{q}(t)$$

Esta relación es lineal y para una localización q determinada permite conocer las velocidades cartesianas a partir de las correspondientes articulares. Además, suponiendo $J(q)$ invertible, lo que ocurrirá siempre que las dimensiones del espacio articular y de la tarea sean iguales y q no sea un punto singular, se podrá expresar:

$$\dot{q}(t) = J(q)^{-1} \dot{j}(t)$$

La implementación de este procedimiento en la unidad de control de un robot, exigirá la discretización de la trayectoria $j(t)$ y su derivada en un número finito de puntos que posteriormente, y tras ser convertidos en velocidades articulares deberán ser interpolados.

Por lo tanto, para estudiar el control cinemático de un robot se deberá fundamentalmente conocer qué tipo de interpoladores pueden ser eficaces para unir los puntos articulares por los que se quiere pasar. Asimismo, será conveniente establecer criterios para seleccionar cuántos y qué puntos serán muestreados en la trayectoria del espacio cartesiano.

Tipos de Trayectorias

Para realizar una turca determinada el robot debe moverse desde un punto inicial a un punto final. Este movimiento puede ser realizado según infinitas trayectorias espaciales. De todas ellas hay algunas que, bien por su sencillez de implementación por panel del control cinemático o bien por su utilidad y aplicación a diversas tareas, son las que en la práctica incorporan los robots comerciales. De este modo, puede encontrarse que los robots dispongan de trayectorias punto a punto, coordinadas y continuas.

Trayectorias Punto a Punto

En este tipo de trayectorias cada articulación evoluciona desde su posición inicial a la final sin realizar consideración alguna sobre el estado o evolución de las demás articulaciones.

Normalmente, cada actuador ira de llevar a su articulación al punto de destino en el menor tiempo posible, pudiéndose distinguir dos casos: movimiento eje a eje y movimiento simultáneo de ejes.

Movimiento de Ejes

Sólo se mueve un eje cada vez. Comenzará a moverse la primera articulación, y una vez que esta haya alcanzado su punto final lo hará la segunda, y así sucesivamente. Este tipo de movimiento da obviamente como resultado un mayor tiempo de ciclo, teniendo como única ventaja un menor consumo de potencia instantánea por parte de los actuadores.

Movimiento simultáneo de ejes

En este caso todos los actuadores comienzan simultáneamente a mover las articulaciones del robot a una velocidad específica para cada una de ellas. Dado que la distancia a recorrer y las velocidades serán en general diferentes, cada una acabará su movimiento en un instante diferente.

El movimiento del robot no acabará hasta que se alcance definitivamente el punto final, lo que se producirá cuando el eje que más tarde concluya su movimiento. De esta manera, el tiempo total invertido en el movimiento coincidirá con el del eje que más tiempo emplee en realizar su movimiento particular, pudiéndose dar la circunstancia de que el resto de los actuadores hayan forzado su movimiento a una velocidad y aceleración elevada, viéndose obligados finalmente a esperar a la articulación más lenta.

Por los motivos expuestos, las trayectorias punto a punto no están implementadas salvo en robots muy simples o con unidades de control muy limitadas

Trayectorias coordinadas o isócronas

Para evitar que algunos actuadores trabajen forzando sus velocidades y aceleraciones.

Teniendo que esperar después la conclusión del movimiento de la articulación más lenta, puede hacer un cálculo previo, averiguando cuál es esta articulación y qué tiempo invertirá. Se ralentizará entonces el movimiento del resto de los ejes para que inviertan el mismo tiempo en su movimiento, acabando todos ellos simultáneamente. Se tiene así que todas las articulaciones se coordinan comenzando y acabando su movimiento a la vez, adaptándose todas a la más lenta.

El tiempo total invertido en el movimiento es el menor posible y no se piden aceleraciones y velocidades elevadas a los actuadores de manera inútil. Desde el punto de vista del usuario la trayectoria que describe el extremo del robot no es significativa, siendo ésta impredecible. Aunque como es obvio, un conocimiento del modelo y control cinemático del robot permitiría su cálculo.

Trayectoria Continua

Cuando se pretende que la trayectoria que sigue el extremo del robot sea conocida por el usuario (trayectoria en el espacio cartesiano o de la tarea), es preciso calcular de manera continua las trayectorias articulares.

Típicamente, las trayectorias que el usuario pretende que el robot describa son trayectorias en línea recta o en arco de círculo.

El resultado será que cada articulación sigue un movimiento aparentemente caótico con posibles cambios de dirección y velocidad y sin coordinación con el resto de las articulaciones.

Sin embargo el resultado será que el extremo del robot describirá la trayectoria deseada.

Generación de Trayectorias

Normalmente el usuario del robot indica el movimiento que éste debe realizar especificando las localizaciones espaciales por las que debe pasar el extremo, junto con otros datos, como instantes de paso, velocidades o tipos de trayectorias. Así, por ejemplo, es frecuente especificar que el robot debe ir de un punto inicial hasta otro final, siguiendo en cartesianas una línea recta a velocidad constante.

Puesto que estos puntos están excesivamente separados es preciso seleccionar puntos intermedios suficientemente cercanos como para que el control del robot consiga ajustar no sólo el punto final al especificado, sino también la trayectoria seguida a la indicada en el programa.

Para ello es preciso establecer un interpolador entre las localizaciones expresadas en el espacio de la tarea que dará como resultado una expresión analítica de la evolución de cada coordenada. La interpolación más frecuente es la lineal, en la que cada coordenada evoluciona a velocidad constante desde su valor inicial j^i hasta el final j^f

$$j(t) = (j^f - j^i) \frac{t - t_i}{t_f - t_i} + j^i$$

donde t^i y t^f son los instantes de tiempo en los que se pretende alcanzar la localización inicial y final, respectivamente.

Interpolación de Trayectorias

Como se ha indicado, una de las funciones del control cinemático es la de unir una sucesión de puntos en el espacio articular por los que se quiere que pasen las articulaciones del robot en un instante determinado. Además, junto con las condiciones de posición-tiempo, es conveniente añadir restricciones en la velocidad y aceleración de paso por los puntos, de manera que se asegure la suavidad de la trayectoria y se limiten las velocidades y aceleraciones máximas. Estas restricciones garantizarán que los actuadores están capacitados para implementar la trayectoria final.

Para ello deberá seleccionarse algún tipo de función (frecuentemente polinómica) cuyos parámetros o coeficientes se ajustarán al imponer las condiciones de contorno: posiciones, velocidades y aceleraciones. En la selección de esta función debe considerarse que tanto el cálculo de sus parámetros, como su posterior utilización para generar puntos de consigna al control dinámico, debe hacerse en tiempo real, por lo que la simplicidad de la función será un factor a valorar.

Se van a presentar a continuación las funciones interpoladoras utilizadas con mayor frecuencia.

Cada una de ellas ha sido desarrollada para un solo grado de libertad, debiendo quedar claro que el mismo cálculo deberá repetirse para cada uno de los grados de libertad del robot. Cabe indicar que si bien las técnicas de interpolación que a continuación se describen están planteadas para el espacio articular son igualmente aplicables para el espacio de la tarea.

Interpoladores Lineales

Supóngase que se pretende que una de las articulaciones q del robot, pase sucesivamente por los valores q_i en los instantes t_i . Una primera solución a este problema consistiría en mantener constante la velocidad de movimiento entre cada 2 valores sucesivos ($q^{i-1}; q_i$) de la articulación. La trayectoria entre dos puntos $q^{i-1}; q_i$ sería entonces:

$$q(t) = (q^i - q^{i-1}) \frac{t - t^{i-1}}{T} + q^{i-1}$$

Como es obvio, esta trayectoria asegura la continuidad de la posición, pero origina saltos bruscos en la velocidad \dot{q} de la articulación, y consecuentemente precisa de aceleraciones \ddot{q} de valor infinito lo que en la práctica no es posible.

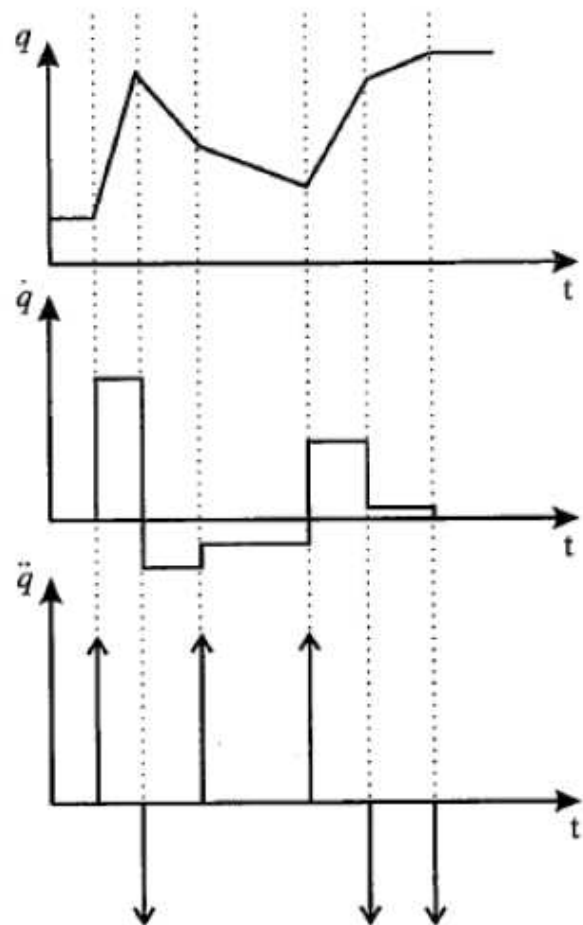
La selección de los instantes de paso i por los puntos q podrá haberse hecho según los diferentes criterios expuestos son:

1. Intentando que cada articulación q alcance el punto de destino en el menor tiempo posible sin considerar las demás articulaciones, lo que resultará en velocidades \dot{q} constantes e iguales a la máxima.
2. Ajustando los instantes de paso a los de la articulación que más tiempo precise, resultando movimientos coordinados.
3. Seleccionando los tiempos a partir de las especificaciones dadas en el espacio de la tarea de modo que el extremo del robot describa una trayectoria predeterminada.

Interpoladores Cúbicos

Para asegurar que la trayectoria que une los puntos por los que tiene que pasar la articulación considerada presente continuidad en velocidad, puede recurrirse a utilizar un polinomio de grado 3 que una cada pareja de puntos adyacentes. De este modo, al tener cuatro parámetros disponibles se podrán imponer cuatro condiciones de contorno, dos de posición y dos de velocidad. Los valores de las velocidades de paso por cada punto deben por tanto ser conocidas a priori.

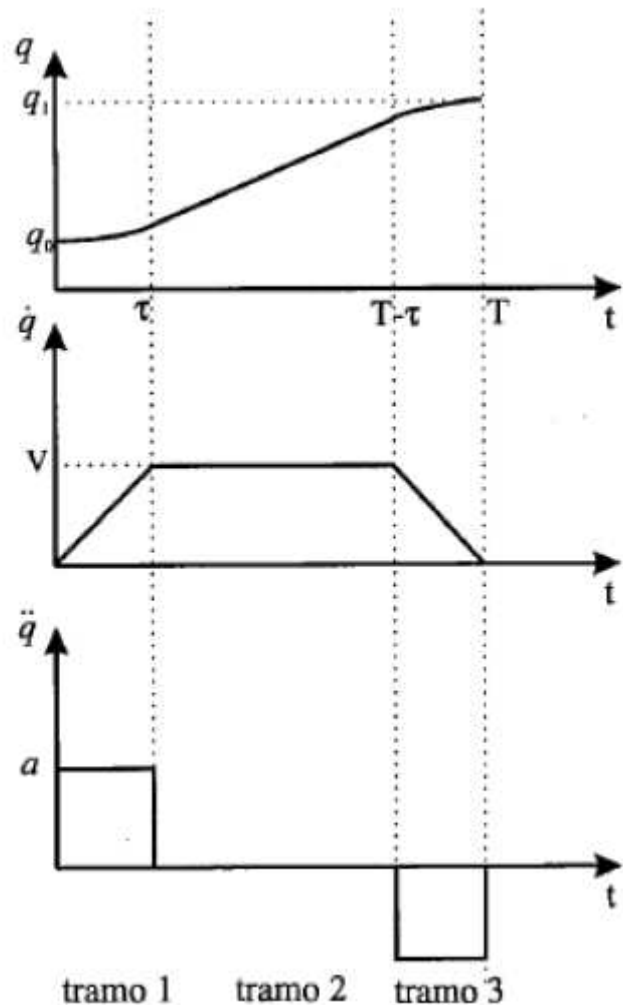
Esta selección no es obvia y se tratará posteriormente. Se consigue así una trayectoria compuesta por una serie de polinomios cúbicos, cada uno válido entre dos puntos consecutivos. Este conjunto de polinomios concatenados, escogidos de modo que exista continuidad en la posición y velocidad, se denominan splines (cúbicos, por ser de tercer grado).



Interpoladores a Tramos

En los interpoladores vistos hasta el momento, se utiliza un polinomio de un grado determinado (1 o 3) para unir dos puntos consecutivos de la trayectoria. El uso de polinomios de tercer grado permite asegurar que el polinomio pasa por los dos puntos y al mismo tiempo permite imponer los valores de velocidad de paso por los mismos. Sin embargo, al contrario de lo que ocurre con el interpolador de primer grado (lineal), la velocidad de la articulación durante el recorrido está variando continuamente, lo que exige un control continuo de la misma.

Una alternativa que proporciona una solución intermedia consiste en descomponer en tres tramos consecutivos la trayectoria que une dos puntos q_0 , q_1 . En el tramo central se utiliza un interpolador lineal, y por lo tanto la velocidad se mantiene constante, no siendo preciso imprimir aceleración alguna al actuador. En los tramos inicial y final se utiliza un polinomio de segundo grado, de modo que en el tramo 1 la velocidad varíe linealmente desde la velocidad de la trayectoria anterior a la de la presente y en el tramo 3 varía desde la velocidad de la trayectoria presente hasta la de la siguiente. Se tiene entonces que en los tramos inicial y final la aceleración toma valores constantes distintos de cero, mientras que en el tramo intermedio la aceleración es nula.



Simulación en Matlab para conocer el área de trabajo del robot

Ya obtenida la matriz de transformación homogénea, podremos utilizarla para obtener, junto con la ayuda de la herramienta Matlab, el entorno y volumen de trabajo en la cual el manipulador de 3 GDL podrá moverse y trabajar sobre la misma. La idea para obtener el volumen físico de trabajo de manipulador, es generar una gran cantidad de trayectoria, realizando movimientos del extremo final muy cercanas unas a las otras, y modulando todas sus articulaciones desde su punto mínimo hasta el máximo, tratando de obtener todas las combinaciones posibles. Sabemos que las combinaciones son infinitas, ya que los puntos intermedios de cada articulación son infinitos, pero servirá hacer este tipo de análisis nos sirve para darnos una certera noción del espacio de trabajo.

Ejemplo 1:

L1 = 50;
L2=20;
 $0 \leq L3 \leq 15$;

$(-3/4)\pi \leq \theta_1 \leq (3/4)\pi$;
 $-\pi/2 \leq \theta_2 \leq \pi/2$;
 $\theta_3=0$;

```
%*****
%UTN - FACULTAD REGIONAL BUENOS AIRES
%MATERIA: ROBOTICA.
%TESIS FINAL: ROBOT TIPO STANFORD DE 3 GDL.
%PROFESOR: ING. HERNAN GIANNETTA.
%JTP: ING. DAMIAN GRANZELLA.
%ALUMNOS: PERELLO JUAN PABLO , PIMENTEL NICOLAS.
%*****

%*****
%Inicio del Script para generar volumen de trabajo del Robot tipo
Stanford
%*****
L1 = 50;
L2 = 20;
XYZ = [];
UVW = [0;0;0;1];
% a y b los utilizo como auxiliar para barrer de izq a der y de der a
izq
% alternadamente.
a = 0;
b = 0;
indx = 1;
for L3 = 0:3:15 %L3 varío de 0 a 15 en pasos de a 3.
    if a == 0, a = 1;
    else a = 0;
    end;
    if a == 0,
        for q1 = -pi*3/4:pi/50:pi*3/4 %thetal varío de (-3/4)pi a
(3/4)pi
            if b == 0, b = 1;
            else b = 0;
            end;
            if b == 0,
                for q2 = -pi*1/2:pi/10:pi*1/2
                    A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                    A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                    A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                    T = A1 * A2 * A3 ;
                    XYZ(:,indx) = T * UVW;
                    indx = indx + 1;
                end
            else

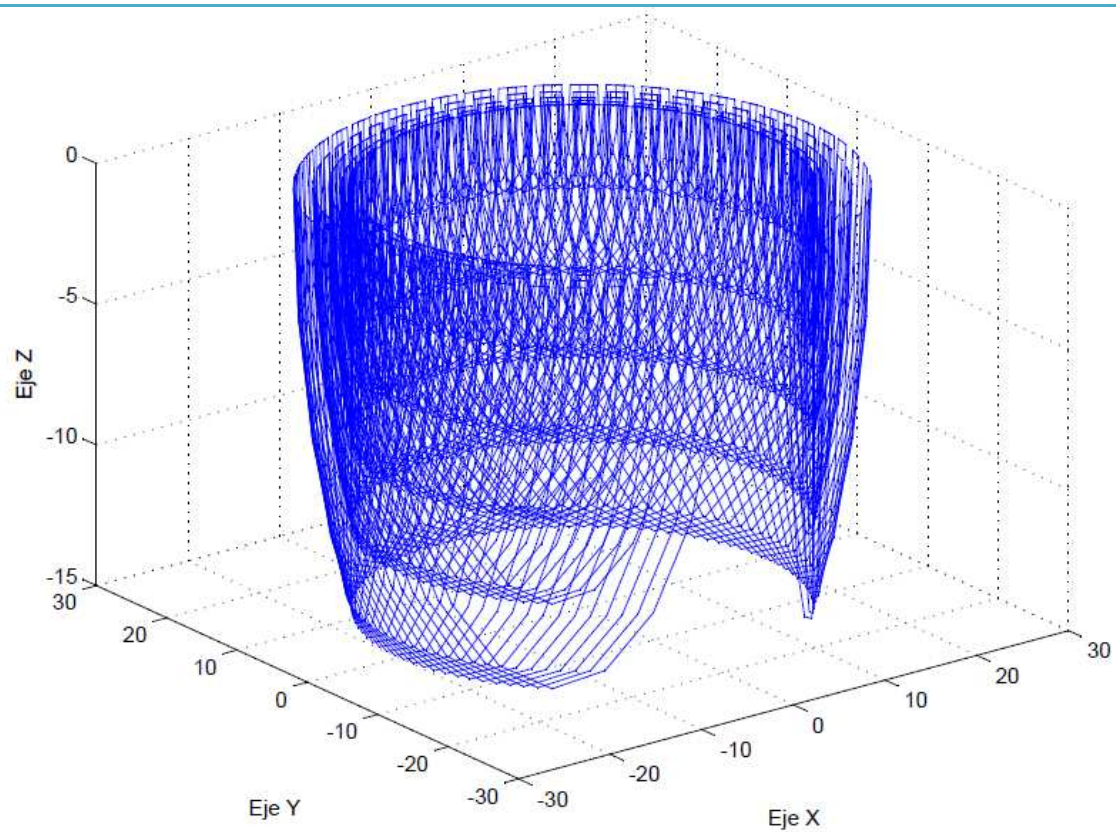
```

```

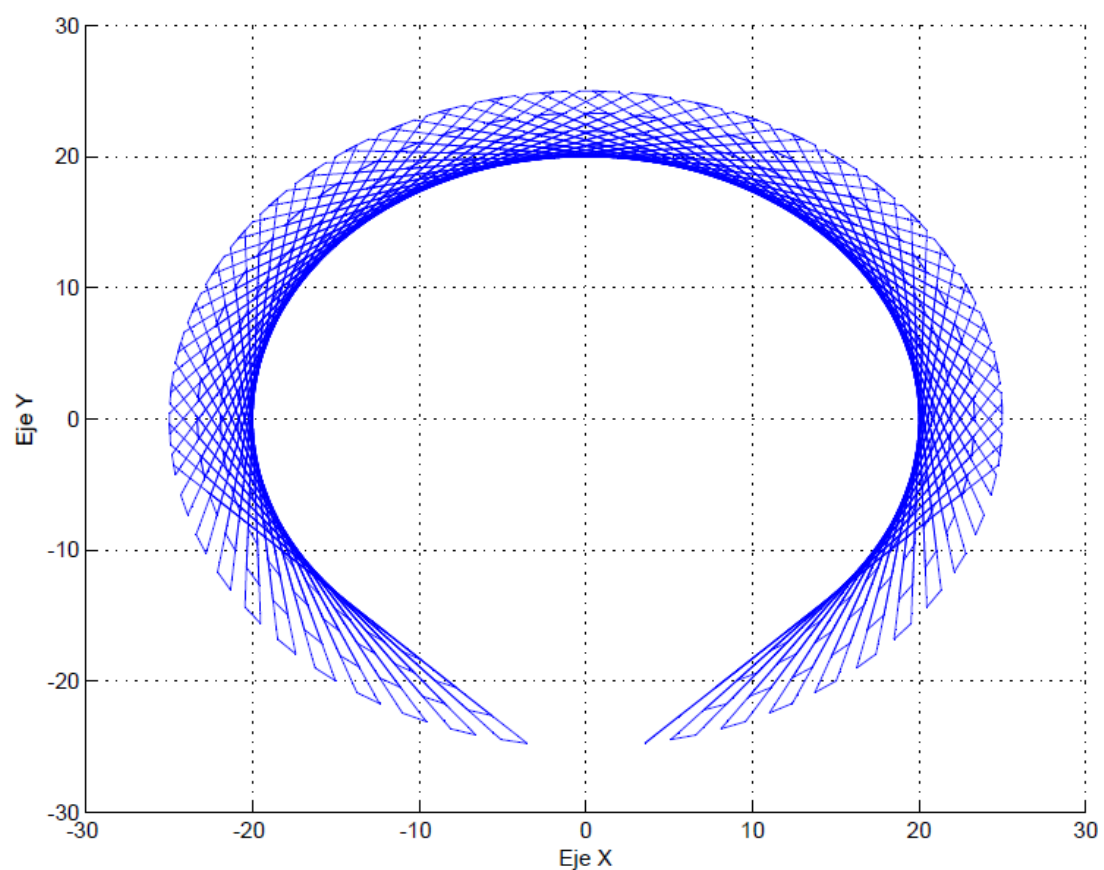
        for q2 = pi*1/2:-pi/10:-pi*1/2
            A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
            A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
            A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
            T = A1 * A2 * A3 ;
            XYZ(:,indx) = T * UVW;
            indx = indx + 1;
        end
    end;
end
else
    for q1 = pi*3/4:-pi/50:-pi*3/4 %thetal varío de (3/4)pi a (-
3/4)pi

        if b == 0, b = 1;
        else b = 0;
        end;
        if b == 0,
            for q2 = -pi*1/2:pi/10:pi*1/2
                A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                T = A1 * A2 * A3 ;
                XYZ(:,indx) = T * UVW;
                indx = indx + 1;
            end
        else
            for q2 = pi*1/2:-pi/10:-pi*1/2
                A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                T = A1 * A2 * A3 ;
                XYZ(:,indx) = T * UVW;
                indx = indx + 1;
            end
        end
    end;
end;
end;
end
X = XYZ(1,:);
Y = XYZ(2,:);
Z = XYZ(3,:);
plot3(X,Y,Z);
colormap;
grid on;
xlabel('Eje X');
ylabel('Eje Y');
zlabel('Eje Z');
axis ([-30 30 -30 30 50 0]);
%*****
% Fin del Script
%*****

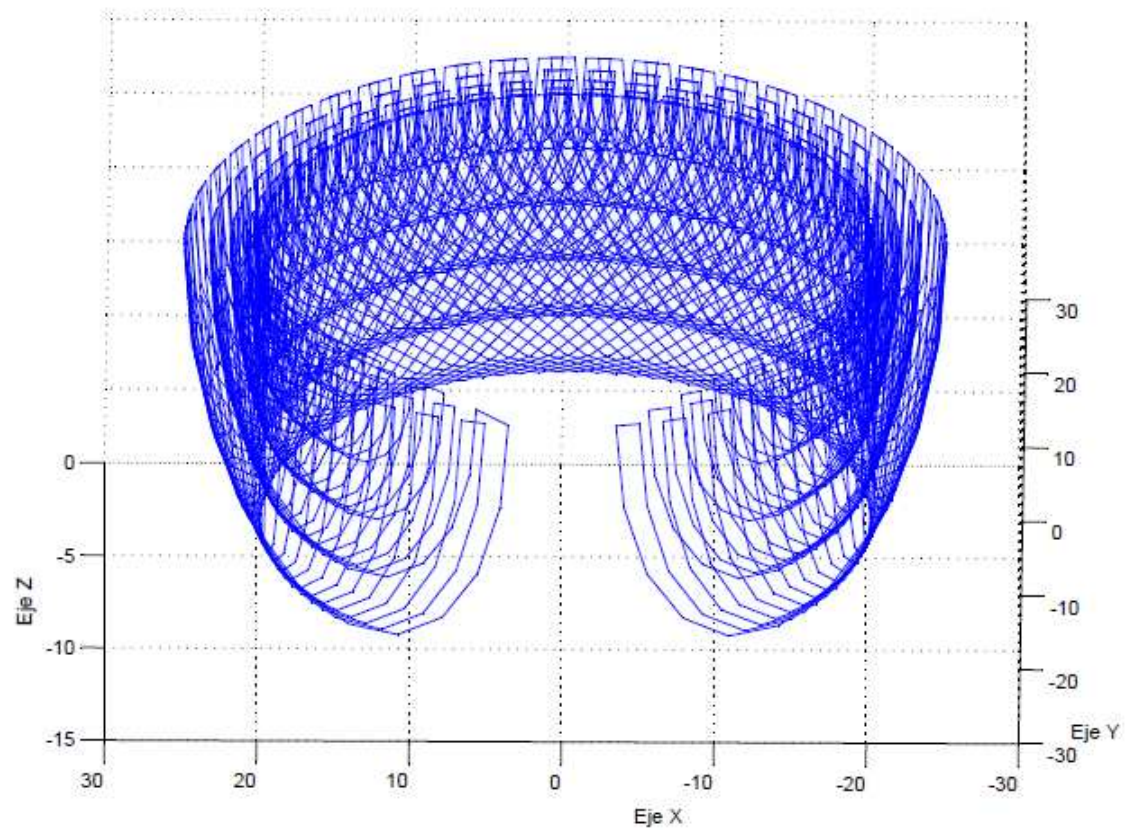
```



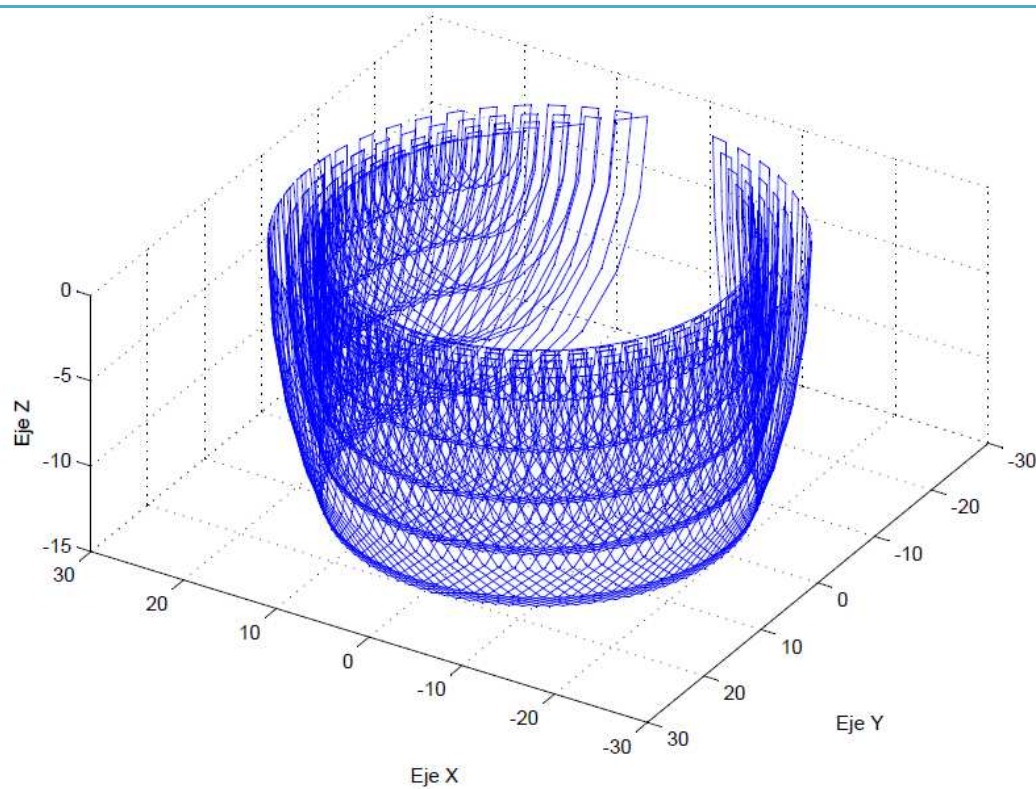
Vista en perspectiva del área de trabajo (Ejemplo1).



Vista superior (plano X - Y) del área de trabajo (Ejemplo1).



Vista en perspectiva del área de trabajo (Ejemplo1).



Vista en perspectiva del área de trabajo tomada desde el extremo opuesto (Ejemplo1).

Ejemplo 2:

$L1 = 50;$
 $L2 = 20;$
 $0 \leq L3 \leq 25;$

$-\pi/4 \leq \theta_1 \leq \pi/4;$
 $-\pi/2 \leq \theta_2 \leq \pi/2;$
 $\theta_3 = 0;$

```
%*****
%UTN - FACULTAD REGIONAL BUENOS AIRES
%MATERIA: ROBOTICA.
%TESIS FINAL: ROBOT TIPO STANFORD DE 3 GDL.
%PROFESOR: ING. HERNAN GIANNETTA.
%JTP: ING. DAMIAN GRANZELLA.
%ALUMNOS: PERELLO JUAN PABLO , PIMENTEL NICOLAS.
%*****

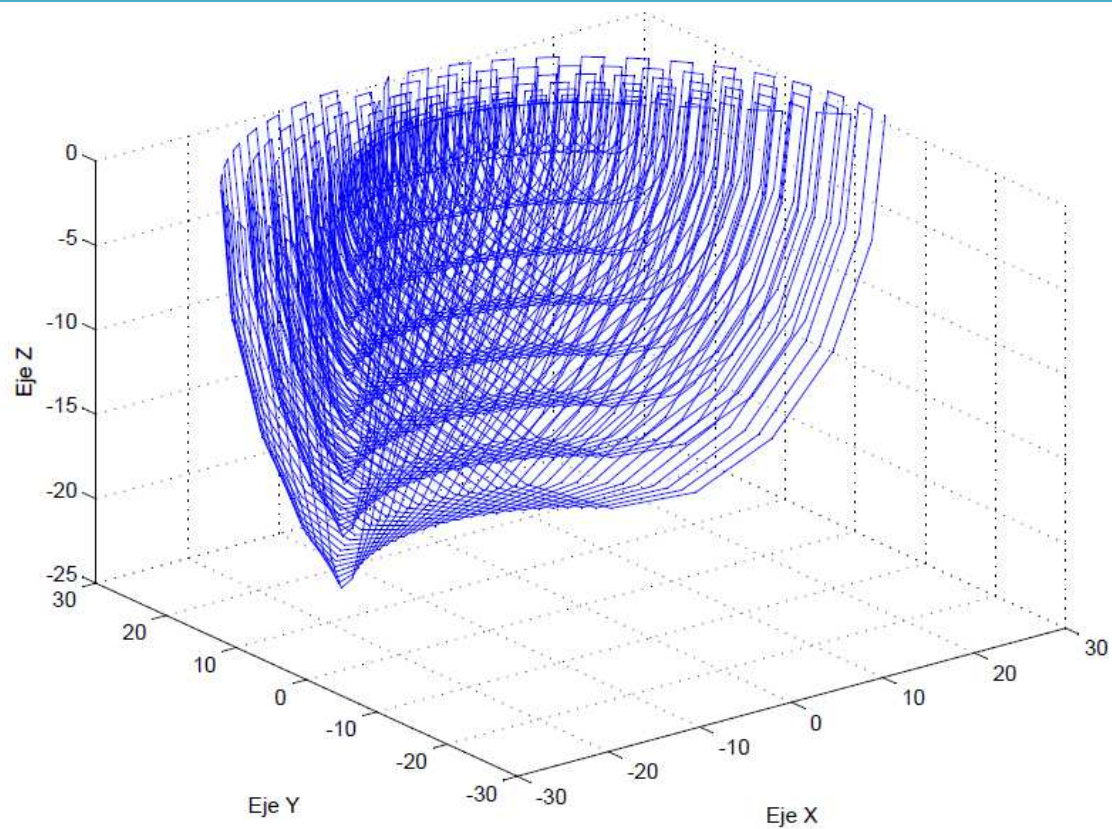
%*****
%Inicio del Script para generar volumen de trabajo del Robot tipo
Stanford
%*****
L1 = 50;
L2 = 20;
XYZ = [];
UVW = [0;0;0;1];
% a y b los utilizo como auxiliar para barrer de izq a der y de der a
izq
% alternadamente.
a = 0;
b = 0;
indx = 1;
for L3 = 0:3:25 %L3 varío de 0 a 15 en pasos de a 3.
    if a == 0, a = 1;
    else a = 0;
    end;
    if a == 0,
        for q1 = -pi*1/4:pi/50:pi*1/4 %thetal varío de (-3/4)pi a
(3/4)pi
            if b == 0, b = 1;
            else b = 0;
            end;
            if b == 0,
                for q2 = -pi*1/2:pi/10:pi*1/2
                    A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                    A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                    A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                    T = A1 * A2 * A3 ;
                    XYZ(:,indx) = T * UVW;
                    indx = indx + 1;
                end
            else

```

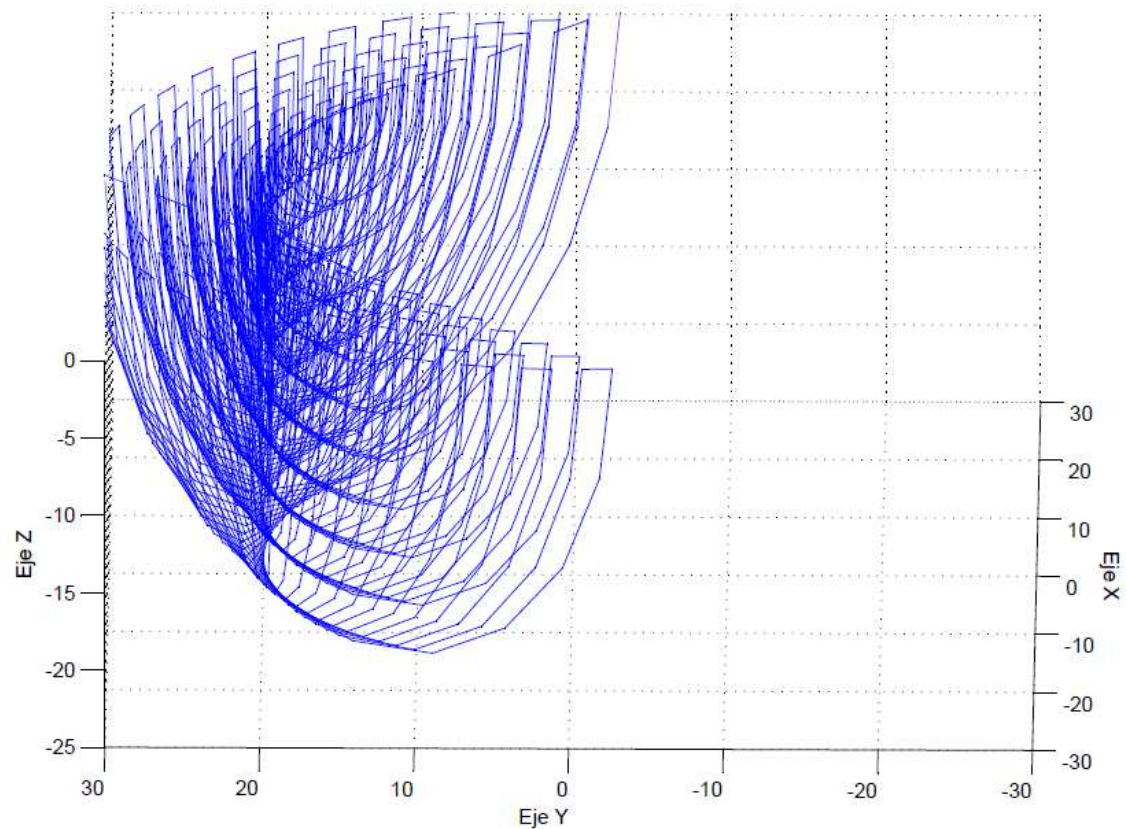
```

        for q2 = pi*1/2:-pi/10:-pi*1/2
            A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
            A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
            A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
            T = A1 * A2 * A3 ;
            XYZ(:,indx) = T * UVW;
            indx = indx + 1;
        end
    end;
end
else
    for q1 = pi*1/4:-pi/50:-pi*1/4 %thetal varío de (3/4)pi a (-
3/4)pi
        if b == 0, b = 1;
        else b = 0;
        end;
        if b == 0,
            for q2 = -pi*1/2:pi/10:pi*1/2
                A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                T = A1 * A2 * A3 ;
                XYZ(:,indx) = T * UVW;
                indx = indx + 1;
            end
        else
            for q2 = pi*1/2:-pi/10:-pi*1/2
                A1 = [cos(q1) -sin(q1) 0 0 ; sin(q1) cos(q1) 0 0 ; 0
0 1 0 ; 0 0 0 1];
                A2 = [cos(q2) -sin(q2) 0 0 ; 0 0 1 L2 ; -sin(q2)
cos(q2) 0 0 ; 0 0 0 1];
                A3 = [1 1 0 0 ; 0 0 -1 -L3 ; 0 1 0 0 ; 0 0 0 1];
                T = A1 * A2 * A3 ;
                XYZ(:,indx) = T * UVW;
                indx = indx + 1;
            end
        end
    end;
end;
end;
X = XYZ(1,:);
Y = XYZ(2,:);
Z = XYZ(3,:);
plot3(X,Y,Z);
colormap;
grid on;
xlabel('Eje X');
ylabel('Eje Y');
zlabel('Eje Z');
axis ([-30 30 -30 30 50 0]);
%*****
% Fin del Script
%*****

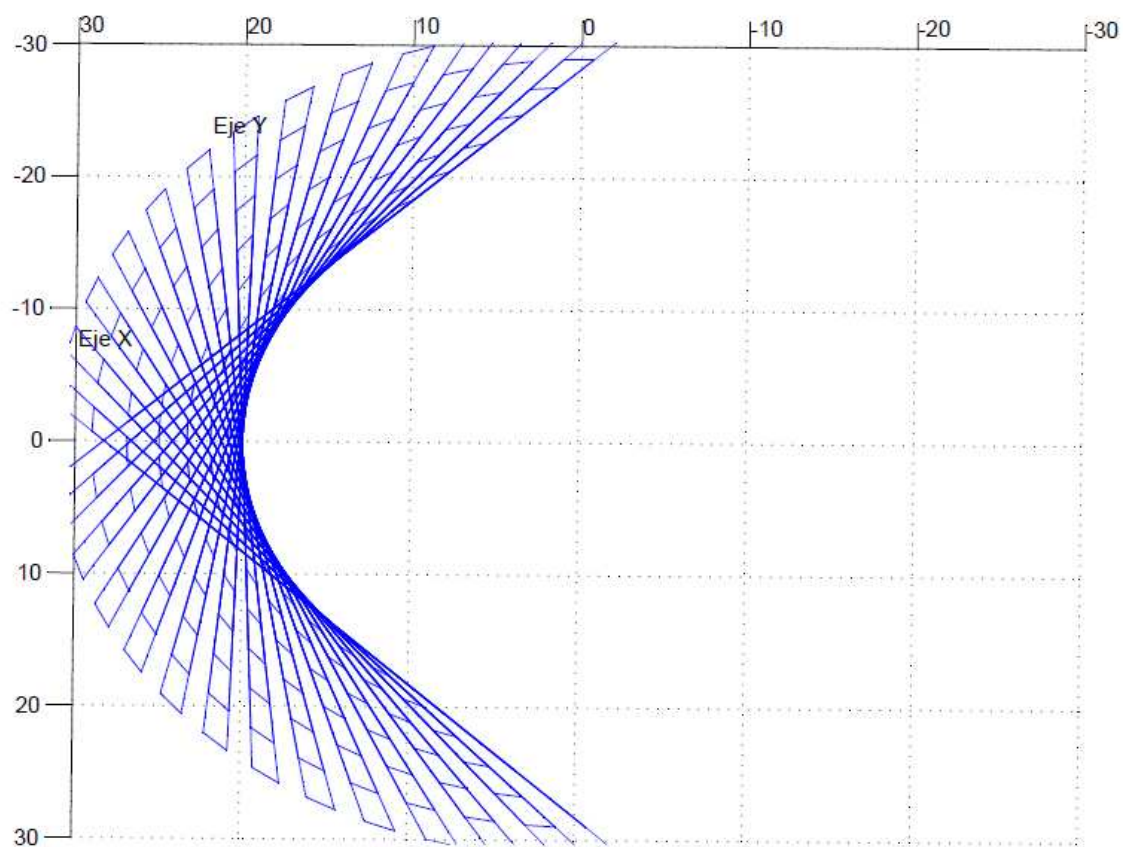
```

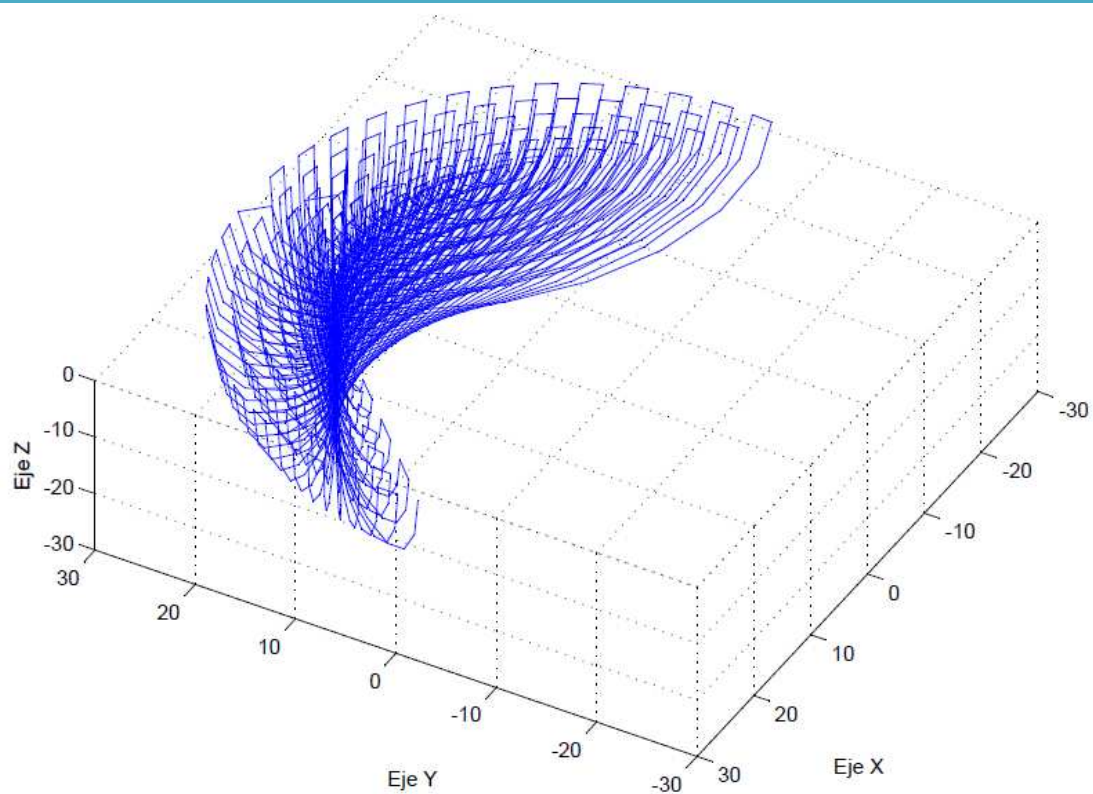
Vista en perspectiva del área de trabajo (Ejemplo2).



Vista en perspectiva del área de trabajo (Ejemplo2).



Vista superior (plano X - Y) del área de trabajo (Ejemplo2).



Vista en perspectiva del área de trabajo (Ejemplo2).

Dinámica del robot

Introducción Teórica:

La dinámica se ocupa de la relación entre las fuerzas que actúan sobre un cuerpo y el movimiento que él origina. 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:

- 1) La localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración.
- 2) Las fuerzas y pares aplicados en las articulaciones (o en el extremo del robot).
- 3) 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 planeamiento 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 2^{do} orden, cuya integración permite 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.

El problema de la obtención del modelo de un robot es, por lo tanto, uno de los aspectos más complejos de la robótica, lo que ha llevado a ser obviado en numerosas ocasiones. Sin embargo, el modelo dinámico es imprescindible para conseguir los siguientes fines:

- 1) Simulación del movimiento del robot.
- 2) Diseño y evaluación de la estructura mecánica del robot.
- 3) Dimensionamiento de los actuadores.
- 4) Diseño y evaluación del control dinámico del robot.

Este último fin es evidentemente de gran importancia, pues de la calidad del control dinámico del robot depende la precisión y velocidad de sus movimientos. La gran complejidad ya comentada existente de la obtención del modelo dinámico del robot, ha motivado que se realicen ciertas simplificaciones, de manera que así pueda ser utilizado en el diseño del controlador.

Es importante hacer notar que el modelo dinámico completo de un robot debe incluir no sólo la dinámica de sus elementos (barras o eslabones) sino también la propia de sus sistemas de transmisión, de los actuadores y sus equipos electrónicos de mando. Estos elementos incorporan al modelo dinámico nuevas inercias, rozamientos, saturaciones de los circuitos electrónicos, etc. aumentando aún más su complejidad.

Por último, es preciso señalar que si bien en la mayor parte de las aplicaciones reales de la robótica, las cargas e inercias manejadas no son suficientes como para originar deformaciones en los eslabones del robot, en determinadas ocasiones no ocurre así, siendo preciso considerar al robot como un conjunto de eslabones no rígidos. Aplicaciones de este tipo pueden encontrarse en la robótica espacial o en robots de grandes dimensiones, entre otras.

En nuestro trabajo caso vamos a realizar el análisis dinámico de la estructura mecánica del **“Robot tipo Stanford”**. Vamos a llevar a cabo el análisis dinámico del robot, complementando el análisis cinemático.

En el presente trabajo se hará el análisis utilizando las formulaciones lagrangianas por ser una herramienta muy eficaz a medida que aumentan los grados de libertad. Es importante señalar que existen otras formulaciones también validas como las newtonianas y variantes entre estas dos que se han ido adaptando para obtener una mejor implementación computacional.

Plantear este análisis complementa el estudio del comportamiento del robot para diseñar las etapas de control del mismo. Para la generación de las trayectorias se implementara un control mediante modulación de ancho de pulso (PWM), para el cual se utilizara un lenguaje de descripción de hardware (VHDL) y será implementado sobre un FPGA.

Para terminar se presentarán los resultados de la simulación, los cuales permitirán analizar el comportamiento del robot y los sistemas mecánicos ante las trayectorias propuestas.

A continuación vamos a desarrollar los métodos más utilizados considerando a los robots como rígidos.

Modelo dinámico de la estructura mecánica de un robot rígido

Este modelo se basa fundamentalmente en el planteamiento del equilibrio de fuerzas establecido en la segunda ley de Newton, o en su equivalente de rotación, la ley de Euler:

$$\sum F = m \cdot \dot{v}$$
$$\sum T = I \cdot \dot{\omega} + \omega \times (I \omega)$$

De este planteamiento del equilibrio de fuerzas y pares que intervienen sobre el robot se obtienen los denominados modelos dinámicos directo e inverso:

- **Modelo dinámico directo:** expresa la evolución temporal de las coordenadas articulares del robot en función de las fuerzas y pares que intervienen.
- **Modelo dinámico inverso:** expresa las fuerzas y pares que intervienen en función de la evolución de las coordenadas articulares y sus derivadas.

En el planteamiento del equilibrio de fuerzas en un robot real de 5 ó 6 grados de libertad, debe tenerse en cuenta que junto con las fuerzas de inercia y gravedad, aparecen fuerzas de Coriolis debidas al movimiento relativo existente entre los diversos elementos, así como de fuerzas centrípetas que dependen de la configuración instantánea del manipulador.

Un planteamiento alternativo para la obtención del modelo puede ser usar la formulación Lagrangiana, basada en consideraciones energéticas. Este método es más sistemático y facilita enormemente la formulación de un modelo.

Lagrange establece la ecuación:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau$$
$$L = k - \mu$$

Con:

q_i : coordenadas generalizadas

τ : vector de fuerzas y pares aplicados a las q_i

L: función lagrangiana

k: energía cinética

μ : energía potencial

Obtención del modelo dinámico de un robot mediante la formulación de Lagrange – Euler

Este planteamiento utiliza, por tanto, las matrices ${}^{i-1}A_i$ que relacionan el sistema de coordenadas de referencia del elemento i con el del elemento i-1.

Se realizan en este caso operaciones de producto y suma innecesarias. Se trata de un procedimiento ineficiente desde el punto de vista computacional, donde tiene una complejidad de $O(n^4)$. Sin embargo, conduce a unas ecuaciones finales bien estructuradas donde aparecen de manera clara los diversos pares y fuerzas que intervienen en el movimiento.

Algoritmo computacional para el modelo dinámico por Lagrange – Euler

1. Asignar a cada eslabón un sistema de referencia de acuerdo a las normas D-H.
2. Obtener las matrices de transformación 0A_i para cada elemento i.
3. Obtener las matrices U_{ij} definidas por:

$$U_{ij} = \frac{\partial}{\partial q_j} \cdot {}^0A_i$$

4. Obtener las matrices U_{ijk} definidas por:

$$U_{ijk} = \frac{\partial U_{ij}}{\partial q_k}$$

5. Obtener las matrices de pseudoinercias J_i para cada elemento, que vienen definidas por:

$$U_{ijk} = \begin{bmatrix} \int x_i^2 dm & \int x_i y_i dm & \int x_i z_i dm & \int x_i dm \\ \int y_i x_i dm & \int y_i^2 dm & \int y_i z_i dm & \int y_i dm \\ \int z_i x_i dm & \int z_i y_i dm & \int z_i^2 dm & \int z_i dm \\ \int x_i dm & \int y_i dm & \int z_i dm & \int dm \end{bmatrix}$$

6. Obtener la matriz de inercias $D=[d_{ij}]$ cuyos elementos vienen definidos por:

$$d_{ij} = \sum_{k=(\max i, j)}^n \text{Traza}(U_{kj} J_k T_{ki}^T)$$

Con $i, j = 1, 2, \dots, n$

n : número de grados de libertad

7. Obtener los términos h_{ikm} por:

$$h_{ikm} = \sum_{j=(\max i, k, m)}^n \text{Traza}(U_{jkm} J_j T_{ji}^T)$$

Con $i, k, m = 1, 2, \dots, n$

8. Obtener la matriz columna de fuerzas de Coriolis y centrípeta $H=[h_i]^T$ cuyos elementos viene definidos por:

$$h_i = \sum_{k=1}^n \sum_{m=1}^n h_{ikm} \dot{q}_k \dot{q}_m$$

9. Obtener la matriz columna de fuerzas de gravedad $C=[c_i]^T$ cuyos elementos viene definidos por:

$$c_i = \sum_{j=1}^n (-m_j g U_{ji}^j r_j)$$

con $i = 1, 2, \dots, n$

g : es el vector de gravedad expresado en el sistema de la base $\{S_0\}$ y viene expresado en $(g_{x0}, g_{y0}, g_{z0}, 0)$

${}^i r_j$: es el vector de coordenadas homogéneas del centro de masa del elemento j expresado en el sistema de referencia del elemento i .

10. La ecuación dinámica del sistema será:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau$$

Donde:

τ : Vector de fuerzas y pares motores efectivos aplicados sobre cada coordenada q_i .

$M(q)$: Matriz de Inercias.

$C(q, \dot{q})$: Matriz columna de fuerzas de Coriolis y Centrípeta.

G : Matriz columna de fuerzas de gravedad.

Modelo dinámico usando Newton-Euler

El modelo dinámico obtenido por el método de Lagrange nos lleva a un algoritmo sumamente complejo a nivel computacional ($O(n^4)$ siendo n el número de grados de libertad). Lo antes mencionado se puede comprender como que a mayor número de grados de libertad el algoritmo se vuelve inutilizable para aplicaciones en tiempo real.

La ecuación de Newton-Euler parte del equilibrio de fuerzas y pares como la segunda ley de Newton:

$$\sum F = m \frac{\partial v}{\partial t} \quad \sum T = I \frac{\partial \omega}{\partial t} + \omega \cdot (I \omega)$$

Siendo:

F : La fuerza sobre la barra.

T : Los pares ejercidos sobre la barra.

m : Masa de la barra

I : Tensor de inercia de la barra entorno a su centro de masa expresado en el sistema de referencia.

v : Las velocidades lineales de las articulaciones.

ω : Las velocidades angulares de las articulaciones.

La fórmula anterior se plantea para cada articulación en particular. La mejor forma de implementar este método es mediante una formulación recursiva, lo cual posibilita la obtención de la posición, velocidad y aceleración del eslabón i con respecto a la base a partir de los correspondientes del eslabón $i-1$ y del movimiento relativo de la articulación. Realizando lo antes dicho empezando del eslabón 1 se llega rápidamente al n . Con los datos hallados anteriormente se obtienen las fuerzas y pares que aparecen sobre el eslabón i referidos nuevamente a la base del robot a partir de los correspondientes al eslabón $i+1$, recorriendo de esta forma todos los eslabones desde el n al primero, es decir de manera inversa.

El método utiliza operaciones vectoriales, las cuales son más eficientes, a nivel computacional, que las operaciones matriciales utilizadas por la formulación de Lagrange. Se puede observar que el método reduce mucho la complejidad desde $O(n^4)$ a $O(n)$ (siendo n el número de grados de libertad).

Los pasos para realizar un análisis mediante el método de Newton-Euler son los siguientes:

1. Asignar a cada eslabón un sistema de referencia teniendo en cuenta las normas D-H.
2. Obtener las matrices de rotación:

$${}^{i-1}R = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i \\ S\theta_i & C\alpha_i S\theta_i & -S\alpha_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i \end{bmatrix}$$

3. Establecer las condiciones iniciales (tomando como sistema base el $\{S_0\}$):

$${}^0\omega : \text{velocidad angular} = [0 \quad 0 \quad 0]^T$$

$${}^0 \frac{\partial \omega}{\partial t} : \text{aceleración angular} = [0 \quad 0 \quad 0]^T$$

$${}^0v : \text{velocidad lineal} = [0 \quad 0 \quad 0]^T$$

$${}^0 \frac{\partial v}{\partial t} : \text{velocidad angular} = [g_x \quad g_y \quad g_z]^T$$

$${}^i z_0 = [0 \quad 0 \quad 1]^T$$

$${}^i p_i : \text{coordenadas del origen del sistema } s\{i\} \text{ respecto a } s\{i-1\} = [a_i \quad d_i S_i \quad d_i C_i]$$

$${}^i S_i : \text{coordenadas del centro de masa del eslabón } i \text{ respecto del sistema } \{S_i\}.$$

$${}^i I_i : \text{Matriz de inercia del eslabón } i \text{ respecto de su centro de masa expresado en } \{S_i\}$$

Para $i = 1$ hasta n hay que realizar los pasos 4 al 7.

4. Obtener la velocidad angular del sistema:

Si el eslabón es de rotación:

$${}^i \omega_i = {}^i R_{i-1} ({}^{i-1} \omega_{i-1} + z_0 \frac{\partial q_i}{\partial t})$$

Si el eslabón es de traslación:

$${}^i R_{i-1} {}^{i-1} \omega_{i-1}$$

5. Obtener la aceleración angular del sistema:

Si el eslabón es de rotación:

$$\frac{\partial {}^i \omega_i}{\partial t} = {}^i R_{i-1} \left(\frac{\partial {}^{i-1} \omega_{i-1}}{\partial t} + z_0 \frac{\partial^2 q_i}{\partial t^2} \right) + {}^{i-1} \omega_{i-1} x z_0 \frac{\partial q_i}{\partial t}$$

Si el eslabón es de traslación:

$${}^i R_{i-1} \frac{\partial {}^{i-1} \omega_{i-1}}{\partial t}$$

6. Obtener la aceleración lineal del sistema:

Si el eslabón es de rotación:

$$\frac{\partial {}^i v_i}{\partial t} = {}^i R_{i-1} \frac{\partial {}^{i-1} v_{i-1}}{\partial t} + \left(\frac{\partial {}^i \omega_i}{\partial t} x^i p_i \right) + {}^i \omega_i x ({}^i \omega_i x^i p_i)$$

Si el eslabón es de traslación:

$${}^i R_{i-1} \left(z_0 \frac{\partial^2 q_i}{\partial t^2} + \frac{\partial {}^{i-1} v_{i-1}}{\partial t} \right) + \left(\frac{\partial {}^i \omega_i}{\partial t} x^i p_i \right) + 2 {}^i \omega_i x {}^i R_{i-1} z_0 \frac{\partial q_i}{\partial t} + {}^i \omega_i x ({}^i \omega_i x^i p_i)$$

7. Obtener la aceleración lineal del centro de gravedad del eslabón i:

$${}^i a_i = \frac{\partial {}^i \omega_i}{\partial t} x^i S_i + {}^i \omega_i x ({}^i \omega_i x^i s_i) + \frac{\partial {}^i v_i}{\partial t}$$

Para i = n hasta 1 hay que realizar los pasos 8 al 10.

8. Obtener la fuerza ejercida sobre el eslabón i:

$${}^i f_i = {}^i R_{i+1} {}^{i+1} f_{i+1} + {}^i a_i m_i$$

9. Obtener el par ejercida sobre el eslabón i:

$${}^i n_i = {}^i R_{i+1} \left[{}^{i+1} n_{i+1} + ({}^i R_{i+1} {}^i p_i) x^{i+1} f_{i+1} \right] + ({}^i p_i {}^i s_i) x^i a_i m_i + {}^i I_i \frac{d {}^i \omega_i}{dx} + {}^i \omega_i x ({}^i I_i {}^i \omega_i)$$

10. Obtener la fuerza o par aplicado a la articulación i:

$$\tau = {}^i n_i^T {}^i R_{i-1} z_0$$

$${}^i f_i^T {}^i R_{i-1} z_0$$

Donde τ es el par o fuerza efectivo (par motor menos pares de rozamiento o perturbación).

Modelo dinámico usando variables de estado

La siguiente ecuación define el modelo dinámico inverso de un robot, indicando los pares y fuerzas que deben realizar los actuadores para que las variables articulares realizan una trayectoria en particular $q(t)$:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau$$

Se debe considerar el vector de pares generalizados τ :

$$\tau = \tau_{motor} - \tau_{perturbador} - \tau_{rozamiento\ viscoso} - \tau_{rozamiento\ seco}$$

La expresión anterior es por lo tanto no lineal. Debido a lo antes mencionado se puede obtener el modelo dinámico que proporciona la trayectoria realizada como consecuencia de la aplicación de unos pares determinados.

Para obtener el modelo de estados del sistema, la mejor elección de variables de estados es usar la posición y la velocidad de las articulaciones. Realizando cálculos matriciales se obtiene la matriz final del modelo.

$$\frac{d}{dt} \begin{bmatrix} q \\ \frac{dq}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \frac{dq}{dt} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$u = D^{-1}(\tau - N)$$

A continuación vamos a definir los parámetros del robot, necesarios para seleccionar los motores que le vamos a utilizar:

Masas

- $m_1=5\text{Kg}$
- $m_2=3\text{Kg}$
- $m_3=2\text{Kg}$

Longitudes

- $L_1=0,20\text{m}$
- $L_2=0,25\text{m}$
- $L_3=0,30\text{m}$

Centros de masas

Para simplificar el cálculo del modelo, vamos a suponer que las masas de las articulaciones se encuentran todas concentradas en el centro de masa, y ubicados éstos en los extremos de dichas articulaciones. Por lo antes dicho vamos a tener los sig. valores:

$$lcm_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad lcm_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad lcm_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Debido a que estamos trabajando con un robot de 3 grados de libertad, la cantidad de operaciones de multiplicación y sumas a realizar se torna un número considerable (obviamente menor que utilizando el método de Lagrange-Euler), por lo que vamos a utilizar el ToolBox de Matlab "Hemero", desarrollado por el señor Aníbal Ollero Baturone, quien explica su funcionamiento en el libro Robótica: "Manipuladores y robots móviles".

Para resolver las ecuaciones se utiliza una matriz llamada dyn. Dicha matriz posee $n \times 20$ elementos, en donde n es la cantidad de articulaciones del robot a estudiar.

Matriz *dyn*

Las columnas de dicha matriz se conforman con los siguientes datos:

1. $\alpha(i-1)$: Parámetros de Denavit-Hartenberg
2. $a(i-1)$: Parámetros de Denavit-Hartenberg
3. $\theta(i)$: Parámetros de Denavit-Hartenberg
4. $d(i)$: Parámetros de Denavit-Hartenberg
5. $\sigma(i)$: Tipo de articulación; 0 si es de rotación y 1 si es prismática
6. masa: Masa del enlace i
7. r_x : Centro de masas del enlace respecto al cuadro de referencia de dicho enlace
8. r_y : Centro de masas del enlace respecto al cuadro de referencia de dicho enlace
9. r_z : Centro de masas del enlace respecto al cuadro de referencia de dicho enlace
10. I_{xx} : Elementos del tensor de inercia referido al centro de masas del enlace
11. I_{yy} : Elementos del tensor de inercia referido al centro de masas del enlace
12. I_{zz} : Elementos del tensor de inercia referido al centro de masas del enlace
13. I_{xy} : Elementos del tensor de inercia referido al centro de masas del enlace
14. I_{yz} : Elementos del tensor de inercia referido al centro de masas del enlace
15. I_{xz} : Elementos del tensor de inercia referido al centro de masas del enlace
16. J_m : Inercia de la armadura
17. G : Velocidad de la articulación / velocidad del enlace
18. B : Fricción viscosa, referida al motor
19. T_{c+} : Fricción de Coulomb (rotación positiva), referida al motor
20. T_{c-} : Fricción de Coulomb (rotación negativa), referida al motor

En nuestro caso, la matriz *dyn* correspondiente sería la siguiente:

0	0	t1	0	0	m1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
-pi/2	0	t2	L2	0	m2	0	0	0	0	0	0	0	0	0	0	1	0	0	0
pi/2	0	0	L3	0	m3	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Luego, para calcular los τ correspondientes a cada articulación, se ejecutan en Matlab el siguiente script:

```
%*****
%UTN - FACULTAD REGIONAL BUENOS AIRES
%MATERIA: ROBOTICA.
%TESIS FINAL: ROBOT TIPO STANFORD DE 3 GDL.
%PROFESOR: ING. HERNAN GIANNETTA.
%JTP: ING. DAMIAN GRANZELLA.
%ALUMNOS: PERELLO JUAN PABLO , PIMENTEL NICOLAS.
%*****

%*****
%Inicio del Script para obtener los torques de los motores
%*****

m1=5;% masa expresada en Kg
m2=3;% masa expresada en Kg
m3=2;% masa expresada en Kg
L1=0.20;%Longitud expresada en metros
L2=0.25;%Longitud expresada en metros
L3=0.30;%Longitud expresada en metros
syms g real%Aceleracion de la gravedad
syms t1 t2 t3 real; %Variables simbólicas de posición de cada
articulación
syms td1 td2 td3 real; %Variables simbólicas de velocidad de cada
articulación
syms tdd1 tdd2 tdd3 real; %Variables simbólicas de aceleración de cada
articulación
syms g real; %Aceleración de la gravedad
dyn=[0 0 t1 0 0 m1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ;
      -pi/2 0 t2 L2 0 m2 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ;
      pi/2 0 0 L3 0 m3 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
q=[t1 t2 t3]; %vector de posiciones
qd=[td1 td2 td3]; %vector de velocidades
qdd=[tdd1 tdd2 tdd3]; %vector de posiciones
grav=[0 -9.8 0];% vector de aceleración de la gravedad
tau=rne(dyn,q,qd,qdd,grav)

%*****
% Fin del Script
%*****
```

Referencias

t1	θ_1
t2	θ_2
t3	θ_3
td1	$\dot{\theta}_1$
td2	$\dot{\theta}_2$
td3	$\dot{\theta}_3$
tdd1	$\ddot{\theta}_1$
tdd2	$\ddot{\theta}_2$
tdd3	$\ddot{\theta}_3$

Con esta última instrucción obtenemos un vector de 1 fila con 3 columnas, en donde están los pares motores de cada articulación.

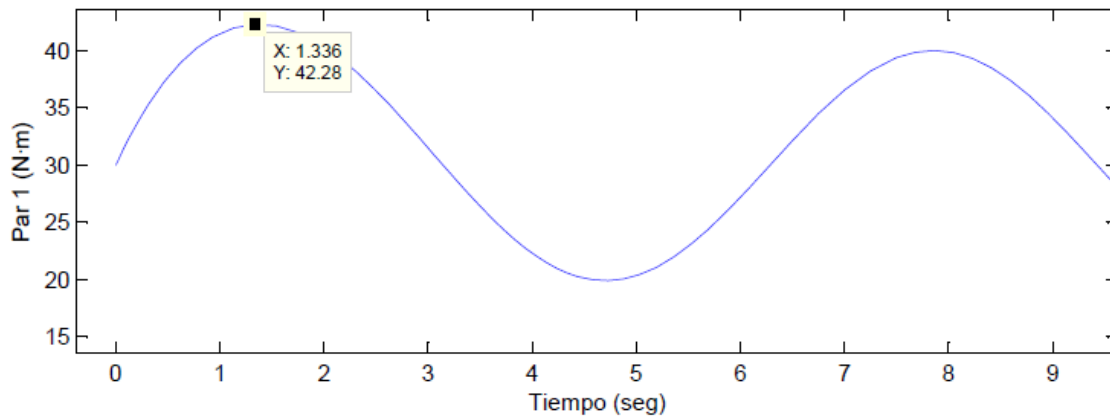
Al ejecutar estas instrucciones en Matlab, obtuvimos los siguientes pares motores, en forma simbólica. O sea, que de acuerdo a los valores de posición, velocidad y aceleración que necesitemos obtener del robot, podemos calcular los pares motores que se deberán aplicar a las articulaciones del mismo.

$$\begin{aligned}\tau_1 = & -\sin(t_2)*(-3/10*\sin(t_3)*(2*\cos(t_3)*(3/10*tdd2- \\ & 3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\sin(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))-3/10*\cos(t_3)*(-2*\sin(t_3)*(3/10*tdd2- \\ & 3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\cos(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))-1/4*\cos(t_2)*(cos(t_3)*(2*\cos(t_3)*(3/10*tdd2- \\ & 3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\sin(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))-\sin(t_3)*(-2*\sin(t_3)*(3/10*tdd2- \\ & 3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\cos(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))+3*\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+1/4*\sin(t_2)*(3/5*td2^2+3/5*\sin(t_2)^2*td1^2-5*\sin(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))\end{aligned}$$

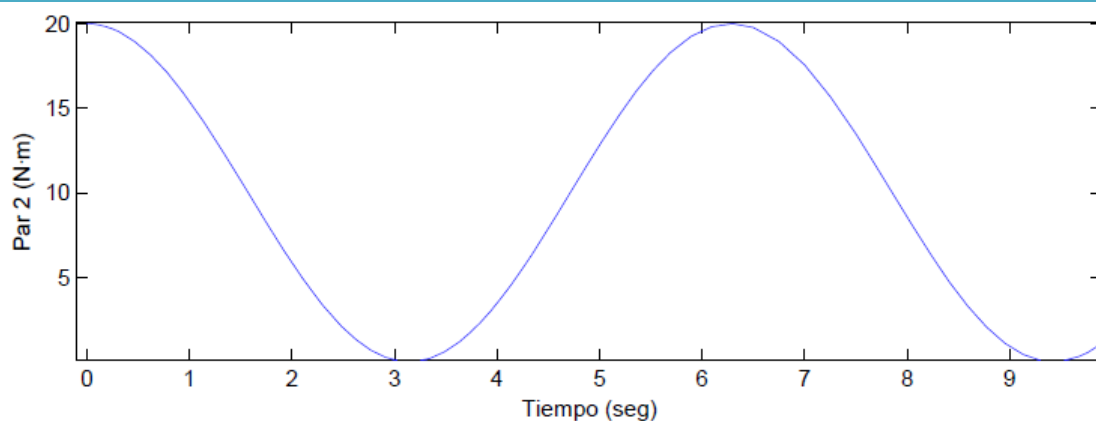
$$\begin{aligned}\tau_2 = & 3/10*\cos(t_3)*(2*\cos(t_3)*(3/10*tdd2-3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\sin(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))-3/10*\sin(t_3)*(-2*\sin(t_3)*(3/10*tdd2- \\ & 3/10*\cos(t_2)*td1^2*\sin(t_2)+\cos(t_2)*(- \\ & 1/4*tdd1+49/5*\sin(t_1)))+2*\cos(t_3)*(3/10*\sin(t_2)*tdd1+3/5*\cos(t_2)*td1*td2- \\ & 1/4*td1^2+49/5*\cos(t_1)))\end{aligned}$$

$$\tau_3 = 0$$

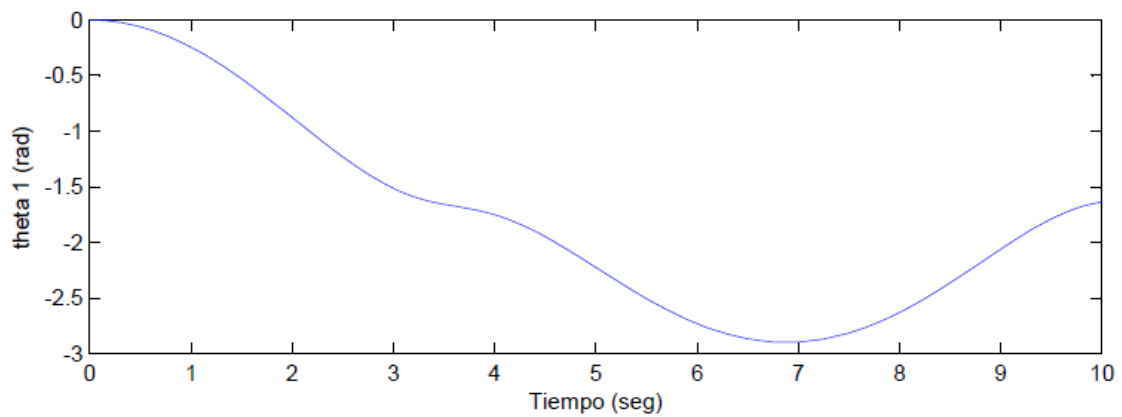
Evaluamos los torques antes descriptos utilizando el toolbox de HEMERO y determinaremos los motores a utilizar en base a los parámetros obtenidos de los distintos gráficos.



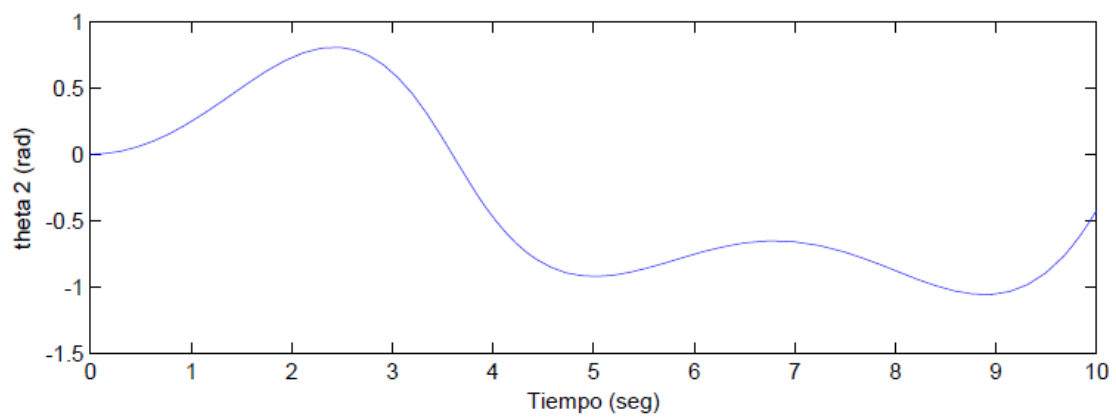
En base a la gráfica obtenida podemos utilizar el motor modelo SYNCHRONOUS MOTOR 1FT7 50 NM (100K), 3000 RPM WATER-COOLED de Siemens que puede proveer hasta 50Nw.m y nosotros tenemos como máximo 42,28 Nw.m (<http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objID=10803978&subtype=133200>).



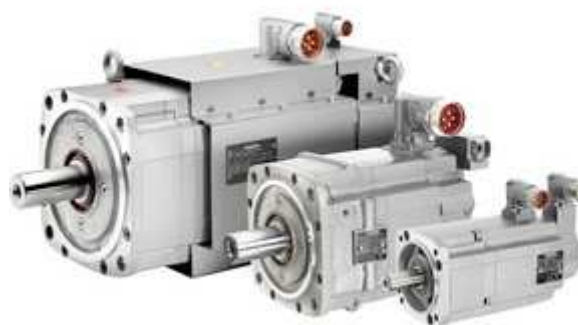
En base a la gráfica obtenida podemos utilizar el motor modelo SYNCHRONOUS MOTOR 1FT7 27 NM (100K), 3000 RPM SEPARATE VENTILATION de Siemens que puede proveer hasta 27 Nw.m y nosotros tenemos como máximo 20 Nw.m (<http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objID=10803978&subtype=133200>).



En base a este gráfico podemos evaluar la velocidad que tendrá θ_1



En base a este gráfico podemos evaluar la velocidad que tendrá θ_2



Fotografía de la familia de motores utilizados para el proyecto
(<http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objID=10803978&subtype=133200>)

Codigo VHDL PWM:

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE user_pkg IS

    function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;

    function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;

END user_pkg ;

PACKAGE BODY user_pkg IS

function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is

    variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);

    begin

XV := X;

for I in 0 to XV'HIGH LOOP

    if XV(I) = '0' then

        XV(I) := '1';

        exit;

    else XV(I) := '0';

    end if;

end loop;

return XV;

end INC;

function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is

    variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);

    begin

XV := X;
```



```
for I in 0 to XV'HIGH LOOP

    if XV(I) = '1' then

        XV(I) := '0';

        exit;

    else XV(I) := '1';

    end if;

end loop;

return XV;

end DEC;

END user_pkg;
```

```
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;
```

Codigo VHDL Control de Motor

```
library IEEE;
```

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

```
USE ieee.std_logic_arith.all;
```

```
ENTITY MotorControl IS
```

```
PORT ( clockSys,resetSys,sentido :in STD_LOGIC;
```

```
      DataSys :in std_logic_vector(7 downto 0);
```

```
      HALLs :in std_logic_vector(2 downto 0);
```

```
      pwm1,pwm2,pwm3 :out STD_LOGIC;
```

```
      salidasQ :out std_logic_vector(2 downto 0)
```

```
);
```

```
END MotorControl;
```

```
ARCHITECTURE estrucMotorContol OF MotorControl IS
```

```
COMPONENT pwm_fpga
```

```
PORT (clock: IN STD_LOGIC;
```

```
      reset: IN STD_LOGIC;
```

```
      data_value: IN std_logic_vector(7 downto 0);
```

```
      pwm: OUT STD_LOGIC);
```

```
END COMPONENT;
```

```
SIGNAL reset1,reset2,reset3 : STD_LOGIC;
```

```
BEGIN
```

--Instanciado nominal de componentes

U1:pwm_fpga PORT MAP(clock => clockSys, reset => reset1, data_value => DataSys,
pwm => pwm1);

U2:pwm_fpga PORT MAP(clock => clockSys, reset => reset2, data_value => DataSys,
pwm => pwm2);

U3:pwm_fpga PORT MAP(clock => clockSys, reset => reset3, data_value => DataSys,
pwm => pwm3);

PROCESS(resetSys)

--las salidas pwm (conectadas en la parte baja del puente) siempre están
funcionando, las salidas Q (conectadas en la parte alta) respetan la sig. secuencia
según el estado de los HALLs

BEGIN

IF (resetSys ='1') THEN

reset1 <= '1';

reset2 <= '1';

reset3 <= '1';

END IF;

END PROCESS;

PROCESS(HALLs,sentido)

BEGIN

IF(sentido = '0') THEN --sentido horario

CASE HALLs IS

WHEN "001" =>

salidasQ <= "001"; reset3 <= '0'; reset2 <= '1'; reset1 <= '1';

WHEN "000" =>

salidasQ <= "001"; reset3 <= '1'; reset2 <= '0'; reset1 <= '1';

WHEN "100" =>

salidasQ <= "100"; reset3 <= '1'; reset2 <= '0'; reset1 <= '1';

WHEN "110" =>

salidasQ <= "100"; reset3 <= '1'; reset2 <= '1'; reset1 <= '0';

WHEN "111" =>

salidasQ <= "010"; reset3 <= '1'; reset2 <= '1'; reset1 <= '0';

WHEN "011" =>

salidasQ <= "010"; reset3 <= '0'; reset2 <= '1'; reset1 <= '1';

WHEN OTHERS => NULL;

END CASE;

END IF;

IF(sentido = '1') THEN --sentido antihorario

CASE HALLs IS

WHEN "011" =>

salidasQ <= "100"; reset3 <= '1'; reset2 <= '0'; reset1 <= '1';

```
        WHEN "111" =>

            salidasQ <= "001"; reset3 <= '1'; reset2 <= '0'; reset1 <= '1';

        WHEN "110" =>

            salidasQ <= "001"; reset3 <= '0'; reset2 <= '1'; reset1 <= '1';

        WHEN "100" =>

            salidasQ <= "010"; reset3 <= '0'; reset2 <= '1'; reset1 <= '1';

        WHEN "000" =>

            salidasQ <= "010"; reset3 <= '1'; reset2 <= '1'; reset1 <= '0';

        WHEN "001" =>

            salidasQ <= "100"; reset3 <= '1'; reset2 <= '1'; reset1 <= '0';

        WHEN OTHERS => NULL;

    END CASE;

END IF;

END PROCESS;


END ARCHITECTURE estrucMotorContol;
```

Codigo VHDL Test Motor Control

```
LIBRARY ieee;

use ieee.std_logic_1164.all;

ENTITY Test_MotorControl IS

END Test_MotorControl;
```


ARCHITECTURE archite_Test_MotorControl OF Test_MotorControl IS

COMPONENT MotorControl

PORT (

clockSys: in std_logic;

resetSys: in std_logic;

sentido: in std_logic;

DataSys: in std_logic_vector(7 downto 0);

HALLs: in std_logic_vector(2 downto 0);

pwm1,pwm2,pwm3: out std_logic;

salidasQ: out std_logic_vector(2 downto 0)

);

END COMPONENT;

-- Internal signal declaration

SIGNAL sig_clock : std_logic;

SIGNAL sig_reset : std_logic;

SIGNAL sig_sentido : std_logic;

SIGNAL sig_data_value : std_logic_vector(7 downto 0);

SIGNAL sig_HALLs : std_logic_vector(2 downto 0);

SIGNAL sig_pwm1, sig_pwm2, sig_pwm3 : std_logic;

SIGNAL sig_salidasQ : std_logic_vector(2 downto 0);

shared variable ENDSIM: boolean:=false;

constant clk_period:TIME:=200 ns;

```
BEGIN

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;

-- Instantiating top level design Component pwm_fpga

inst_MotorControl : MotorControl

PORT MAP(

    clockSys    => sig_clock,

    resetSys    => sig_reset,

    sentido => sig_sentido,

    DataSys     => sig_data_value,

    HALLs => sig_HALLs,

    pwm1        => sig_pwm1,

    pwm2 => sig_pwm2,

    pwm3 => sig_pwm3,

    salidasQ => sig_salidasQ

);
```

stimulus_process: PROCESS

-- la idea de este test es simular el giro del motor a una velocidad real por ejemplo
1200 RPM

-- para esto calculo que el tiempo entre las secuencia del los Halls es aprox. 8ms

-- $1200/60 = 20$ vueltas x segundo 1 vuelta = 0,05 segundos como son 6 estados $0,05/6$
= 8,333 ms

BEGIN

sig_sentido <= '0'; --sentido horario

sig_reset <= '1';

wait for 500 ns;

sig_reset <= '0';

sig_data_value <= "10000000"; --duty 25%

wait for 500 ns;

for i in 1 to 5 loop --simulo 5 vueltas del motor(sentido horario)aprox 1200RPM

sig_HALLs <= "001";

wait for 8 ms;

sig_HALLs <= "000";

wait for 8 ms;

sig_HALLs <= "100";

wait for 8 ms;

sig_HALLs <= "110";

wait for 8 ms;

sig_HALLs <= "111";

wait for 8 ms;

sig_HALLs <= "011";

wait for 8 ms;

```
end loop;

wait for 1000 ns;

sig_sentido <= '1'; --sentido antihorario

for i in 1 to 5 loop --simulo 5 vueltas del motor(sentido antihorario)

    sig_HALLs <= "011";

    wait for 7 ms;

    sig_HALLs <= "111";

    wait for 7 ms;

    sig_HALLs <= "110";

    wait for 7 ms;

    sig_HALLs <= "100";

    wait for 7 ms;

    sig_HALLs <= "000";

    wait for 7 ms;

    sig_HALLs <= "001";

    wait for 7 ms;

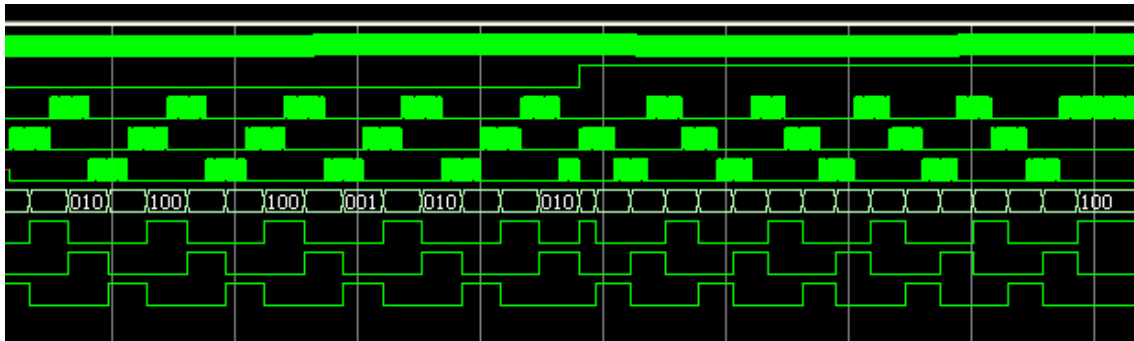
end loop;

wait;

END PROCESS stimulus_process;

END archite_Test_MotorControl;
```

Resultados Simulados:



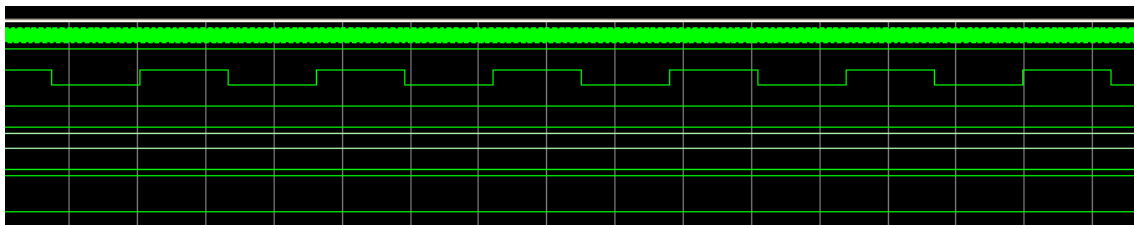
Donde la primera señal ("mancha verde") es el clock. Es lógico que se vea de esta manera ya que es mucho más rápido que el resto de las señales.

La segunda indica el sentido "0" sentido horario, "1" sentido anti horario.

Las siguientes tres señales son los tres PWM.

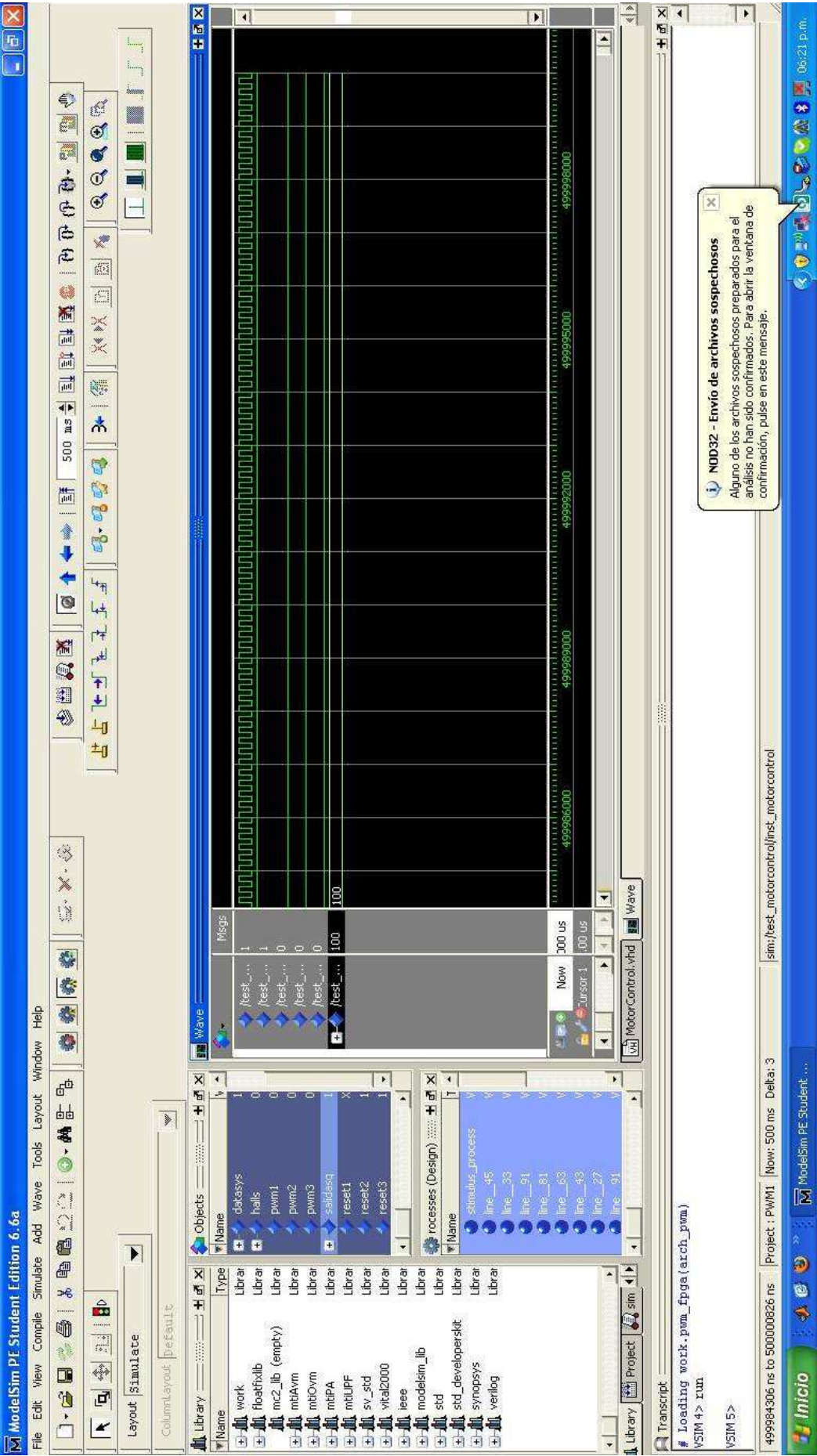
Las últimas tres son las señales de activación de los transistores de la parte alta del puente, estas respetan la secuencia indicada por los sensores de efecto Hall.

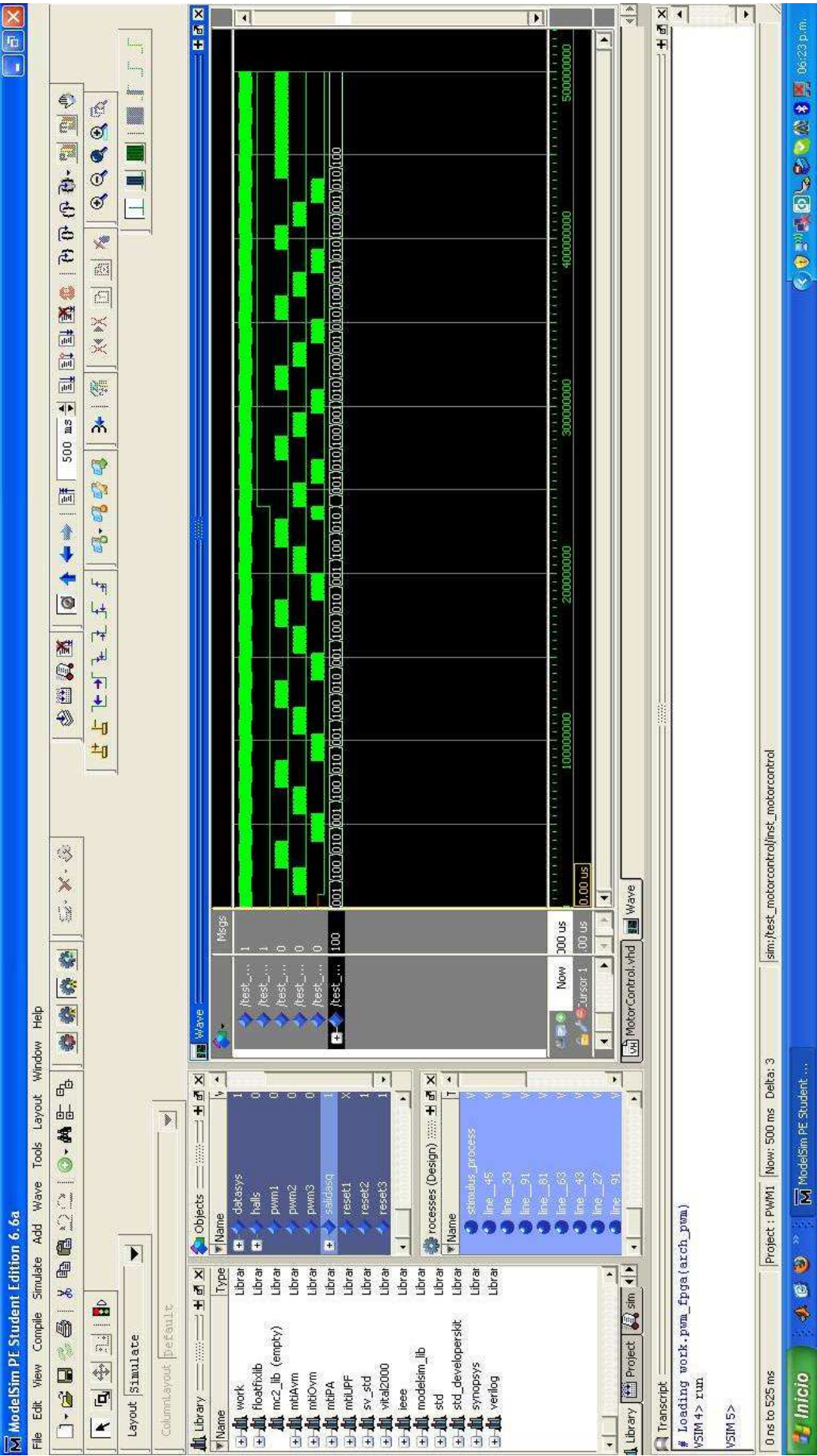
Por ultimo en esta figura se aprecia como al cambiar el sentido se invierte la secuencia de excitación de los transistores (en test se realizan cinco vueltas en un sentido y cinco en el otro).

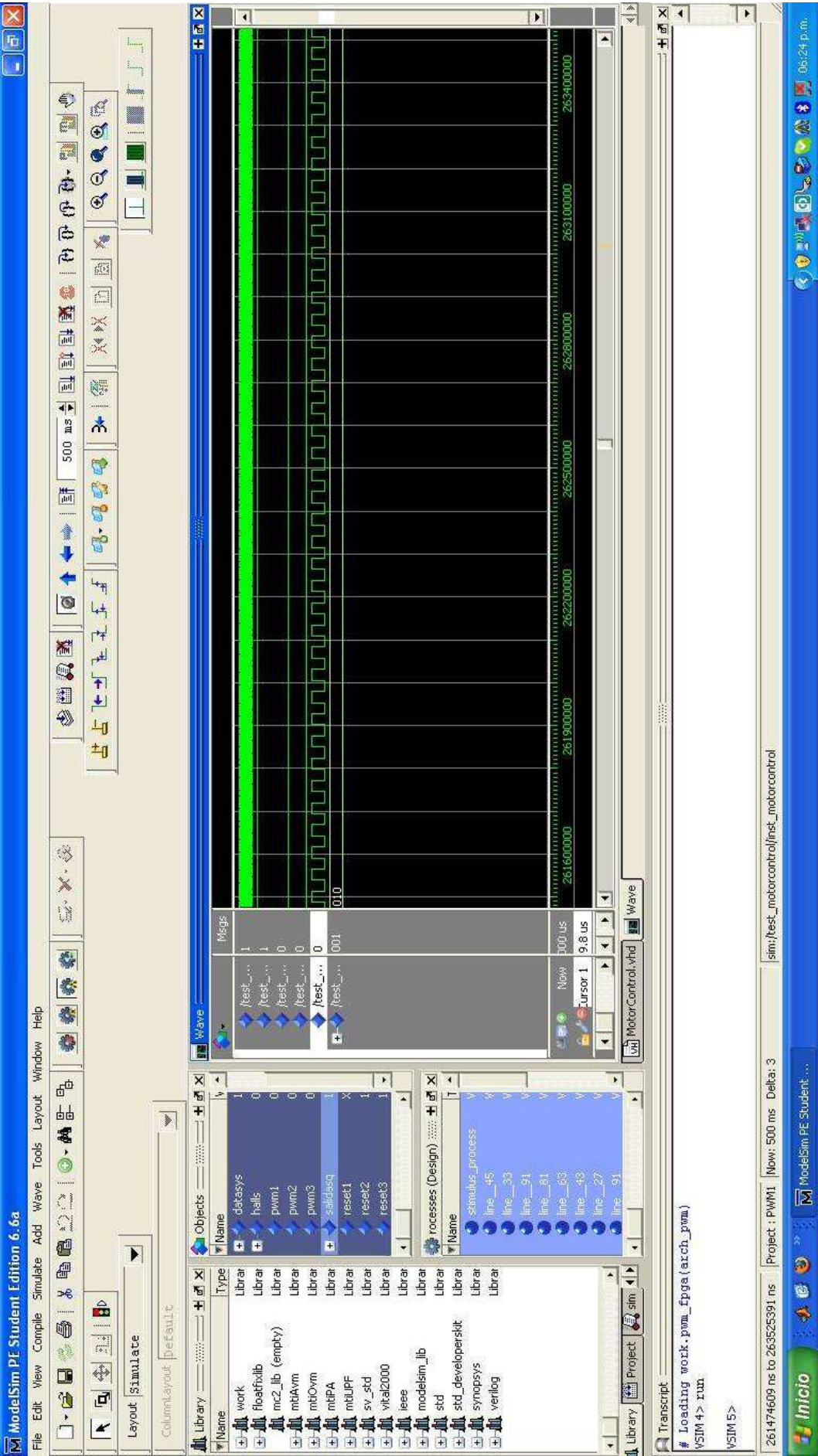


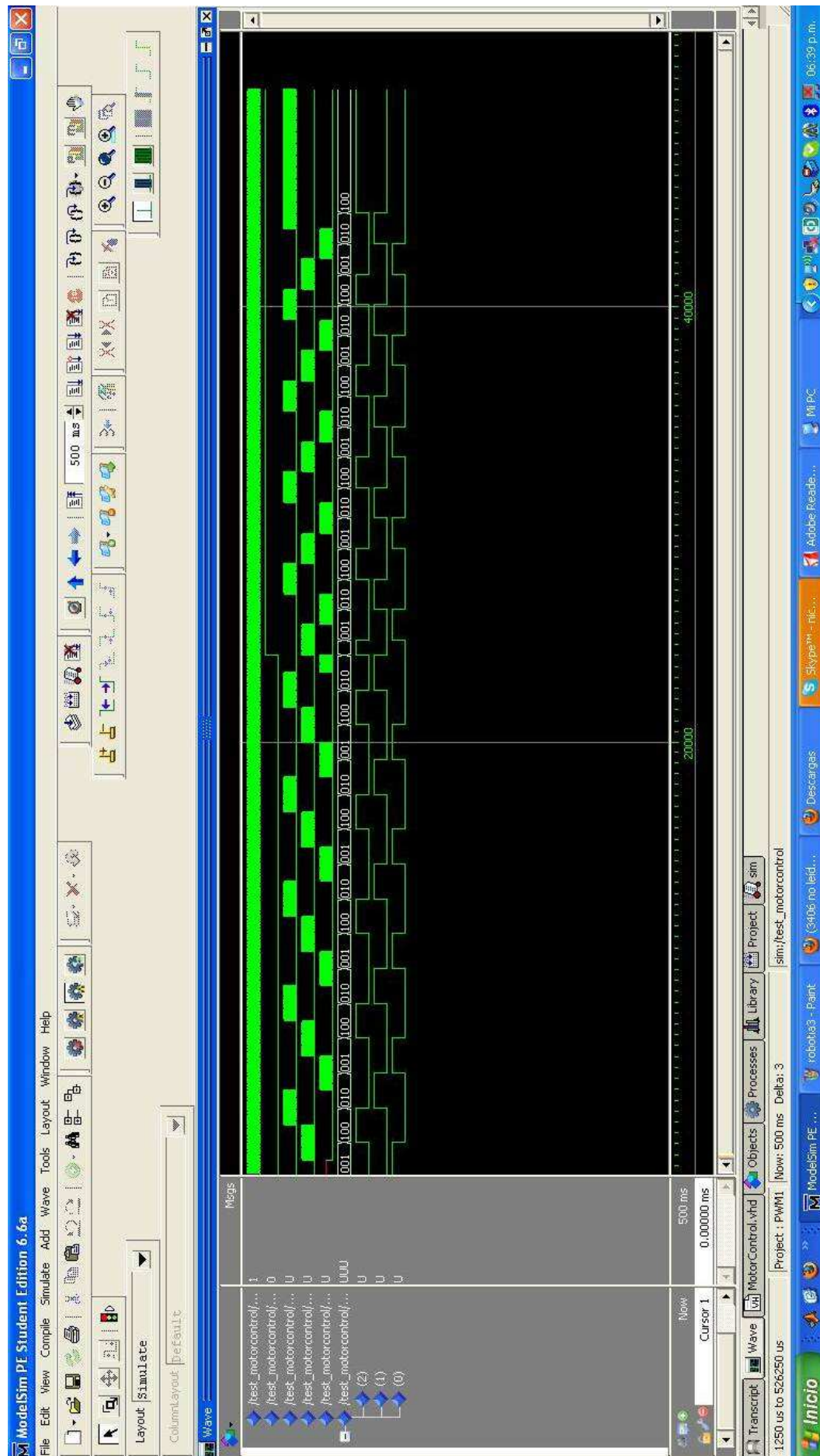
En esta imagen se aprecia una de las señales de PWM, que en este caso esta al 50%, esta señal tiene una frecuencia aprox. de 20khz esto es así para evitar zumbidos en el motor, aunque este valor es aproximado y debe ajustarse según la aplicación y el motor de que se trate.

A continuación ponemos algunas impresiones de pantalla extraídas del programa ModelSim que utilizamos para realizar la simulación del VHDL.









Compilador

-Programa que se encarga de buscar y ejecutar las instrucciones que correspondan con el léxico y la gramática del lenguaje de que se trate.

En este caso implementamos un lenguaje para controlar nuestro robot, para ello utilizamos los programas Lex y Bison, que a partir de la descripción del léxico (Lex) y la gramática (Bison) generan código en lenguaje "C" que luego compilándolo con el Gcc generamos el ejecutable.

En resumen nuestro compilador, llamado PP, se encarga de buscar instrucciones de nuestro lenguaje en un archivo.txt y al encontrarlas las inserta en una lista para luego ejecutarlas.

Como resultado genera tres archivos "out.txt", donde se imprimen los resultados de la ejecución de las instrucciones, "debug.txt", donde se imprimen el estado y numero de línea en donde se encuentra el proceso de compilación, indicándose si se produjeron errores o no, de esta manera el usuario puede identificar y corregir los posibles errores, y "Graficacion.m" que es un script de Matlab que permite graficar el recorrido del robot (consecuencia de las distintas instrucciones de movimiento que se indicaron).

Modo de uso:

Escribir en un archivo de texto la lista de instrucciones que se desean procesar y luego hay dos opciones;

La primera:

En una consola tipo DOS (inicio->ejecutar->cmd) , posicionado en la carpeta donde se encuentra PP.exe, se lo invoca de la siguiente manera " PP nombre del archivo.txt ".

La segunda:

Simplemente haciendo doble click sobre PP.exe, en este caso se buscara un archivo llamado robot.pp (este debe contener las instrucciones deseadas) para procesar, de no encontrarlo se lo informa y se termina la ejecución.

Instrucciones Implementadas:

- "ImprimeConNewLine": imprime en el archivo de salida el texto que se le pase como argumento y queda apuntando a una nueva línea.

Ej. `ImprimeConNewLine("testing");`

- “Imprime”: imprime en el archivo de salida el texto que se le pase como argumento.

Ej. `Imprime("testing");`

- “Movcd”: instrucción de movimiento de cinemática directa, se le pasan como argumento tres desplazamientos (tita1, tita2 y d3), como resultado devuelve el tiempo y el valor del PWM de cada motor, ya que el proceso de movimiento se realiza en tres fase se informaran los tiempos de cada fase y su correspondiente valor de PWM con su sentido.

Ej. `Movcd(10.5,43.2,5.4);`

El resultado es de la forma:

Sun Aug 22 13:35:46 2010

Los valores ingresados son

q1= 10.500000

q2= 43.200001

d3= 5.400000

valor inicial X: 0.000000

valor inicial Y: 0.000000

valor inicial Z: 0.000000

valor final X: -0.010054

valor final Y: 4.318350

valor final Z: -3.936433

posición final Q1: 10.500000
5.400000

posición final Q2: 43.200001

posición final D3:

sentido motor1: 0

sentido motor2: 0

sentido motor3: 0

PWM1:

255

170

86

0

tiempo etapa1: 2.380952

tiempo etapa2: 1.595238

tiempo etapa3: 0.809524

PWM2:

255

170

86

0

tiempo etapa1: 0.578704

tiempo etapa2: 0.387731

tiempo etapa3: 0.196759

PWM3:

255

170

86

0

tiempo etapa1: 4.629629 tiempo etapa2: 3.101852 tiempo etapa3: 1.574074

Esta ultima instrucción está pensada para ser usa en conjunto con el modulo PWM implementado en VHDL.

Código Fuente del Léxico (CompiladorPP.I)

```
%{  
  
//includes  
  
#include <stdlib.h>  
  
#include <stdio.h>  
  
#include <string.h>  
  
#include "y.tab.h"  
  
extern FILE *yyout;  
  
%}  
  
Letras      [a-zA-Z/\.-_ ]  
Num         [0-9]  
Espacio     [ \t]*  
Comilla     ["]  
Numeros     -?{Num}+("."{Num}*)?  
Comentarios "//"+(.)*?  
Descripciones [{Letras}{Num}{Espacio}]  
  
%%  
  
{Espacio}      {return(ESPACIO);}   
  
{Numeros}      {  
                    
                  yylval.fotante = atof(yytext);  
                    
                  return(NUMERO);  
                    
                }  
  
{Comilla}      {return(COMILLA);}   
  
"("            {return(PARENTESIS_IZQ);}
```

```
)"          {return(PARENTESIS_DER);}

 "["        {return(CORCHETE_IZQ);}

 "]"        {return(CORCHETE_DER);}

\n          {return(ENDLINEA);}

";"         {

              fprintf(yyout,"fin de instruccion detectado\n");

              return(ENDINSTRUCCION);

            }

","         {return(COMA);}

"Imprime"   {

              fprintf(yyout,"comando de impresion detectado\n");

              return(MNEMONICO_DYS);

            }

"ImprimeConNewLine"   {

              fprintf(yyout,"comando de impresion detectado\n");

              return(MNEMONICO_DYS_NL);

            }

"Movcd"     {

              fprintf(yyout,"comando de movimiento detectado\n");

              return(MNEMONICO_MOVCD);
```

```
}
```

```
"Start"          {return(INICIO);}
```

```
"EOF"           {return(ENDFILEINSTRUCCTION);}
```

```
{Comentarios}   {return(COMENTARIO);}
```

```
{Letras}+       {  
                  fprintf(yyout,"cargando textos\n");  
                  yyval.txt = (char *)malloc(255 * sizeof(char));  
  
                  strcpy(yyval.txt, yytext);  
                  return(TEXTO);  
                }
```

```
.               {return(NO_IMPLEMENTADO);}
```

```
%%
```

```
int yywrap(void) {  
    return 1;  
}
```

Código Fuente de la Gramática (CompiladorPP.y)

```
%{  
  
//Includes  
  
#include <stdlib.h>  
  
#include <stdio.h>  
  
#include <string.h>  
  
  
#include "compiladorPP.h"  
  
  
// estructura de las instrucciones a compilar  
  
struct instruccion  
{  
  
    int mnemonico;  
  
    char * texto;  
  
    float *argNumericos;  
  
    struct mnemonico *sig;  
  
};  
  
//variables internas  
  
int nroDeLinea=0;  
  
int nroDeInstruccion=0;  
  
int contErrores=0;  
  
FILE *outFile;  
  
struct instruccion *inicial=NULL;  
  
extern FILE *yyin,*yyout;  
  
extern char contMovValidos;  
  
float tmpPosicion[3];
```



```
//prototipos

int yyerror(char *s);

void newInstrucction (int,char*,float*,char);

void compilarListaDeInstrucciones(void);

void limpiarListaDeInstrucciones(void);

%}

%union{

    float fotante;

    char* txt;

};

%token <fotante> NUMERO

%token <txt> TEXTO

%token ESPACIO

%token PARENTESIS_IZQ

%token PARENTESIS_DER

%token CORCHETE_IZQ

%token CORCHETE_DER

%token COMA

%token COMILLA

%token MNEMONICO_DYS

%token INICIO
```

%token ENDFILEINSTRUCCTION

%token COMENTARIO

%token NO_IMPLEMENTADO

%token ENDLINEA

%token MNEMONICO_DYS_NL

%token MNEMONICO_MOVCD

%token ENDINSTRUCCION

%left COMA

%start Input

%%

Input: Line

 | Input Line
 ;

```
Line:  ENDLINEA      {  
                        nroDeLinea++;  
                        fprintf(yyout,"nueva linea numero:");  
                        fprintf(yyout,"%d\n",nroDeLinea);  
                        }
```

```
| MNEMONICO_DYS_NL PARENTESIS_IZQ COMILLA TEXTO COMILLA
PARENTESIS_DER ENDINSTRUCCION {

    nroDeInstruccion++;

    fprintf(yyout,"Agregando a la lista instruccion
numero:");

    fprintf(yyout,"%d\n",nroDeInstruccion);

newInstruccion(MNEMONICO_DYS_NL,$4,NULL,0);

}

| MNEMONICO_DYS PARENTESIS_IZQ COMILLA TEXTO COMILLA
PARENTESIS_DER ENDINSTRUCCION {

    nroDeInstruccion++;

    fprintf(yyout,"Agregando a la lista instruccion
numero:");

    fprintf(yyout,"%d\n",nroDeInstruccion);

    newInstruccion(MNEMONICO_DYS,$4,NULL,0);

}

| MNEMONICO_DYS COMILLA      TEXTO COMILLA

{

    nroDeInstruccion++;

    fprintf(yyout,"Agregando a la lista instruccion
numero:");

    fprintf(yyout,"%d\n",nroDeInstruccion);

    newInstruccion(MNEMONICO_DYS,$3,NULL,0);

}

| MNEMONICO_MOVCD PARENTESIS_IZQ NUMERO COMA NUMERO COMA
NUMERO PARENTESIS_DER ENDINSTRUCCION
```

```
{  
    nroDeInstruccion++;  
    fprintf(yyout,"Agregando a la lista instruccion  
numero:");  
  
    fprintf(yyout,"%d\n",nroDeInstruccion);  
    contMovValidos++;  
    tmpPosicion[0]=$3;  
    tmpPosicion[1]=$5;  
    tmpPosicion[2]=$7;  
    fprintf(yyout," Los valores ingresados son \n %f \n  
%f \n %f \n\n",tmpPosicion[0],tmpPosicion[1],tmpPosicion[2]);  
  
    newInstruccion(MNEMONICO_MOVCD,NULL,tmpPosicion,3);  
}  
  
| COMENTARIO {  
    /**/  
}  
  
| errores {  
    yyerror("instruccion desconcida \n");  
    fprintf(yyout,"ERROR <Linea> %d \n",nroDeLinea);  
    contErrores++;  
    nroDeLinea++;  
    yyerrok;  
}  
  
;
```

errores: error ENDLINEA

```
| NO_IMPLEMENTADO noPerteneceAlLenguaje  
;
```

noPerteneceAlLenguaje: ENDLINEA

```
| error ENDLINEA  
| NO_IMPLEMENTADO noPerteneceAlLenguaje  
| COMENTARIO noPerteneceAlLenguaje  
| NUMERO noPerteneceAlLenguaje  
| PARENTESIS_IZQ noPerteneceAlLenguaje  
| PARENTESIS_DER noPerteneceAlLenguaje  
| COMA noPerteneceAlLenguaje  
  
;
```

%%

```
int main(int argc, char *argv[])
```

```
{
```

```
    if(argc != 2)  
    {  
        if((yyin=fopen("robot.pp","r"))==NULL)  
        {
```

```
        printf("No se puede abrir el archivo de instrucciones\n");  
        exit(1);  
    }  
}else{  
  
    if((yyin=fopen(argv[1],"r"))==NULL)  
    {  
        printf("No se puede abrir el archivo de instrucciones\n");  
        exit(1);  
    }  
}  
  
if((yyout=fopen("debug.txt","w"))==NULL)  
{  
    printf("No se puede abrir el archivo indicado\n");  
    exit(1);  
}  
  
if((outFile=fopen("out.txt","w"))==NULL)  
{  
    printf("No se puede abrir el archivo indicado\n");  
    exit(1);  
}  
  
imprimeFecha(outFile);
```

```
yyparse();

if(contErrores==0)
{
    printf(" No hay errores\n Compilando ..... \n ejecutando ..... \n");
    fprintf(yyout,"No hay errores. Compilando ..... \n");
    compilarListaDeInstrucciones();
}
else
{
    printf("No es posible realizar la compilacion\n ver debug.txt\n");
    fprintf(yyout,"No es posible realizar la compilacion\n");
}

limpiarListaDeInstrucciones();
generaFileM();
fclose(yyin);
fclose(yyout);
fclose(outFile);
if(contErrores==0)
    return 0;
else
    return 1;
```

```
}
```

```
int yyerror(char *s) {
```

```
    fprintf(yyout,"%s\n",s);
```

```
}
```

```
void newInstruccion (int mnemonico,char*txt,float *arg,char cantArgNros)
```

```
{
```

```
    struct instruccion *aux,*procesando;
```

```
    char i;
```

```
    procesando=(struct instruccion*)malloc(sizeof(struct instruccion));
```

```
    if(procesando==(struct instruccion*)(NULL))
```

```
    {
```

```
        printf("sin memoria ..... \n");
```

```
        contErrores++;
```

```
        exit(1);
```

```
    }
```

```
    procesando->sig=(struct instruccion*)(NULL);
```

```
    procesando->mnemonico=mnemonico;
```

```
    procesando->texto=txt;
```

```
    procesando->argNumericos = (float *)malloc(cantArgNros * sizeof(float));
```

```
    for(i=0;i<cantArgNros;i++)
```

```
        procesando->argNumericos[i]=arg[i];
```

```
    aux=inicial;
```

```
    if(inicial==(struct instruccion*)(NULL))
```

```
{
```



```
        inicial=procesando;

    }

    else

    {

        while(aux->sig !=(struct instruccion*)(NULL)){

            aux=aux->sig;

        }

        aux->sig=procesando;

    }

}

void compilarListaDeInstrucciones(void)

{
    int contInstr=0;

    struct instruccion *aux;

    aux=inicial;

    if(aux==(struct instruccion*)(NULL)){

        fprintf(yyout,"no hay instrcciones\n");

        return;

    }

    do

    {

        switch(aux->mnemonico)

        {

            case MNEMONICO_DYS:

                fprintf(yyout,"ejecutando la instruccion numero:");

                fprintf(yyout,"%d\n",contInstr);

                imprimir(outFile,aux->texto);
```

```
        break;

    case MNEMONICO_DYS_NL:

        fprintf(yyout,"ejecutando la instruccion numero:");

        fprintf(yyout,"%d\n",contInstr);

        imprimirConNewLine(outFile,aux->texto);

        break;

    case MNEMONICO_MOVCD:

        fprintf(yyout,"ejecutando la instruccion numero:");

        fprintf(yyout,"%d\n",contInstr);

        moverCinematicaDirecta(outFile,*(aux-
>argNumericos),*((aux->argNumericos)+1),*((aux->argNumericos)+2));

        break;

    }

    aux=aux->sig;

    contInstr++;

}

while(aux!=(struct instruccion*)(NULL));

}

void limpiarListaDeInstrucciones(void)

{

    struct instruccion *aux,*limpiar;

    aux=inicial;

    if(aux==(struct instruccion*)(NULL))

        return;
```

```
while(aux->sig!=(struct instruccion*)(NULL))
{
    limpiar=aux;
    aux=aux->sig;
    free(limpiar);
}
free(aux);
}
```

Código Fuente de la Librería (CompiladorPP.h)

```
#include "time.h"

#include <math.h>

#define PI 3.14159

#define RPM 1000

#define GRADOSxSEGal100 RPM*0.5/60

#define GRADOSxSEGal67 RPM*0.5*0.67/60

#define GRADOSxSEGal34 RPM*0.5*0.34/60


//parametros fijos

#define d1 50

#define d2 20

//parametros maximos

#define q1MAX 135 //grados

#define q2MAX 90 //grados

#define d3MAX 20 //unidades

//parametros minimos

#define q1MIN -135 //grados

#define q2MIN -90 //grados

#define d3MIN -20 //unidades

//representa a los puntos xyz

typedef struct{

    float x;

    float y;

    float z;

}cordenadas;
```

//representa los valores para cada tipo de desplazamiento(Stanford de 3 grados de libertad)

```
typedef struct{
```

```
    float q1;
```

```
    float q2;
```

```
    float d3;
```

```
}posicion;
```

//representa el sentido de giro de cada motor(0:horario, 1:antihorario)

```
typedef struct{
```

```
    char senMot1;
```

```
    char senMot2;
```

```
    char senMot3;
```

```
}sentidos;
```

```
cordenadas puntoDePartida={0.0,0.0,0.0};
```

```
posicion posInicial={0.0,0.0,0.0};
```

```
cordenadas tablaCordenadas[250];
```

```
char contMovValidos=0;
```

```
FILE * fpm;
```

```
extern FILE *yyout;
```

```
void imprimir(FILE * out,char * txt){
```

```
    fprintf(out,"%s",txt);
```

```
}
```

```
void imprimirConNewLine(FILE * out,char * txt){
```

```
    fprintf(out,"%s \n",txt);
```

```
}
```

```
void imprimeFecha(FILE * out){  
  
    time_t tSec = time(NULL);  
  
    time_t* tPtr = &tSec;  
  
    char* sTptr = ctime(tPtr);  
  
    fprintf(out,"%s \n",sTptr);  
  
}  
  
void eperarNseg(float seg){  
  
    unsigned int mSeg=(float)(1000*seg);  
  
    //sleep(mSeg);  
  
}  
  
/*
```

ecuaciones de cinematica directa:

$$X = \cos \hat{l}_1 \sin \hat{l}_2 d_3 \hat{a}'' \sin \hat{l}_1 d_2$$

$$Y = \sin \hat{l}_1 \sin \hat{l}_2 d_3 + \cos \hat{l}_1 d_2$$

$$Z = \hat{a}'' \cos \hat{l}_2 d_3$$

*/

```
void moverCinematicaDirecta(FILE * out,float vq1,float vq2,float vd3 ){  
  
    cordenadas puntoFinal;  
  
    sentidos sentido;  
  
    int pwms[3];  
  
    float q1rad,q2rad;  
  
    float tiempos[3];  
  
    float cteGrdSeg[3];  
  
    char i,j;  
  
    static char contIngresos=1;  
  
    posicion val={vq1,vq2,vd3};
```

```
fprintf(out," Los valores ingresados son \n q1= %f \n q2= %f \n d3= %f
\n\n",val.q1,val.q2,val.d3);

cteGrdSeg[0]=GRADOSxSEGal100;

cteGrdSeg[1]=GRADOSxSEGal67;

cteGrdSeg[2]=GRADOSxSEGal34;


//valores respetan los valores max o minimos??

if((val.q1>q1MAX || val.q1<q1MIN || val.q2>q2MAX || val.q2<q2MIN ||
val.d3>d3MAX || val.d3<d3MIN) || (val.q1==0.0 && val.q2==0.0 && val.d3==0.0))

{

    fprintf(out," Terner en cuenta que: \n");

    fprintf(out," Los valores ingresados no respetan los valores maximos o minimos
establecidos \n");

    fprintf(out," Los valores no pueden ser 0 \n");

    return;

}

//cargo el sentido de cada motor y los valores de posicion final de cada articulacion

if((val.q1)>posInicial.q1)

    sentido.senMot1=0;

else

    sentido.senMot1=1;

if((val.q2)>posInicial.q2)

    sentido.senMot2=0;

else

    sentido.senMot2=1;
```

```
if((val.d3)>posInicial.d3)

    sentido.senMot3=0;

else

    sentido.senMot3=1;

    posInicial.q1 = ((posInicial.q1) + val.q1);
    posInicial.q2 = ((posInicial.q2) + val.q2);
    posInicial.d3 = ((posInicial.d3) + val.d3);

//la nueva posicion esta dentro de los limites??

if((posInicial.q1>q1MAX || posInicial.q1<q1MIN || posInicial.q2>q2MAX
|| posInicial.q2<q2MIN || posInicial.d3>d3MAX || posInicial.d3<d3MIN) || (val.q1==0.0
&& val.q2==0.0 && val.d3==0.0))

{

    fprintf(out,"la posicion final no respetan los valores maximos o minimos establecidos
\n");

    return;

}

fprintf(out,"valor inicial X: %f \t valor inicial Y: %f \t valor inicial Z: %f
\n",puntoDePartida.x,puntoDePartida.y,puntoDePartida.z);

q1rad=val.q1*PI/180.0;

q2rad=val.q2*PI/180.0;

puntoFinal.x= cos(q1rad)*sin(q2rad)*val.d3 - sin(q1rad)*d2;

puntoFinal.y= sin(q1rad)*sin(q2rad)*val.d3 + sin(q1rad)*d2;

puntoFinal.z= -cos(q2rad)*val.d3;

//realizo la diferencia con el punto de partida. Cargo el punto de partida con el nuevo
valor

puntoDePartida.x = puntoDePartida.x + puntoFinal.x;
```



```
puntoDePartida.y = puntoDePartida.y + puntoFinal.y ;

puntoDePartida.z = puntoDePartida.z + puntoFinal.z ;


//imprimo los valores calculados

fprintf(out,"valor final X: %f \t valor final Y: %f \t valor final Z: %f
\n\n",puntoDePartida.x,puntoDePartida.y,puntoDePartida.z);

fprintf(out,"posicion final Q1: %f \t posicion final Q2: %f \t posicion final D3: %f
\n\n", posInicial.q1, posInicial.q2, posInicial.d3);

fprintf(out,"sentido motor1: %d \t sentido motor2: %d \t sentido motor3: %d \n\n",
sentido.senMot1, sentido.senMot2, sentido.senMot3);


tablaCordenadas[contIngresos].x=puntoDePartida.x;

tablaCordenadas[contIngresos].y=puntoDePartida.y;

tablaCordenadas[contIngresos].z=puntoDePartida.z;


contIngresos++;


//suponiendo que el motor mas la caja reductora provacan defasajes de 0,5 grados por
cada vuelta del motor y ademas con duty de 100% el motor gira a 1000 rpm


// como primera aprox ponemos los duty al 100%,lo que prponemos es ir
disminuyendo gradualmente la velocidad(en 3 pasos 33%)


//calculo el tiempo de cada etapa


//primera articulacion

fprintf(out,"PWM1: \t\t");

for(j=0;j<3;j++)

{

    for(i=0;i<3;i++){

        if(val.q1!=0)

            tiempos[i]=(float)(cteGrdSeg[i]*3)/val.q1;
```

```
    else

        tiempos[i]=0;

    }

    pwms[j]= (int)((float)(255*(100-(33*j)))/100);

    fprintf(out,"%d \t\t", pwms[j]);

    //esperarNseg(tiempos[j]);

}

pwms[0]= 0;

fprintf(out,"%d \t\t", pwms[0]);

fprintf(out,"\n");

fprintf(out,"tiempo etapa1: %f \t tiempo etapa2: %f \t tiempo etapa3: %f \n\n",
tiempos[0],tiempos[1], tiempos[2]);

//segunda articulacion

fprintf(out,"PWM2: \t\t");

for(j=0;j<3;j++)

{

    for(i=0;i<3;i++){

        if(val.q2!=0)

            tiempos[i]=(float)(cteGrdSeg[i]*3)/val.q2;

        else

            tiempos[i]=0;

    }

    pwms[j]= (255*(100-(33*j)))/100;

    fprintf(out,"%d \t\t", pwms[j]);
```

```
// eperarNseg(tiempos[j]);

}

pwms[0]= 0;

fprintf(out,"%d \t\t\t", pwms[0]);

fprintf(out,"\n");

fprintf(out,"tiempo etapa1: %f \t tiempo etapa2: %f \t tiempo etapa3: %f \n\n",
tiempos[0],tiempos[1], tiempos[2]);


//tercera articulacion considero que las cte unidades(mm,cmm) por segundo es igual a
lascte grados por segundo

fprintf(out,"PWM3: \t\t");

for(j=0;j<3;j++)

{

for(i=0;i<3;i++){

if(val.d3!=0)

tiempos[i]=(float)(cteGrdSeg[i]*3)/val.d3;

else

tiempos[i]=0;

}

pwms[j]= (255*(100-(33*j)))/100;

fprintf(out,"%d \t\t\t", pwms[j]);

//eperarNseg(tiempos[j]);

}

pwms[0]= 0;

fprintf(out,"%d \t\t\t", pwms[0]);

fprintf(out,"\n");
```

```
fprintf(out,"tiempo etapa1: %f \t tiempo etapa2: %f \t tiempo etapa3: %f \n\n",
tiempos[0],tiempos[1], tiempos[2]);

}

void generaFileM (void){

char buffercontMovValidos;

char i=0;

if(contMovValidos==0)

{

    fprintf(yyout,"No hay instrucciones de movimiento. No se va a genera el archivo.m
de graficacion\n");

    return;

}

if((fpm=fopen("Graficar.m","w"))==NULL)

{

    printf("No se puede abrir el archivo .m\n");

    return;

}

contMovValidos++; //incremento para incluir el punto inicial

buffercontMovValidos=contMovValidos;

fprintf(fpm,"disp('Graficando los puntos recorridos por el robot');\n");

fprintf(fpm,"x=[");

while(contMovValidos>0)

{

    fprintf(fpm,"%f\t",tablaCordenadas[i].x);

    i++;

    contMovValidos--;

}
```

```
fprintf(fpm, "];\n");

contMovValidos=buffercontMovValidos;

i=0;

fprintf(fpm, "y=[");

while(contMovValidos>0)

{

    fprintf(fpm, "%f\t", tablaCordenadas[i].y);

    i++;

    contMovValidos--;

}

fprintf(fpm, "];\n");

contMovValidos=buffercontMovValidos;

i=0;

fprintf(fpm, "z=[");

while(contMovValidos>0)

{

    fprintf(fpm, "%f\t", tablaCordenadas[i].z);

    i++;

    contMovValidos--;

}

fprintf(fpm, "];\n");

fprintf(fpm, "plot3(x,y,z)\n");

fprintf(fpm, "title('Recorrido')\n");

fprintf(fpm, "xlabel('x')\n");

fprintf(fpm, "ylabel('y')\n");

fprintf(fpm, "zlabel('z')\n");
```

```
fclose(fpm);  
  
contMovValidos=0;  
  
}
```