Algoritmo cinemático y lenguaje de programación para Robot SCARA de 5 grados de libertad

Javier Garbini, Ignacio Tamayo
UTN FRBA, Departamento de Electrónica

***Resumen*-Trabajo de diseño, programación y verificación por simulación de robot SCARA de 5 grados de libertad. Se pretende generar un nuevo algoritmo cinemático inverso basado en posiciones, algoritmos de trayectorias simples y apropiadas para cada movimiento y un lenguaje de programación de alto nivel propio orientado a repeticiones. Se usa simulación integrada de todos los elementos diseñados para verificar las trayectorias cumplidas según programas y los movimientos fluidos de las articulaciones.**

I. INTRODUCCIÓN

En el trabajo estudiantil final de la cátedra de Robótica de la carrera de Ingeniería Electrónica se realizó el diseño, programación y simulación de un robot SCARA con 5 grados de libertad. Se tomó como base un trabajo anterior con esta misma arquitectura [1], al cual se le agregó un grado más de libertad. Si bien ya existen algoritmos cinemáticos desarrollados por matrices DH, se buscó un algoritmo que use operaciones simples y comparaciones. Se buscó simplificar la programación del robot, creando un lenguaje de alto nivel similar a los lenguajes comerciales. Para el desarrollo se usó un modelo de simulación en Robotics Toolbox 9.9 de Peter Corke [2] sobre MatLab™ 2008. Para el lenguaje de programación se usó el software ANTLRWorks[3] en conjunto con lenguaje Java™.

II. MEMORIA DESCRIPTIVA

Como toda configuración SCARA, la arquitectura elegida está formada por dos articulaciones de rotación con respecto a dos ejes paralelos entre sí y perpendiculares al plano de trabajo, una articulación de desplazamiento en dirección paralela a la de los ejes de rotación y perpendicular al plano de trabajo, y una articulación de desplazamiento en sentido perpendicular a los ejes de rotación y paralela al plano de trabajo. Finalmente una última articulación de rotación es colocada en la punta del robot a fin de orientar el muñón final. Esto es un modelo SCARA clásico con el agregado de un grado más de libertad, permitiendo que el robot se desplace sobre el eje Y del sistema de referencia inicial, como se ve en la fig. 1.

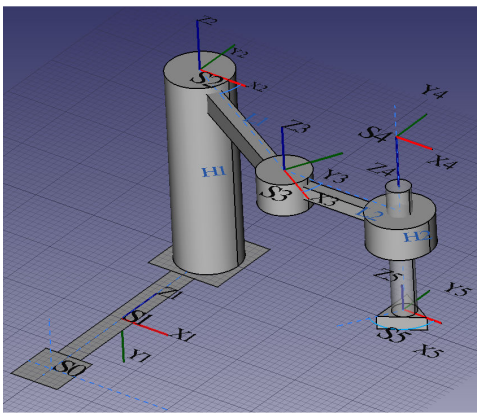

Fig. 1: Modelo cinemático

III. CINEMATICA DEL ROBOT

A. *Cinemática Directa*

Para realizar el cálculo de la cinemática directa se utilizó el algoritmo de Denavit-Hartenberg. Se validó el diseño y las matrices DH obtenidas para el modelo descripto en la fig. 1 usando el Robotics Toolbox, cuyo resultado y simulación se ven en la fig. 2.

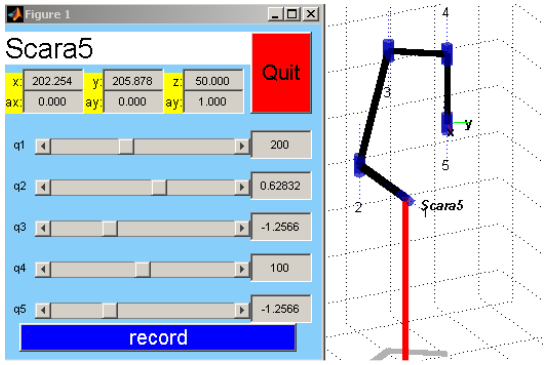

Fig. 2: Simulación del robot Scara5 en Robotics Toolbox

B. *Espacio de trabajo*

Se determinaron los límites de movimiento de las articulaciones según los límites físicos por sus dimensiones, tomando como referentes modelos SCARA comerciales [3]. Los límites del espacio de trabajo están formados por un círculo interior y exterior que no alcanza el robot en una dada posición del eje Y, según la fig. 3. Sin embargo, los círculos interiores se pueden alcanzar desplazando la articulación sobre el eje Y de manera que el punto espacial sea alcanzable fuera de los ángulos críticos. Al mover el eje adicional de movimiento se tiene una zona rectangular. En esta premisa se basa el algoritmo cinemático inverso usado para el robot Scara5. Se verificó el espacio de trabajo mediante simulación en MatLab, haciendo uso de la matriz jacobiana para encontrar puntos singulares. El resultado mostró que el espacio de trabajo considerado es válido y no tiene puntos singulares

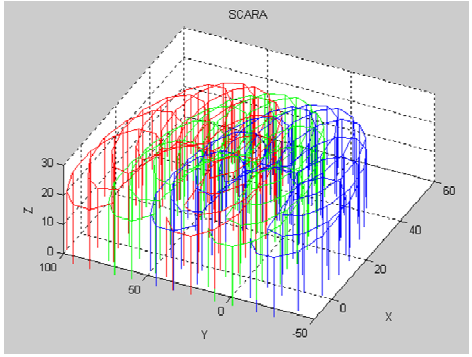

Fig. 3: Imagen de espacios de trabajo superpuestos

C. *Cinemática Inversa*

Las coordenadas espaciales que se definieron para posicionar el robot son **{(Px, Py, Pz), Alfa}**, siendo **(Px, Py, Pz)** las coordenadas del muñón del robot respecto al sistema de referencia del origen del robot y **Alfa** el ángulo de orientación del muñón deseado respecto al sistema de referencia del robot. Se separó el problema en tres: la ubicación en el punto (Px, Py), la ubicación del punto Pz y la orientación del muñón.

El reto reside en el punto (Px, Py) pues el eje principal de rotación del robot se desplaza sobre el eje Y para alcanzar al punto deseado. El sistema de ecuaciones resultante tiene 3 incógnitas y 2 ecuaciones (1), y para resolverlo se diseñó un algoritmo discreto que trabaja por tramos del espacio como se muestra en la fig. 4. El algoritmo diseñado es simple y requiere pocos recursos de procesamiento.

$$P_X = L_1 \cos q_2 + L_2 \cos(q_2 + q_3)$$
$$P_Y = L_1 sen q_2 + L_2 \text{sen}(q_2 + q_3) + q_1 \quad (1)$$

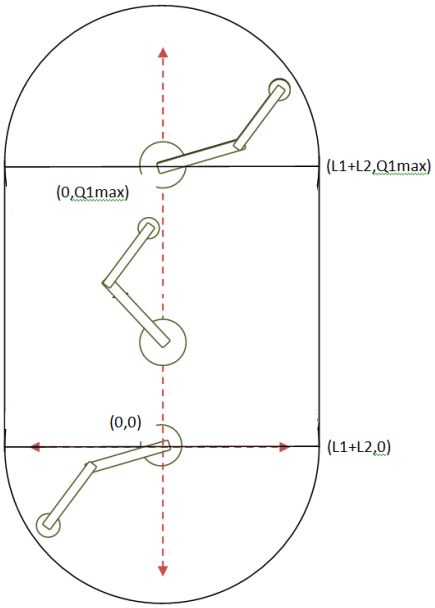

Fig. 4: Zonas de plano XY

Una consideración importante para evitar saltos bruscos en las coordenadas articulares es mantener la orientación de los brazos del robot conforme se mueve, dando finalmente el algoritmo completo implementado en MatLab. El algoritmo se prueba usando el flujo de datos de la fig. 5

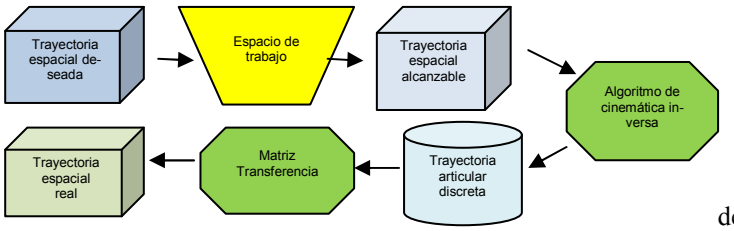

Fig. 5: Flujo de prueba de algoritmo cinemático

D. *Generación de trayectorias articulares*

Para el robot SCARA se usaron dos generadores de trayectorias: las articulaciones para el movimiento XY están en trayectoria polinómica de 3er orden, mientras que las demás articulaciones, más simples y de menor torque, están en trayectoria lineal. La trayectoria de 3er orden evita tener aceleraciones articulares bruscas a costa de mayores requerimientos de procesamiento y mayor complejidad en el control de los motores, mientras que la trayectoria lineal es más fácil de implementar y controlar. La fig. 6 muestra las curvas de ambos generadores de trayectorias aplicadas a articulaciones deferentes.

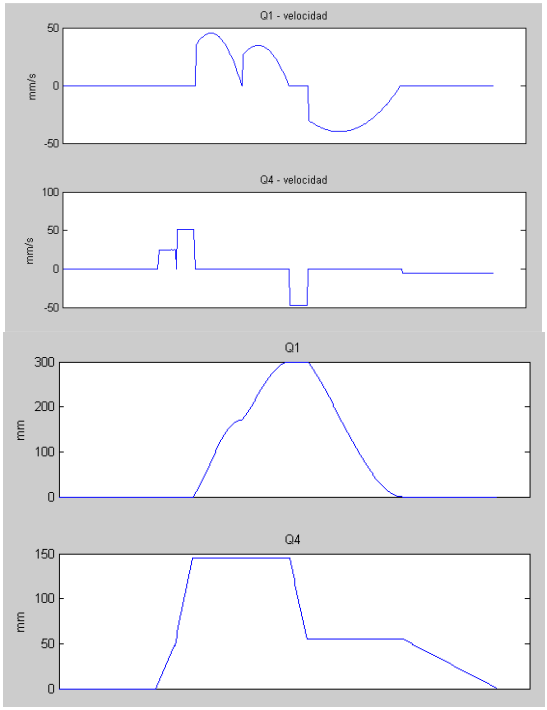

Fig. 6: Velocidad y posición de articulaciones prismáticas

IV. LENGUAJE DE PROGRAMACION

A fin de programar lógicamente el conjunto de movimientos del robot según el algoritmo previo, se diseñó un lenguaje de programación propio con su correspondiente compilador de alto nivel, con un set de instrucciones reducido que pueda ser compilado y traducido en las diferentes señales a enviar a cada articulación del robot, como se muestra la fig. 7.

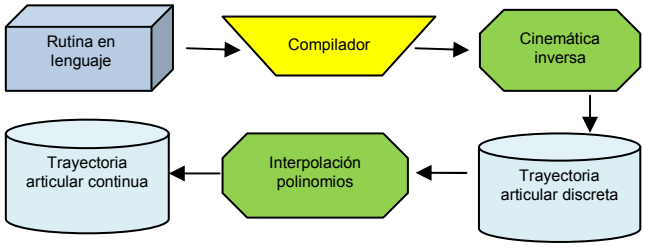

Fig. 7: Flujo de prueba del compilador y algoritmo

El lenguaje de programación desarrollado permite bloques de repetición (loops) de movimientos. Las instrucciones de movimientos se dividen en el plano XY, luego plano Z, y la orientación angular de la última articulación para posicionar la pieza a manipular como muestra la fig. 8. Cada instrucción tiene un tiempo definido para dicho movimiento, dando la posibilidad de programar la duración total de la rutina. Las señales

de control enviadas a los drivers de los motores tienen diferentes valores de velocidad según el tiempo definido para cada movimiento, como se muestra en la fig. 6, y dicha señal de control refleja la trayectoria polinómica y sin saltos bruscos que se diseñó a través del algoritmo cinemático inverso.

```
HOME [espacios]   [tiempo en segundos];
PAUSE [espacios]   [tiempo en segundos];
MOVEZ [espacios] [+/- valor en mm]  [ , ] [tiempo en segundos];
MOVETITA [espacios] [+/- valor en grados]  [ , ] [tiempo en segundos];
MOVEXY [espacios] [+/- valor en mm]  [ , ] [+/- valor en mm]  [ , ]  [tiempo en segundos];
// [Comentario]
LOOP [espacios] [veces]
[Instrucciones dentro del lazo]
END
```

Fig. 8: Set de instrucciones del lenguaje de programación

V. SIMULACIÓN Y RESULTADOS

Para verificar el algoritmo cinemático inverso diseñado, según el flujo de datos indicado antes, se definieron trayectorias espaciales en MatLab y se verificó que la trayectoria articular obtenida respetaba los puntos a seguir, como se ve en la fig. 9. En el gráfico el trazo azul corresponde a la trayectoria deseada con un punto fuera del espacio, que es ignorado por el trazo rojo, el cual es la trayectoria resultante del algoritmo cinemático inverso simulado

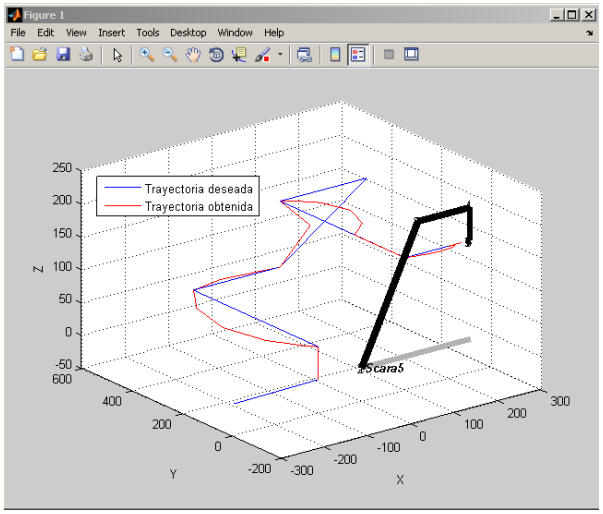

Fig. 9: Trayectoria ejecutada con puntos fuera del espacio de trabajo

De manera final integral, para verificar las señales de movimiento articular generadas con la programación en el set de instrucciones proprio, el compilado y algoritmo cinemático inverso diseñados, se genera un archivo de texto como salida de la simulación del compilador y el algoritmo cinemático (ambos en lenguaje JAVA), que se usa como entrada para el modelo del robot en el Robotics ToolBox, donde se visualizan los movimientos y trayectoria resultantes de todo el proceso. El resultado esperado fue ingresar una trayectoria repetitiva (loop) en el lenguaje de programación propio y al final del compilador y algoritmos verificar que el modelo de simulación efectúe los movimientos programados de manera continua, sin saltos bruscos en sus movimientos ni en las señales de control. La fig. 10 muestra una gráfica del resultado de la simulación, que en realidad es una secuencia continua en movimiento fluido.

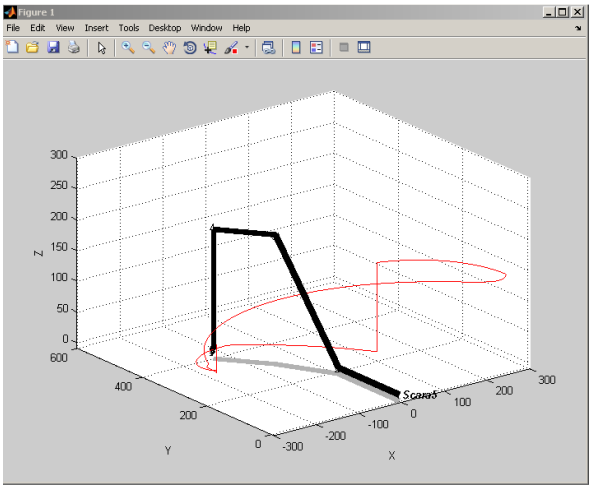

Fig. 10: Resultado de simulación del programa en el lenguaje propio

VI. CONCLUSIONES

Se desarrolló adecuadamente el modelo cinemático del Robot SCARA de 5 grados de libertad, usando el algoritmo de Denavit-Hartenberg. Se desarrolló un nuevo algoritmo cinemático inverso y se usaron generadores de trayectorias articulares polinomiales de 3er orden y 1er orden. Se logró generar un lenguaje de programación propio y un compilador para programar los movimientos del robot, incluyendo la generación de ciclos repetitivos, programado en JAVA. Se verificó mediante una simulación completa el cumplimiento de la trayectoria programada en el lenguaje propio, de manera continua y sin saltos bruscos en los movimientos ni en las señales de control simuladas que serían enviadas a los drivers de los motores, verificando que los algoritmos desarrollados no generan sobreesfuerzos en los motores como se diseñó

REFERENCIAS Y BIBLIOGRAFIA

[1] Alonso, Di Donato, Morella. “Trabajo Integrador Robot Scara”. Cátedra Robótica, UTN-FRBA. 2012.

[2] Peter Corke. “Robotics Toolbox”. http://*petercorke.com/Robotics_Toolbox.html*

[3] ANTLRWorks. http://www.antlr3.org/works/

[4] Epson Scara G Series. http://robots.epson.com

- Spong, Hutchinson y Vidyasagar. “Robot Modeling and Control”. 2005.
- Barrientos, Peñin, Balaguer y Aracil. “Fundamentos de robótica”. 2001.
- Zhenhua Zhao , “The simulation of SCARA robot based on OpenGL and STL”. Mechanic Automation and Control Engineering (MACE), 2011.
- Lee, M.C. “Integrated SCARA robot control system with OLP and its performance evaluation”. SICE 1997

Anexo

Javier Garbini, Ignacio Tamayo
UTN FRBA, Departamento de Electrónica

I. CINEMATICA DEL ROBOT

A. Cinemática Directa

Para realizar el cálculo de la cinemática directa se utilizó el algoritmo de Denavit-Hartenberg, obteniendo la Tabla 1.

TABLA 1
Matrices de Transferencia DH

	θ	**d**	**A**	**α**
T0[1]	0	**0**	0	-90 °
T1	0	**q1**	0	90 °
T2	**q2**	H1	L1	0
T3	**q3**	0	L2	180 °
T4	0	**q4**	0	180 °
T5	**q5**	0	0	0

[1]Se debe agregar una matriz de transformación adicional T0'que sirve solo para orientar el eje de coordenadas en el sentido del desplazamiento de la primera articulación.

Se obtuvo la matriz de transformación homogénea (1), por multiplicación de matrices DH de cada elemento T calculado anteriormente:

```
[cos(q2+q3+q5)    -sin(q2+q3+q5)    0    L2*cos(q2+q3)+L1*cos(q2)]
[sin(q2+q3+q5)    cos(q2+q3+q5)     0    L2*sin(q2+q3)+L1*sin(q2)+q1]
[0                0                 1    -H2-q4+H1]
[0                0                 0    1]                          (1)
```

También se obtuvo su matriz Jacobiana (2), usada para la cinemática de velocidades y para encontrar los puntos singulares del robot.

```
[0    -L2*sin(q2+q3)-L1*sin(q2),   -L2*sin(q2+q3)    0]
[1    L2*cos(q2+q3)+L1*cos(q2)     L2*cos(q2+q3)     0]
[0    0                            0                 -1]              (2)
```

Se validó el diseño y las matrices obtenidas usando el Robotics Toolbox, como se ve en la fig. 1 según el modelo descripto en la Tabla 2.

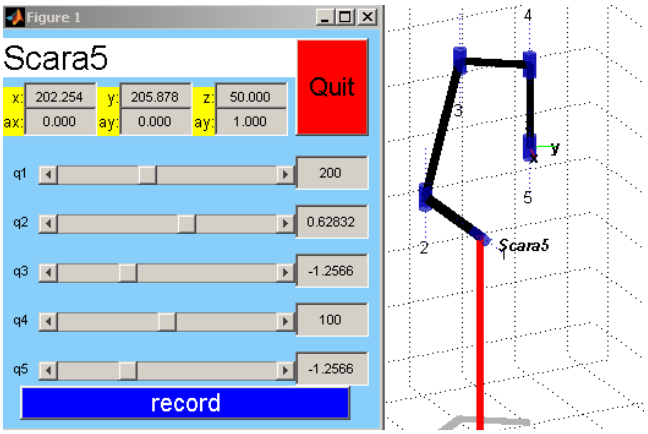

Fig2: Simulación del robot Scara5 en MatLab

TABLA 2
Modelo en Robotics Toolbox

```
Lk0 = Link([0 0 0 -pi/2] ) ;
Lk1 = Link( [ 0 0 0  pi/2 1] );
Lk1.qlim = [Q1min Q1max];
Lk2 = Link ( [ 0  H1  L1 0]);
Lk3 = Link( [ 0 0 L2  pi] );
Lk4 = Link( [ 0 0 0 pi 1 H2] );
Lk4.qlim = [Q4min Q4max];
Lk5 = Link();
Scara5 = SerialLink([ Lk1 Lk2 Lk3 Lk4 Lk5]);
Scara5.name = 'Scara5';
Scara5.base = Lk0.A(0);
Scara5.teach;
```