APLICACIÓN DE MATLAB Y SIMULINK PARA ESTACIONES ROBÓTICAS

Sandra Arévalo Fernández (sandra.arevalo@alumnos.uva.es)
Alberto Herreros López (albher@eis.uva.es)
GIR Tecnologías avanzadas de la producción
EII-Universidad de Valladolid (Spain)

Resumen

Se ha diseñado una herramienta de Matlab/Simulink (SimRob) de uso didáctico para diseñar y simular estaciones robóticas. Dicha aplicación se ha realizado usando la herramienta de Matlab Robotic System Toolbox, la herramienta de Simulink SimScape Multi-Body, así como otros paquetes de software libre de Matlab. La mayoría de las aplicaciones de Matlab relacionadas con Robótica no usan el entorno Simulink en el diseño y en las simulaciones de la estación. La aplicación propuesta intenta usar los recursos de Simulink tanto para el diseño de los componentes de la estación como para la simulación de la misma. El control de la simulación, básicamente su cinemática inversa, ha sido desarrollada con las herramientas de Matlab. Con ello se pretende que el alumno disponga de un entorno agradable para diseñar y simular la estación, y unas herramientas en Matlab para su control. Los resultados de la simulación pueden ser exportados a los robot reales usando distintos tipos de comunicación, por ahora, la comunicación OPC.

Palabras clave: Matlab, Simulink, Robótica, Simulación, Comunicación OPC.

1 Introducción

La robótica se ha convertido en una de las áreas de docencia emergentes en los últimos años debido a su demanda industrial y social. La mayoría de los robot industriales tienen sus propios software de control y simulación. Esto hace que sea difícil la docencia en este área de una forma universal, ya que, normalmente, hay que usar el software de una determinada marca de robots. Un ejemplo significativo son las unidades usadas para medir traslación y rotación. Es casi imposible encontrar dos robot de distinta marca que usen las mismas unidades.

Un robot manipulador puede ser definido como un conjunto de elementos (brazos) en serie con articulaciones (ejes) entre ellos. Los brazos del robot tiene propiedades dinámicas de masa y momentos de inercia y los ejes pueden se normalmente rotacionales (giro) y prismáticos (desplazamiento)[9]. Cada eje de un robot implica normalmente un grado de libertad para poder alcanzar una posición y giro en el espacio cartesiano de su herramienta (posición TCP, *Tool Central Point*). La definición de los ejes de un robot se ha realizado de forma tradicional usando la parametrización de Denavit-Hartenberg (DH) [8]. Las traslaciones y rotaciones de un eje con respecto al anterior son definidas por 4 parámetros $[\theta, d, a, \alpha]$ que definen la rotación en z, desplazamiento en z, desplazamiento en x y rotación en x. Un robot con 6 ejes quedaría definido por una matriz de dimensión $[6 \times 4]$. Recientemente se ha definido los brazos y ejes de un robot en formato URDF (Unified Robot Description Format). El formato URDF define la cinemática, dinámica, ficheros gráficos asociados y colisiones de un robot en un fichero tipo *xml*. Este formato es ampliamente usado en el entorno para el desarrollo de software para robots llamado ROS (*Robot Operating System*) [6].

Se define como cinemática directa la relación que existe entre las posiciones de los ejes de un robot, $\theta = [\theta_1, \ldots, \theta_6]$ en el caso de un robot con 6 ejes rotacionales, y la posición y orientación cartesiana de su posición TCP, por ejemplo $[x, y, z, r_z, r_y, r_x]$ usando ángulos de Cardano [9]. La cinemática inversa puede ser obtenida de forma analítica para un determinado robot, o por optimización local a partir de una posición inicial de ejes θ_0 [10]. La mayoría de las aplicaciones de software usan este segundo método por ser universal para todo robot manipulador.

Una de las primeras herramientas de Matlab que intentó crear un entorno universal para análisis y control de robots fue la aplicación *Robotic Toolbox* de Coker [1] basado en el libro del mismo autor [2]. Dicho autor tiene un amplio historial en curso MOOC y todo tipo de herramientas didácticas. Este software está basado en la parametrización de Denavit-Hartenberg (DH) [2] para definir los ejes del robot, usa algoritmos de optimización local para obtener la cinemática inversa y usa una representación de los robot basada en sus ejes, di-

https://doi.org/10.17979/spudc.9788497498043.625

agrama de hilos, para acelerar las simulaciones.

A nivel nacional, una herramienta de Matlab muy amplia y muy interesante es la aplicación ARTE [3]. Dicho software define la cinemática directa basada en Denavit-Hartenberg (DH) e inversa basada en cálculos analíticos de muchos robot industriales, así como sus características dinámicas. La representación de los robots se realiza usando ficheros gráficos *stl*. Los resultados son muy interesantes, pero el principal problema de esta herramienta es que las gráficas de Matlab no están pensadas para simulación. Esto hace que las simulaciones sean lentas.

La herramienta oficial de Matlab para robótica *Robotic System Toolbox* es relativamente reciente [4]. Se basa en un objeto llamado *RigidBody* que puede guardar la configuración de un robot en su formato, clásico Denavit-Hartenberg (DH), o leyendo un fichero en formato URDF. La simulación de un objeto *RigidBody* puede ser por hilos o con ayuda de ficheros gráficos *stl*, pero en Matlab. El problema de esta herramienta es el mismo que el de ARTE, las gráficas de Matlab no están pesadas para simulación, y las simulaciones obtenidas son muy lentas.

Simulink tiene una herramienta de simulación llamada *SimScape Multibody* [7], pensada para la simulación de mecanismos mecánicos. Dicha herramienta puede leer ficheros en formato URDF y traducir la información de estos ficheros en iconos de inercia y gráficos para cada brazo e iconos de revolución o traslación para los ejes rotacional o prismático. En los iconos de rotación y traslación se pueden definir actuadores (posición o par) como entradas al sistema. La simulación de dicho sistema con las entradas definidas se representa en un entorno gráfico llamado *Mechanics Explores* con una resolución y velocidad equivalente a un software de robot industrial. El problema de esta herramienta es que solo simula, no controla la cinemática directa e inversa del sistema. Por ello no es posible mover al robot a un punto deseado, solo simular el movimiento del robot ante unas entradas definidas.

Un fichero de Simulink con elementos *SimScape Multibody* puede ser leído como un objeto *RigidBody* de *Robotic System Toolbox*. Esto permite traducir la estación realizada con la primera herramienta en un objeto con el que podemos controlar los movimientos del sistema, cinemática inversa.

El objetivo de la aplicación propuesta *SimRob* es definir una estación robótica con ayuda de la aplicación *SimScape Multibody*de forma sencilla y con ayuda de iconos ya definidos de robots y herramientas. Tras ello, convertir esta estación a un objeto *RigidBody* para poder controlar la cinemática directa e inversa del mismo. Y por último, simular los movimientos que se deseen en dicha estación representando los resultados en el entorno *Mechanics Explores*.

2 Modelado de la estación robótica

Se han diseñado una serie de iconos de robot, herramientas y piezas para facilitar al alumno a que pueda diseñar su propia estación. Sin embargo, el alumno podría, sin mucho esfuerzo, hacer sus propios iconos.

2.1 Iconos de robots

En la figura 1 se muestra varios iconos con robots ya traducidos al lenguaje de *SimScape Multibody*. En la figura 2 se muestra un detalle de un robot y la figura 4 el brazo de un robot. Se ve que dispone de iconos para definir las articulaciones y otros para definir los componentes gráficos y dinámicos de cada brazo del robot. La entrada de Simulink es la posición de cada eje, con sus derivadas. En la mascara de los iconos de robots se ha puesto una imagen del robot para facilitar su uso por parte de los alumnos.

Estos iconos han sido obtenidos por dos métodos,

- Importando un fichero de formato URDF del robot. De este fichero se obtiene toda la información del mismo.
- Modelando un robot estándar con el formato de Denavit-Hartenberg (DH) y cambiando sus parámetros. En la mascara de estos iconos están definidas la matriz DH y las propiedades dinámicas del robot, ver figura 4. Solo es preciso cambiar estas variables y los ficheros *stl* de las gráficas para obtener el icono de otro robot.

Normalmente, los robots tienen una entrada de Simulink, la posición de cada eje, y una entrada y salida de *SimScape* para acoplar el icono a otros de la estación. En los robot para problemas de control, la entrada Simulink es el par de cada eje y tiene una salida Simulink que mide su posición.

La máscara de cada icono tiene dos variables que deben ser definidas, la posición de la base del robot y la posición inicial de los ejes del robot.

2.2 Iconos de Objetos de trabajo y herramientas

En la figura 5 y 6 se han definido iconos para robot y herramientas. La máscara de estos iconos

https://doi.org/10.17979/spudc.9788497498043.625

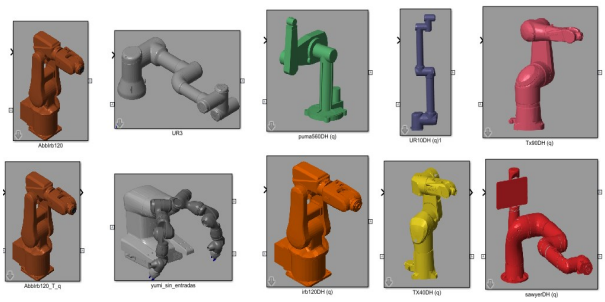

Figura 1: Iconos con diversos robots

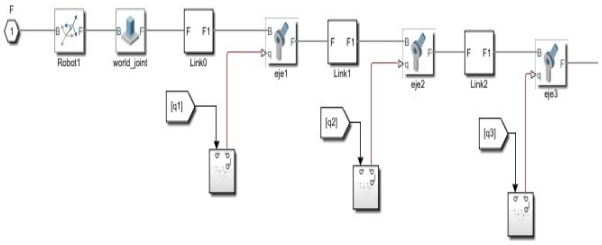

Figura 2: Estructura interna de un robot

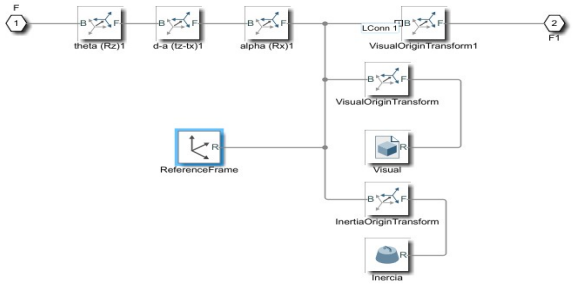

Figura 3: Estructura interna de un brazo del robot

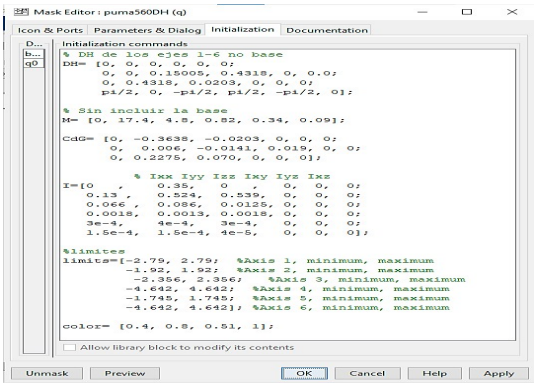

Figura 4: Máscara de un robot con parametrización DH

tiene una variable tipo estructura donde se definen las características principales del icono. Estas son: Posición base, posición TCP, color y características dinámicas. Se dispone de un función *Piezas* para definir esta estructura por defecto.

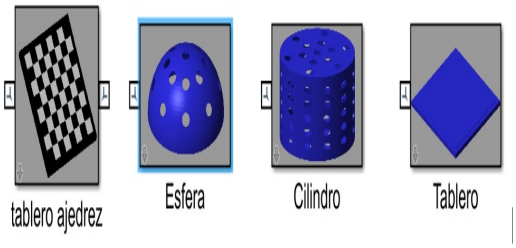

Figura 5: Iconos de piezas

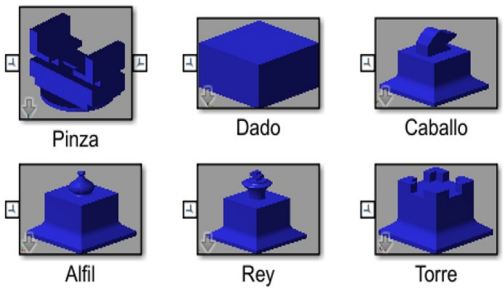

Figura 6: Iconos de herramientas y cargas

2.3 Composición de una estación estática

Para modelar la estación se puede acudir a los iconos ya definidos o definir otros similares. Se deben introducir los iconos base de *SimScape*, referencia total, propiedades de gravedad y el *solver* de las ecuaciones diferenciales. Luego se introducen los iconos propios de la estación.

En la figura 7 se ha definido una estación estática compuesta por un tablero, un caballo, un rey, una torre y un alfil. El caballo y el rey tienen como referencia el tablero, mientras que la torre y el alfil tienen como referencia la base. Si modificamos la posición inicial del tablero caballo y rey también se mueven, ver figura 8. Si movemos caballo y rey lo haremos respecto del tablero.

2.4 Composición de una estación simple con robot

En la figura 9 se ha definido una estación con un robot ABB irb120, que tiene una pinza como herramienta y un lapiz como carga. La entrada de Simulink son los ejes del robot y el resto de entradas y salidas son conexiones de *SimScape*. La

https://doi.org/10.17979/spudc.9788497498043.625

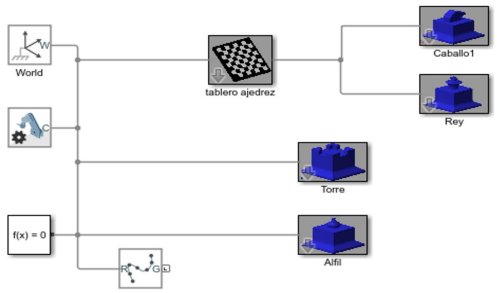

Figura 7: Estación estática de tablero con piezas

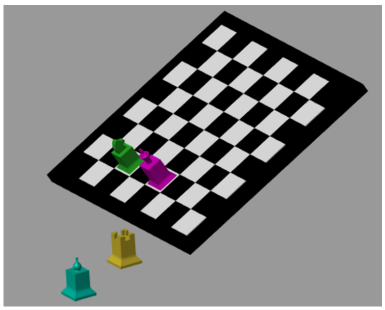

Figura 8: Simulación estación estática de tablero con piezas

figura 10 muestra la posición final de una simulación. La pinza va unida al sexto eje del robot y el lápiz al TCP de la pinza. Se ha girado 30^o grados en el eje 1 y 90^o grados en el eje 5. En este ejemplo solo se ha simulado los cambios en los ejes del robot, no se ha intentado buscar un punto en el espacio cartesiano.

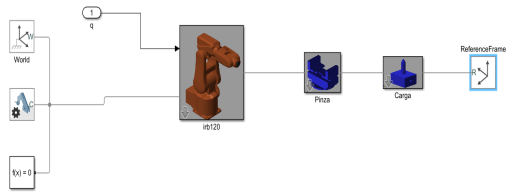

Figura 9: Estación con robot ABB irb120

3 Elementos de control de una estación

La estación que hemos modelado con *SimScape* es útil para simulación de cinemática directa, pero no se puede usar para que el robot describa una trayectoria deseada. Para ello se han definido dos objetos *Hmat* y *Kin* que nos ayuden en la cinemática inversa.

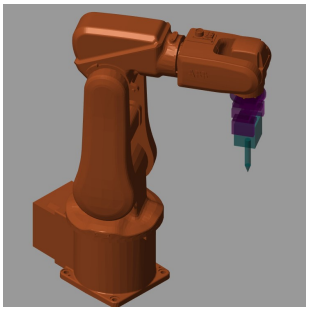

Figura 10: Simulación de estación con robot ABB irb120

3.1 Objeto *Hmat*

Se ha definido un objeto matemático para poder hacer trasformaciones entre diferentes formatos de definición de traslación y rotación. La propiedad principal del objeto es la matriz homogénea. Los métodos de este objeto han sido obtenidos básicamente de la aplicación de Corke. Se pueden dividir en diferentes apartados,

- Cambios de formato: Se puede pasar de ángulos de Euler, Cardano, cuaternios y vectores.
- Trasformación de puntos y posiciones de un eje a otro.
- Tratamiento de la matriz de Denavit-Hartenberg (DH).
- Trayectorias entre posiciones.

3.2 Objeto *Kin*

Para poder usar la aplicación *Robotic System Toolbox* es preciso convertir el modelo de Simulink en un objeto *RigidBody*. Esto es posible con la misma función con la que se importan fichero URDF *importrobot*. Los componentes *body* del objeto obtenido son los iconos de la estación de Simulink, numerados en forma de lista, *body1, body1, ...* Es preciso apuntar los componentes interesantes de la estación, por ejemplo, el TCP del robot o de la herramienta y carga.

El objeto *Kin* requiere el nombre de la estación *SimScape*, el objeto *RigidBody* de la estación y el *body* que en ese momento va a ser el TCP del robot. Con ello este objeto va a poder controlar la estación y simularla de la siguiente forma,

- Puede calcular la cinemática inversa de una trayectoria.
- Puede simular esa trayectoria en el modelo Simulink a una velocidad dada.

https://doi.org/10.17979/spudc.9788497498043.625

Los métodos principales del objeto *Kin* son las sentencias de movimiento de todo robot comercial. Los movimientos en coordenadas cartesianas pueden ser respecto de la base (por defecto) o respecto a una referencia dada.

- *MoveAbsJ*: Mueve el robot de su posición actual a otra por coordenadas articulares.
- *MoveJ*: Mueve el robot de su posición actual a otra por coordenadas cartesianas con formato $[x, y, z, r_z, r_y, r_x]$. Si solo se introduce la posición, la orientación permanece.
- *MoveL*: Mueve el robot de su posición actual a otra por coordenadas cartesianas con formato $[x, y, z, r_z, r_y, r_x]$ en linea recta.
- *MoveC*: Mueve el robot de su posición actual, pasando por otro a un tercero en coordenadas cartesianas con formato $[x, y, z]$ describiendo un arco de semi-circunferencia. La orientación permanece.

Por defecto se trazan 10 puntos en cualquier trayectoria, para ver el movimiento del robot por medio de un rastreador de *SimScape*. Además, el objeto puede guardar las distintas acciones que se han realizado, para luego simularlas a la vez. Es posible cambiar la velocidad de la simulación. Es posible cambiar el TCP respecto del que se quiere trazar la trayectoria.

3.3 App de control

Con el fin de ayudar a mover el robot y determinar posiciones dentro del espacio de trabajo, se ha desarrollado una app con *appdesinger* de Matlab, tal y como se muestra en la figura 11. Los componentes de la app son los siguientes,

- Importar y exportar datos: Importa y exporta el objeto *Kin* del robot que se quiere mover y el punto donde está el robot.
- Búsqueda de un puntos $[x, y, z]$: Se puede buscar un punto apretando los botones correspondienes, con un determinado paso (variable).
- Cambio de posición del robot: Se puede mover el robot al punto deseado con dos tipos de movimientos *MoveJ* y *MoveL*.

La app es universal porque mueve puntos, no ejes del robot. Dependiendo del robot, el objeto *Kin* correspondiente, mueve el robot de una determinada forma al punto elegido.

Figura 11: App para mover el robot

4 Ejemplo de estación Multi-Robot

La estación consta de dos robot con pinzas, ver figura 12. Uno de los robot no tiene carga al inicio, pero va a tomar y dejar piezas de ajedrez. El segundo tiene un lápiz de carga y va a describir trayectorias sobre una mesa inclinada. Al importar con *RigidBody* la estación tenemos en el mismo objeto los dos robots. Es preciso cambiar de TCP dependiendo del robot que queramos mover. No es posible mover ambos robots simultáneamente.

4.1 Tomar y dejar piezas

El primer robot tiene una carga nula, en realidad es una esfera de radio mínimo. Cuando el robot quiere tomar una pieza (torre) debe ir a la pieza y en ese momento se intercambia el contenido de la pieza (torre) por el contenido de la carga (nulo). Para ello se usa la función *MoverEn*. Como la pieza (torre) ya está en la carga, los movimientos del robot implican los movimientos de la carga. Lo mismo, pero en sentido inverso, se realizará cuando se quiera dejar una pieza.

4.2 Describir trayectorias sobre la mesa inclinada

El segundo robot se mueve respecto a los ejes de la mesa inclinada con movimientos lineales y circulares que va guardando. Al final, muestra la trayectoria completa. Parece que está escribiendo sobre la mesa, ver figura 13.

5 Ejemplo de estación con control de sistemas

La figura 14 muestra un ejemplo de control de robot y análisis de la dinámica de una estación. El robot de la estación en este caso tiene como entrada del icono de Simulink, el par de cada eje, y tiene como salida, la posición actual del robot.

https://doi.org/10.17979/spudc.9788497498043.625

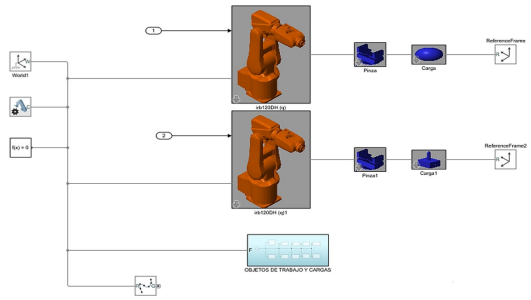

Figura 12: Estación Multi-Robots jugando al ajedrez

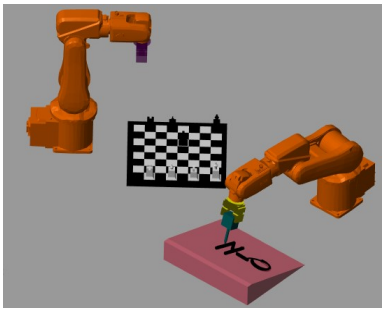

Figura 13: Simulación estación Multi-Robots jugando al ajedrez

Para conseguir que el robot vaya a una posición necesitamos realimentar el sistema e introducir 6 controladores, uno por eje. Los controladores son variables que desde Matlab se van a definir como controladores proporcionales o proporcionales-derivativos.

Los tres cubos de la estacuón tiene peso muy diferentes, para ver los efecto dinámicos de tomar uno de los cubos.

La estación devuelve el par de cada eje, y la posición de los mismos para poder comprobar el efecto del cambio de controladores y el efecto del cambio en tomar uno u otro cubo con distinto peso. La figura 14 muestra el inicio de la simulación.

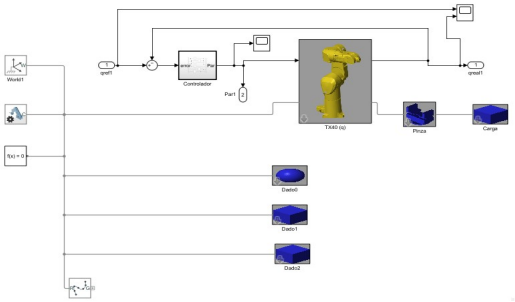

Figura 14: Estación para control de robot

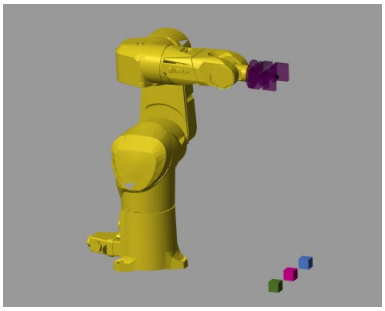

Figura 15: Simulación de estación para control de robot

6 Ejemplo de estación con comunicación OPC

La figura 16 muestra un ejemplo de comunicación con un robot real. Se tiene un robot ABB irb120 con pinza y lápiz. Las posiciones del robot son mandadas a un servidor OPC de ABB [12] con los iconos de la aplicación *OPC ToolBox* de Matlab [13]. Esto implica que debemos tener un servidor ABB activo que detecte el robot y en el icono tenemos que definir las variables persistentes de dicho robot que queramos modificar [11]. Todo ello se realiza con un periodo de T_s s.

El robot de ABB debe tener un programa que mueva el robot a las posiciones cambiadas definidas por las variables persistentes cada T_s s.

El experimento no se ha hecho con el robot real, si no con un robot de ABB simulado en RobotStudio [11]. La figura 17 muestra el final de la simulación en Simulink y RobotStudio.

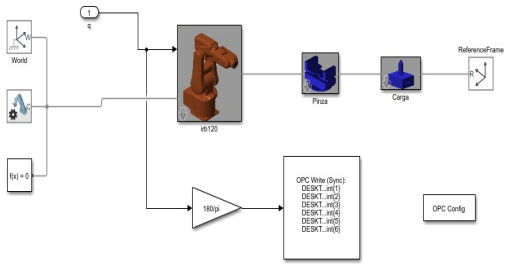

Figura 16: Estación para comunicación OPC

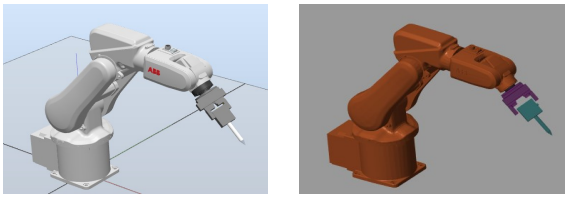

Figura 17: Simulación estación para comunicación OPC

https://doi.org/10.17979/spudc.9788497498043.625

7 Conclusiones y lineas futuras

Se ha descrito una aplicación que mezcla la aplicación de Simulink *SimScape Multi-Body* con la de Matlab *Robotic System Toolbox* con el fin de conseguir una herramienta didáctica en el estudio de la robótica.

La aplicación de Simulink permite diseñar una estación a partir de iconos de robot, de herramientas y objetos de trabajo. La aplicación de Matlab consigue definir las trayectorias deseadas para luego ser simuladas en Simulink. Para ello se ha realizado dos objetos, *Hmat* y *Kin*. Se ha diseñado varios ejemplos donde se demuestra la utilidad de la aplicación.

Esta aplicación puede ser usada para mover robot reales por medio de comunicación OPC, o ROS si se tuviera los robots adecuados.

English summary

MATLAB AND SIMULINK APPLICATION FOR ROBOTIC STATIONS

Abstract

A Matlab / Simulink (it SimRob) tool for didactic use has been designed to design and simulate robotic stations. This library has been made using the Matlab tool it Robotic System Toolbox, the Simulink tool it SimScape Multi-Body, as well as other free Matlab software packages. Most Robotics-related Matlab libraries do not use the Simulink environment in station design and simulations. The proposed library tries to use Simulink resources both for the design of the station components and for the simulation of the same. The control of the simulation, basically its inverse kinematics, has been developed with the Matlab libraries. This is intended for the student to have a pleasant environment to design and simulate the station, and some tools in Matlab to control it. Simulation results can be exported to real robots using different types of communication, for now, OPC communication.

Keywords: Matlab, Simulink, Robotics, Simulation, OPC Communication.

Referencias

[1] Corke, P., (2014) Robotics Toolbox. http://petercorke.com/Robotics Toolbox.htm.

[2] Corke, P., (2013) Robotics, Vision and Control, Fundamental Algorithms in MATLAB. Springer Tract in Avance Robotic.

[3] Gil Aparicio, A. (2014) ARTE (A Robotic Toolbox for Education) Universidad Miguel Hernández (Elche, España)

[4] Robotic System Toolbox (versión 2021a). Software Mathworks.

[5] Robotics System Toolbox. User Guide. (versión 2021a).

[6] ROS (Robot Operating System), http://wiki.ros.org/es.

[7] Simscape Multibody (versión 2021a). Software Mathworks.

[8] Niku, S.. An introduction to robotics: analysis, control, applications. John Wiley Sons (2nd ed.).

[9] Barrientos A., Peñin L.F., Balaguer C. y Aracil P. (2007) Fundamentos de Robótica (2º edición), MacGraw-Hill.

[10] Sugihara, T. Solvability-Unconcerned Inverse Kinematics by the Levenberg–Marquardt Method. IEEE Transactions on . Vol. 27, No. 5 (2011): 984–91. doi:10.1109/tro.2011.2148230.

[11] ROBOTSTUDIO (versión 2020). ABB robotic. Manual del operador RobotStudio 6.01. ID de documento: 3HAC032104-005 Revision:K.

[12] OPC SERVER ABB. ABB robotic. Manual del operador: IRC5 OPC Server help. ID de documento: 3HAC023113-001 Revision: 9.9.

[13] OPC Toolbox (versión 2021a). Software Mathworks.

https://doi.org/10.17979/spudc.9788497498043.625