Brazo UR3e para clasificar piezas con RoboDK

Eva María González Álvarez

Resumen

Este trabajo utiliza la herramienta RoboDK para simular y controlar un brazo robot UR3e. El objetivo es que el robot clasifique unas piezas, cogiéndolas de la mesa y llevándolas a una de las nueve zonas predeterminadas de una estantería. Nos centraremos en que el robot sea capaz de moverse desde la superficie de la mesa a cada una de esas nueve posiciones.

1. Introducción

El sistema que se va a simular es un robot UR3e de Universal Robots. Este robot tendrá la tarea de coger piezas de una mesa y colocarlas en su lugar correspondiente, uno de los nueve espacios disponibles de una estantería. El robot, las piezas y la estantería se situarán encima de una mesa de trabajo, ver Fig. 1.

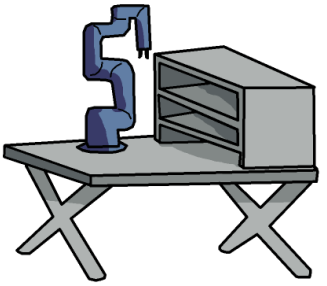

Fig. 1. Esquema de la simulación mostrando el brazo robots, la mesa y la estantería con sus 9 espacios, 3 por cada fila.

Para la simulación del robot usaremos el programa RoboDK, que nos permitirá controlar el robot tanto en simulación como en la realidad si nos hemos conectado previamente.

La simulación deberá tener en cuenta el tipo de pinza utilizada para agarrar las piezas, así como, el espacio de trabajo del robot. No tiene sentido mandar al robot dejar una pieza en una casilla a la que no puede llegar. Utilizaremos el mapa de colisiones de RoboDK para comprobar que el robot no se mueve chocando consigo mismo o con la estantería. Aunque la simulación permita el movimiento atravesando objetos, esto no tiene sentido en la realidad.

En los siguientes apartados exploraremos más a fondo los detalles de este trabajo, empezando por el programa utilizado y siguiendo con las decisiones tomadas para el desarrollo de la aplicación, así como conclusiones obtenidas tras su finalización.

2. RoboDK

Vamos a explorar en este apartado qué es el programa RoboDK que hemos utilizado, así como explicar y dar las razones por las que hemos decidido utilizar esta herramienta y no otra.

Según su página, RoboDK es un simulador de robots industriales capaz de programar también el robot real. Es decir, no solo podemos simular un brazo robot en un entorno virtual, sino que también podemos generar el programa para los drivers del robot real.

RoboDK posee una amplia librería de brazos robot, más de 600 según indican. Tienen diferentes modelos como robots de ABB, KUKA, Universal Robots, etc. Esto va bien con la ventaja que presenta RoboDK, nos permite programar cualquier controlador de cualquier robot directamente desde nuestro ordenador. Con RoboDK podríamos estar simulando y controlando un robot de Universal Robots y uno de ABB desde el mismo programa y con el mismo lenguaje sin falta de aprender el programa específico para cada uno de ellos.

Nuestro robot en concreto, el UR3e, no tiene un programa propio para controlarlo, no como ABB y su RobotStudio. El brazo UR3e se puede programar desde su tableta de control directamente y de una manera muy intuitiva pero que no permite mucha libertad.

Al usar RoboDK hemos podido controlar el UR3e desde un programa de Python, utilizando las librerías propias del programa que al final nos ayudan a traducir las poses que queremos al lenguaje que quieren los controladores de cada robot.

3. Desarrollo del programa

En este apartado vamos a hablar de cómo hemos desarrollado el programa final, así como explicar cómo funciona y qué es lo que hace el mismo.

Lo primero que hemos hecho ha sido buscar todos los archivos 3D que necesitamos para representar lo que queremos simular. Para nuestro caso esto incluye el modelo del robot UR3e, una mesa, una estantería con los espacios entre baldas similar a la estantería real y un gripper. Podemos ver cómo queda todo cuando lo situamos en el entorno de simulación en la Fig. 2.

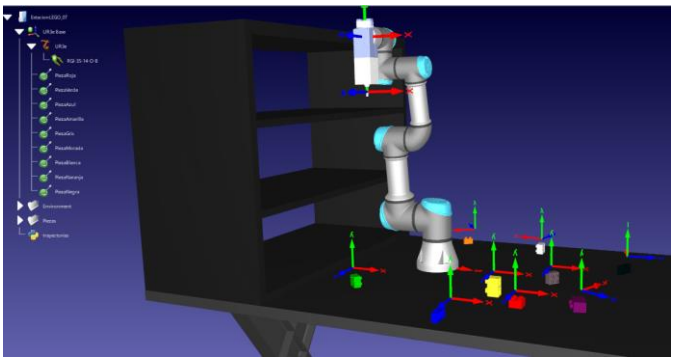

Fig. 2. Captura de la pantalla de simulación. En el lado izquierdo se puede ver el árbol de elementos del programa. Los elementos de la simulación visibles son la mesa de trabajo, estantería, UR3e y 9 piezas de colores situadas por la mesa simulando cómo podrían estar en la mesa.

El archivo del gripper que usamos aparece en la simulación al revés. Aparece con las pinzas donde la muñeca y el cable en el extremo libre. Sin embargo, como esto sólo es una simulación, no vamos a poder controlar el gripper haciendo que se habrá o cierre, de esta manera, nos es indiferente que aparezca dado la vuelta. Vamos a tomar el cable como si fuera la pinza del gripper (ver Fig. 3 para un zoom sobre el modelo 3D del gripper).

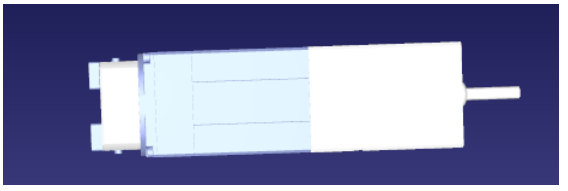

Fig. 3. Gripper en la simulación. En el extremo derecho tenemos el cable y en el izquierdo la parte con la pinza.

Para programar el robot podemos crear un programa dentro del propio RoboDK que se parece mucho a cómo se programan los Universal Robots desde sus terminales, una manera muy intuitiva donde declaramos targets y realizamos llamadas a movimientos (MoveJ, MoveL, …). Un ejemplo de cómo quedaría el programa en el árbol de elementos se puede ver en Fig. 4.

Fig. 4. Ejemplo de un programa hecho con targets en RoboDK. Este programa se mueve hasta la posición definida como “pos1_MasBajo”, luego se mueve linealmente hasta “pos2_IrAPieza”, seguido se mueve hasta la “pos3” y por último hasta “pos0_Vertical” donde termina el programa. Una instrucción no se ejecutará hasta que haya finalizado la anterior. Los targets están guardados en la jerarquía debajo del robot.

En nuestro caso, nosotros vamos a utilizar una programación en lenguaje Python. RoboDK también nos permite esta programación de alto nivel en otros lenguajes como C++ y a través de MATLAB. Desde MATLAB lo que deberíamos hacer es conectar no sólo el robot, sino RoboDK al entorno de MATLAB para poder trabajar desde él.

Para programar en Python tendremos que añadir el programa, el script de Python como un elemento al árbol de elementos como podemos ver en Fig.5. Para ejecutarlo es tan sencillo como dar doble clic al nombre en el árbol y ver cómo se ejecuta la simulación, o clic derecho “Ejecutar en Robot” si tenemos el robot conectado para cargar el programa en el controlador.

Hemos de tener en cuenta que, aunque veamos la posibilidad de “Editar el script”, este no nos abrirá el archivo fuente que le hemos cargado de fuera, sino un archivo temporal. Mientras lo tengamos abierto, cada cambio que hagamos al script y guardemos, se guardará para su ejecución. Si cerrásemos el editor y lo volvamos a abrir, es posible que nos cree un segundo archivo temporal y los cambios previamente realizados no estén presente, pues el archivo temporal parte del que esté en el árbol de elementos que no hemos modificado.

Para solucionar esto deberíamos de prestar atención a qué archivo es el que ha abierto el editor de código para saber con cuál estamos trabajando actualmente.

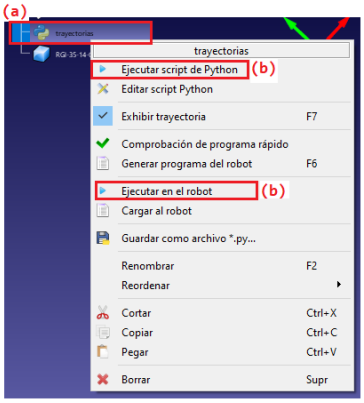

Fig.5. (a) Un programa de Python para ejecutar la simulación. (b) Opciones de ejecución del programa, primero sólo en simulación y segundo en el robot real que debería estar conectado previamente a darle a esta opción.

3.1. Programa de Python

En esta parte explicaremos un poco qué hemos hecho en el programa de Python que hemos creado para la simulación. Todo es programación normal en Python, salvo por el uso de las funciones especiales de las librerías de RoboDK.

Para la resolución hemos decidido simular cómo el robot se acerca a las piezas apoyadas en la mesa y las lleva a su lugar. Cada una de las 9 piezas piezas es llevada a un compartimento distinto de la estantería. Entendemos por "llevada" que el robot hace el movimiento de llevar la pieza hasta el sitio, pero no hay animación de la pinza moviéndose.

El ejercicio se ha resuelto con una combinación de objetivos creados en la simulación y creados con código. Estos primeros se han generado moviendo el robot en el propio entorno de simulación y capturando la posición del actuador final. Estos targets en una ampliación del trabajo serían el resultado de detectar las piezas sobre la mesa.

Los otros, se han realizado con una aplicación de cinemática directa. Le hemos ido dando valores de ángulo a cada una de las articulaciones hasta llegar a la configuración final de la pose que queremos en el actuador final.

Durante la ejecución se puede apreciar que el actuador pasa ligeramente a través de la estantería. Esto se lleva a que nosotros estamos programando trayectorias donde le damos el punto de inicio y el final. La función encargada de generar la trayectoria no sabe que hay una posible colisión y por sí misma no puede lidiar con ella.

Si activásemos el mapa de colisiones que posee RoboDK, veríamos que, al chocar el robot con la estantería, inmediatamente detiene la simulación. No obstante, tras observarse unas colisiones leves (no está atravesando el techo de la estantería), se ha decidió mantener las colisiones apagadas e ignoradas.

4. Conclusiones

Podemos decir que nuestra simulación funciona pues puede ir desde cualquiera de las 9 piezas hasta las 9 posiciones de la estantería.

Hemos visto una limitación no sólo con este robot sino también con el gripper que utiliza. La combinación de ambos hace que el espacio de trabajo del robot sea pequeño. Esto hace que debamos colocar el robot cerca de la estantería y las piezas cerca del robot para que sea capaz de llegar a todos los sitios a los que necesita llegar.

El robot tampoco lo podemos colocar donde queramos fijándonos exclusivamente en su área de trabajo, área que RoboDK puede simular y visualizar. Si ponemos el robot demasiado cerca de la estantería, para las posiciones más bajas de la misma, el robot acabaría chocando consigo mismo y no tendría una manera real de alcanzar esa posición.

Nuestra simulación como mencionamos presenta unas ligeras colisiones con la estantería por parte del gripper. Si garantizamos que en la realidad estas colisiones no se darían, podríamos ejecutar sin problema el programa en el robot real. Hemos de destacar que antes de probarlo en el entorno real, deberemos comprobar que todas las distancias definidas en la simulación pueden ser alcanzables con el robot real y no empezaría a chocar con toda la estantería por incompatibilidad de tamaños entre simulación y realidad.