https://doi.org/10.17979/spudc.9788497498081.0272

Desarrollo de una aplicación para la operación local del robot manipulador POWERBALL: Powerbsim

J. A. Martínez Navarro, J. C. Moreno, J. L. Guzmán
Departamento de Informática, Universidad de Almería, Ctra. Sacramento s/n 04120 Almería
(martineznavarrojoseangel@gmail.com, jcmoreno@ual.es, joseluis.guzman@ual.es)

Resumen

En este trabajo se describe el desarrollo de una aplicación con fines docentes para la simulación y la operación del robot manipulador Powerball. Dicha aplicación ha sido desarrollada en Python utilizando el framework de robótica ROS y el simulador Gazebo.

Palabras clave: ROS, simulación, robot manipulador, Powerball.

1 INTRODUCCIÓN

En robótica, los simuladores son herramientas que ofrecen un entorno virtual en el que emular tanto el comportamiento de los robots como el del resto de elementos típicos de un sistema robotizado.

El objeto de este trabajo es proporcionar al robot Powerball un entorno desde el que se puedan realizar tareas básicas como, por ejemplo, moverlo en coordenadas cartesianas o articulares a partir de una interfaz gráfica, y desde el que se puedan crear programas que a partir de un juego de instrucciones típico para este tipo de robots permitan realizar las tareas tradicionales de manipulación. Esta aplicación se pretende utilizar en las prácticas de las asignaturas que traten con robótica de manipulación.

La aplicación tendrá dos modos de funcionamiento, modo off-line, sin estar conectada al robot real, y modo on-line, conectada al robot. En el primer caso los alumnos podrán trabajar con el simulador para implementar sus algoritmos y podrán comprobar el correcto funcionamiento antes de ejecutar el programa sobre el robot real. Una vez comprobado el correcto funcionamiento de su desarrollo, el alumno cargará el programa desarrollado en el mismo simulador pero del equipo conectado al robot real, que estará trabajando en modo on-line y que permitirá observar los resultados en el mundo real.

Existen numerosos entornos y aplicaciones destinadas al control y a la simulación de robots. Estas se han ido tomando como referencia en el desarrollo de este trabajo, a la hora de diseñar una interfaz amigable, sencilla y fácil de usar.

En algunos casos, las propias compañías que construyen el robot desarrollan también una aplicación para trabajar con él, como Intelitek [9], con el robot Scorbot y el software Robocell. O ABB con RobotStudio [1], aunque se trate en este caso de robots industriales. Otro software conocido es RoboLab [13], para los robots de Lego claramente concebidos para el ámbito docente.

Hay alternativas más genéricas al nivel de las anteriores, una de ellas es Webots [22], que puede simular desde simples vehículos hasta robots humanoides con gran realismo. También Microsoft tiene un entorno de trabajo para robots: Microsoft Robot Studio [11].

También Matlab puede utilizarse como simulador, aprovechando sus herramientas para crear interfaces y sus posibilidades para el cálculo [20]. Basta citar como ejemplo el toolbox para robótica de Peter Corke [6].

La mayoría de programas nombrados hasta ahora presentan algunas desventajas, principalmente que son de código cerrado, algo que limita sus funciones y los limita si se quisieran usar para la investigación o para la docencia en los niveles superiores, principalmente en los títulos de máster.

En el ámbito universitario también se han desarrollado trabajos referentes a interfaces gráficas para el manejo de robots, ya sea para un robot real [12] o incluyendo también la simulación [2] [8].

A continuación se describen los principales materiales y el software utilizado para el trabajo realizado en este artíulo, así como una explicación de la herramienta final desarrollada.

2 MATERIALES Y MÉTODOS

En este proyecto se ha utilizado el robot Powerball (figura 1) [19] desarrollado por la empresa Schunk. Se trata de un robot manipulador de seis grados de libertad destinado a tareas de investigación y docencia.

Tanto la cinemática directa como la inversa se han desarrollado en anteriores trabajos [3] y también se han implementado en Python [5].

La comunicación del robot con el computador se realiza mediante el protocolo CANOpen [4]. Para

el control y envío de órdenes al robot se utiliza ROS [14], un sistema operativo de código abierto concebido para ser usado con robots. Permite la reutilización del código y la abstracción del hardware, por lo que pone al alcance del programador una gran cantidad de paquetes y herramientas con las que trabajar.

Para la simulación con el robot se utiliza Gazebo [7], un simulador 3D independiente de ROS pero integrado en este, también de código abierto. Permite la inclusión de múltiples robots y tiene una API que permite extender sus funcionalidades mediante plugins.

Tanto Schunk como la comunidad proporcionan los paquetes de ROS necesarios para establecer la comunicación con el robot [10] y para simular tanto su aspecto como su comportamiento en Gazebo [17] [18].

Figura 1: Robot Schunk Powerball

La interfaz del simulador se ha implementado usando el lenguaje de programación Python [16], utilizando el *binding* PyQt [15] de la librería gráfica Qt.

3 HERRAMIENTA DESARROLLADA

La interfaz de la aplicación desarrollada, inspirada en la de Robocell [9], consta de una ventana principal (figura 7, ventana superior) desde la cual se pueden mostrar el resto de ventanas que contienen las funcionalidades que componen el trabajo. Para activar la conexión con el robot y que este responda a las órdenes es necesario pulsar el botón *ON/OFF* (derecha) habiendo seleccionado previamente el tipo de robot con el que se está trabajando (real o simulación). El resto de botones muestran las ventanas descritas a continuación.

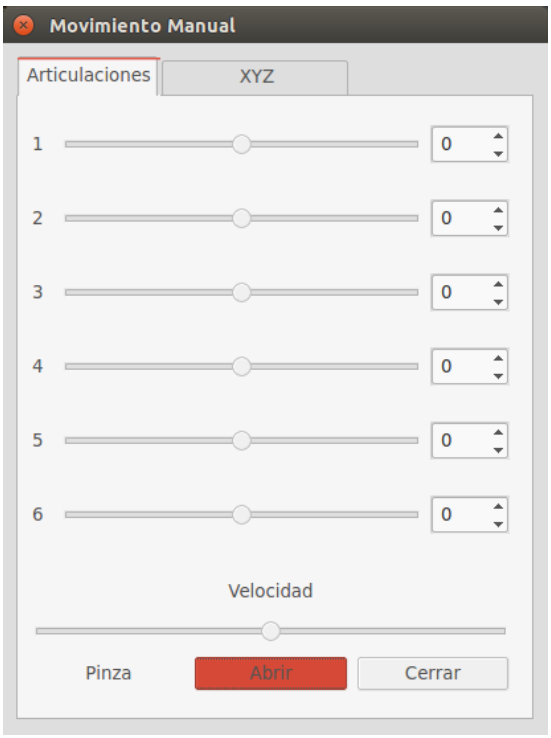

Figura 2: Movimiento Manual de Articulaciones

Figura 3: Movimiento Manual Cartesiano

3.1 Movimiento Manual

La ventana para el movimiento manual (figuras 2 y 3) tiene dos pestañas que por un lado posibilitan el movimiento directo de las articulaciones y por otro el movimiento cartesiano del extremo del robot.

La primera opción (figura 2) permite mover cada articulación de forma independiente, ya sea a través del slider, con el botón (más/menos) o introduciendo por teclado el valor deseado (en grados). Existe uno por cada articulación. En la parte inferior hay otro slider, en este caso para seleccionar la velocidad en un rango de 0 a 100 (de estar parado a la máxima del robot). Además, se incluyen dos botones para tener la opción de abrir y cerrar la pinza.

En la pestaña de movimiento cartesiano (figura 3) se dispone de botones para mover el extremo del

Tabla 1: Movimientos Mando *DualShock*

Articulación	+	-
1	L2	R2
2	↑	↓
3	L1	R1
4	←	→
5	△	×
6	□	○
Velocidad	Trigger Derecha	Trigger Izquierda

robot a través de los ejes X, Y y Z. El valor del desplazamiento en cada pulsación lo determina el valor del paso, incluido abajo. También existe la posibilidad de modificar su orientación (definida en *pitch*, *roll* y *yaw*). De la misma forma que en la pestaña anterior, puede seleccionarse la velocidad y abrir y cerrar la pinza. En este caso también se añade la opción de realizar estas acciones desde el teclado, pulsando la tecla indicada en cada botón.

Para mover las articulaciones también se puede utilizar un mando o joystick. La aplicación se ha preparado para que se pueda realizar el control del brazo con un mando tipo *DualShock* (tabla 1), pero puede utilizarse cualquiera que funcione por USB. Cada par de botones mueve una articulación en ambos sentidos, y con otros dos se puede aumentar o disminuir la velocidad.

3.2 Guardar Posiciones

En la ventana de la figura 4 se da la opción de guardar posiciones. Se puede guardar directamente la posición en la que se encuentra el robot en ese instante, y también se puede seleccionar una posición guardada previamente para trabajar con ella o borrarla. Además es posible llevar al robot a dicha posición.

La ventana se expande dando la posibilidad de introducir un valor de posición (X, Y, Z) y orientación (*pitch*, *roll*, *yaw*). Puede ser una posición absoluta o relativa a otra ya guardada. También se pueden mostrar los datos de cada número de posición previamente introducida.

Todas las posiciones guardadas pueden verse en la ventana de la figura 6, donde aparece una tabla con:

- El número de posición.
- Si la posición es absoluta o relativa.
- La posición de referencia, en el caso de que sea relativa.
- Los valores de posición: X, Y, Z.
- Los valores de orientación: *pitch*, *roll*, *yaw*.

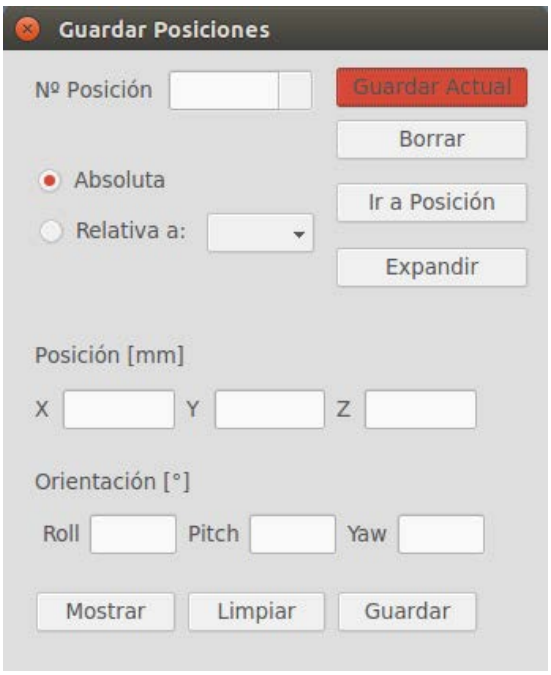

Figura 4: Guardar Posiciones

3.3 Objetos y sensores en Gazebo

Con la ventana de la figura 5 se pueden introducir objetos y sensores en la célula robotizada. En el caso de los objetos es necesario indicar la localización (posición y orientación) donde se van a ubicar, así como sus dimensiones y color. Se pueden introducir cajas, cilindros y esferas.

Figura 5: Introducir objetos

En cuanto a sensores, se han incluido sensores de contacto, de presencia y cámaras, gracias a la variedad de plugins disponibles en Gazebo.

Con dichos plugins se ha desarrollado un alimentador de piezas. A este alimentador hay que indicarle, además de lo anterior, el número de piezas

que se quiere generar y el tipo. Cuando se introduzca en el simulador se mostrará una pieza, y una vez esta se haya cogido, aparecerá otra, así hasta que llegue al límite indicado.

3.4 Lista de comandos

En la lista de comandos (figura 7, ventana derecha) están todos los comandos que se pueden introducir en el programa (figura 7, ventana izquierda).

Respecto a la interacción con el robot, se pueden dar las siguiente ordenes:

- Abrir pinza.
- Cerrar pinza.
- Ir a una posición libremente.
- Ir según trayectoria de articulaciones.
- Ir según trayectoria cartesiana.
- Ir linealmente a una posición.
- Ir circularmente a una posición final pasando por una intermedia.
- Memorizar la posición actual.

En el caso de *Ir según trayectoria de articulaciones*, lo que se introduce como argumento es un archivo que contiene una lista de articulaciones asociadas a un tiempo. Por ejemplo:

```
0.1 0.2 0.3 0.4 0.5 0.6 1
0.2 0.3 0.4 0.5 0.6 0.7 3
0.5 0.6 0.3 1.5 1.1 0.6 5
```

Cada fila representa una posición (seis valores de articulaciones) a la que se debe llegar en un tiempo determinado (último valor de la fila). Al ejecutar esta orden el robot ejecuta cada línea como si fuese una orden *Ir a una posicion*.

Con *Ir según trayectoria cartesiana* ocurre lo mismo pero lo que se introducen son la posición (X, Y, Z) y la orientación (*pitch*, *roll*, *yaw*). Por ejemplo:

```
350 350 650 0 50 90 1
400 -330 660 90 20 0 3
-350 300 650 40 30 50 5
```

Esto supone una ventaja desde el punto de vista docente, pues estas instrucciones permiten generar trayectorias desde programas como Matlab y obtener archivos que se pueden manejar directamente con el simulador, dejando al alumno la posibilidad de implementar todo lo relacionado con el control de trayectorias (ensayando trayectorias lineales, splines cúbicos,...).

Referente a los desplazamientos, en todos estos casos se indicará bien el tiempo en que se debe completar la acción, o bien la velocidad (en un rango de 1 a 10) que se desea que tome el robot.

Asimismo hay órdenes referentes al flujo del programa que permiten:

- Esperar X segundos.
- Poner comentarios.
- Introducir y modificar variables.
- Añadir etiquetas.
- Saltar a etiquetas. En este caso, el salto puede ser directo o realizarse en función de una sentencia *if*, donde se evalúa el valor de una variable o el estado de un sensor.

También se pueden crear subrutinas. En éstas es posible introducir la órdenes descritas anteriormente. Las subrutinas se pueden utilizar para invocarlas directamente o para hacerlo automáticamente como consecuencia de la ocurrencia de interrupciones.

Para las interrupciones, se introduce una orden para habilitarla o deshabilitarla. A cada interrupción se le asocia un sensor y cuando este se activa, se ejecuta la subrutina asociada. Tras la ejecución, el sistema vuelve al punto donde estaba antes de ocurrir la interrupción y continúa con la ejecución del programa.

Todas las órdenes necesitan parámetros, así que cuando se quiera introducir una aparecerá otra ventana donde se deberán introducir los que correspondan (número de orden, velocidad, tiempo de espera, etc).

Por último, está la interfaz de la lista del programa (figura 7, ventana izquierda), donde irán apareciendo las instrucciones que se van introduciendo en el programa. Desde esta ventana se pueden abrir archivos guardados previamente, o guardar el programa y posiciones con las que se esté trabajando.

Los archivos guardados contienen el tipo de orden con los valores necesarios y las posiciones, y se guardan con un formato determinado como un archivo de texto, por lo que se pueden modificar manualmente. El contenido de un programa de ejemplo:

```
ABRIR_PINZA
IR 1 tiempo 4
ESPERAR 1
IR 2 tiempo 3
CERRAR_PINZA
IR 1 tiempo 1
IR 3 velocidad 5
IR 4 velocidad 3
ABRIR_PINZA
IR 2 tiempo 2
```

Tabla de posiciones

	Nº	Absoluta	Relativa	A:	X	Y	Z	YAW	PITCH	ROLL
1	1	X			293.13	303.15	226.53	-89.00	-0.67	179.23
2	2	X			293.20	303.23	93.26	-89.00	-0.68	179.22
3	3	X			-578.47	283.11	131.38	19.03	-2.64	-174.31
4	4	X			-401.34	292.39	209.28	-91.01	-0.62	-179.17
5	5		X	1	100.00	-20.00	0.00	0.00	0.00	0.00
6	6		X	3	100.00	0.00	20.00	0.00	0.00	0.00
7										

Figura 6: Tabla de posiciones

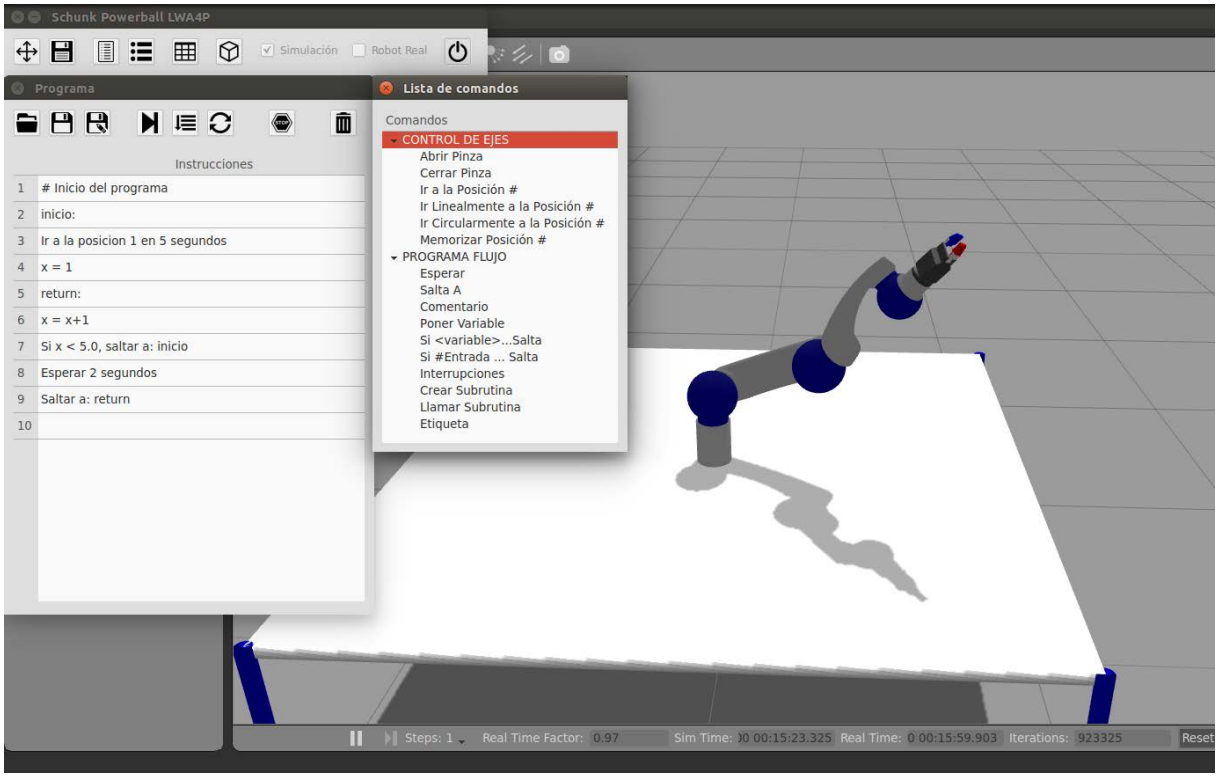

Figura 7: Simulador Gazebo con la aplicación Powerbsim

```
POSICION 1 1 0 297 305 254 -90 0 180
POSICION 2 1 0 297 305 86 -90 0 180
POSICION 3 1 0 -400 400 145 -90 0 180
POSICION 4 1 0 -400 400 80 -90 0 180
```

Para ejecutar las órdenes que se hayan introducido hay tres opciones:

- Ejecutarlas una a una.
- Ejecutar un ciclo completo.
- Ejecutarlas todas de forma continua. En este caso la única forma de pararlo sería pulsando el botón *Stop*.

Respecto a la edición de las órdenes introducidas, también es posible borrarlas, modificar sus parámetros e introducir nuevas en cualquier línea, para facilitar la creación de los programas.

El botón de *Stop* para al robot y deja de ejecutar el ciclo en el caso de que haya comenzado.

En el siguiente video se puede ver el simulador ejecutandose: [21]

4 CONCLUSIONES

Se ha presentado una aplicación para controlar y trabajar con el robot Powerball utilizando las herramientas que proporciona ROS, Gazebo y Python.

Con la herramienta desarrollada se puede manejar el robot de una forma sencilla, desde la interfaz, el teclado o un mando. También se pueden crear programas para que el robot realice tareas en las que puede interactuar con sensores.

Es posible desarrollar estos programas con el simulador, sin tener el robot real cerca, y después pasarlos a este sin ninguna modificación, lo cual favorece su uso en un entorno docente.

Que tanto el proyecto como las herramientas utilizadas sean de código abierto permite desarrollar nuevas funcionalidades en el futuro y mejorar las actuales.

Referencias

[1] ABB RobotStudio (Consultado: 1/06/2016): http://new.abb.com/products/robotics/es/robotstudio

[2] Beltrán Blanco, M., Feliu Batlle, J., Cano Izquierdo, J. M., (2004) Simurob. Simulador del robot IRB-1400

[3] Bradley, C., (2014) Robotic Arm Calibration and Control 6-DOF Powerball LWA 4P.

[4] CANOpen (Consultado: 2/06/2016): http://doc.ingeniamc.com/emcl2/command-reference-manual/communications/canopen-protocol

[5] Cinemática en Python (Consultado: 2/06/2016): http://www.cs.rpi.edu/foswiki/bin/view/RoboticsWeb/PowerballSchunk

[6] Corke, P. I., (2011) Robotics, Vision & Control: Fundamental Algorithms in Matlab, Springer.

[7] Gazebo (Consultado: 31/06/2016): http://gazebosim.org/

[8] Gómez Rubio, A., (2014) Control de la mano robótica BarrettHand BH8-262 en entorno ROS (Trabajo fin de grado). Universidad de Alcalá, España.

[9] Intelitek (Consultado: 1/06/2016): http://www.intelitek.com/robots/robotics-and-automation&-technology-competition/

[10] IPA CANOpen ROS (Consultado: 31/06/2016): http://wiki.ros.org/ipa_canopen_ros

[11] Microsoft Robot Studio (Consultado 1/06/2016): https://www.microsoft.com/en-us/download/details.aspx?id=29081

[12] Navío Haro, L., (2013) Diseño y desarrollo del interfaz para la teleoperación de un brazo robótico (Proyecto fin de carrera). Universidad Carlos III de Madrid, España.

[13] Robolab Lego (Consultado: 2/06/2016): http://www.legoengineering.com/robolab-2-9-4c-patch/

[14] ROS (Consultado: 31/05/2016): http://www.ros.org/

[15] PyQt (Consultado: 2/06/2016): https://sourceforge.net/projects/pyqt/

[16] Python (Consultado: 2/06/2016): https://www.python.org/

[17] Schunk Description (Consultado: 31/06/2016): http://wiki.ros.org/schunk_description

[18] Schunk Modular Robotics (Consultado: 31/06/2016): https://github.com/ipa320/schunk_modular_robotics

[19] Schunk Powerball (Consultado: 31/05/2016): http://mobile.schunk-microsite.com/en/produkte/products/powerball-lightweight-arm-lwa-4p.html

[20] Simulador Robot ABB Matlab (Consultado: 1/06/2016): https://www.youtube.com/watch?v=fa7GwwA3498

[21] Simulador Schunk Powerball en Youtube (Consultado: 13/06/2016): https://youtu.be/Prbh__ylieI

[22] Webots (Consultado: 1/06/2016): https://www.cyberbotics.com/