Práctica No.5. Robot Operating System - ROS

1st Juan Carlos Garduño Gutiérrez 157302
Ingeniería en Telecomunicaciones
Instituto Tecnológico Autónomo de México
jgarduo6@itam.mx

2nd Humberto Martínez Barrón y Robles 166056
Ingeniería en Mecatrónica
Instituto Tecnológico Autónomo de México
hmartin3@itam.mx

3rd Sebastián Aranda Bassegoda 157465
Ingeniería en Computación e Ingeniería Industrial
Instituto Tecnológico Autónomo de México
sarandab@itam.mx

***Resumen*—Se reporta el trabajo realizado en esta práctica que consiste en aprender a utilizar las bases de ROS y conectarlo con Arduino. Para la introducción de ROS se utiliza el paquete $turtlesim$ que resultó ser una herramienta muy buena para aprender lo básico de ROS; para la conexión con Arduino se proponen algunas aplicaciones prácticas en las que se involucra la relación Arduino-ROS y las soluciones se implementan utilizando la librería de $roserial$ para intercomunicarse con el Arduino y para utilizar ROS desde Arduino, se necesita importa $ros.h$ así como otras librerías que utiliza el programa.**

***Index Terms*—$Arduino$,ROS,$turtlesim$,$roserial$,$ros.h$**

I. Introducción

El aprendizaje de robótica ha ido avanzando últimamente hacia el uso de sistemas operativos ya implementados para robótica que sean flexibles, escalables, intuitivos y gratis. Una de estas opciones es ROS. La robótica en general, tanto para aplicaciones industriales como académicas, ha tendido cada vez más hacia el uso de este sistema operativo en particular. La razón a la que se debe esto no sólo son las mencionadas anteriormente (excepto por lo intuitivo), sino que, también, ROS cuenta con amplia documentación, múltiples tutoriales y soporte para varios lenguajes de programación. También vale la pena mencionar que, aunque ROS tiene múltiples ventajas, también tiene una serie de problemas que no se han logrado resolver porque forman parte de la misma flexibilidad que promete ROS. Dos de ellos - los más importantes - son: (1) es difícil de aprender para principiantes y (2) dado que existe un problema de control de versiones, suele ser un mayor problema descargar los paquetes y verificar que las versiones sean compatibles que entender el algoritmo que se está implementando.

Con todo, ROS es una solución satisfactoria para el problema que deseaba resolver originalmente: cada vez que se armaba un robot, había que programar desde cero el sistema operativo que controlaría las operaciones que cada componente debía llevar a cabo para garantizar su funcionamiento, y estas soluciones solían depender del $hardware$ con el que se trabajaba. Por lo tanto, la razón por la que este aprendizaje resulta interesante y de gran importancia es que este sistema se utiliza actualmente en una amplia gama de aplicaciones comerciales, que están impulsando enormemente la investigación en el área. Entonces, se genera un círculo virtuoso: las aplicaciones comerciales incitan la investigación, la cual logra descubrimientos que se pueden comercializar. Mientras este proceso continúa, la herramienta base - ROS, por ahora - se sigue consolidando. Entonces, para una futura carrera en robótica, hoy resulta crucial estar familiarizado al menos en un nivel básico con ROS.

Esta práctica sirve como una muy breve introducción a esta herramienta: se revisan los requerimientos básicos para correr uno de estos programas, se corre una simulación basada en un tutorial ya implementado, se aplica el equivalente a esto en $hardware$ (un microcontrolador) y, finalmente, se utiliza la herramienta para controlar un paquete desde otra computadora.

II. Marco teórico

En esta sección se describen los dispositivos utilizados, otras tecnologías y conceptos aplicados. Es decir, muestra los fundamentos teóricos y matemáticos del porqué las cosas funcionan, así como las suposiciones y las restricciones.

Se utiliza $Arduino$ y, como ya se ha explicado lo que es y en qué consiste en prácticas anteriores, se omite esta explicación. También se utiliza ROS $(RobotOperatingSystem)$, que es una herramienta que proporciona bibliotecas para ayudar a los desarrolladores de software a crear aplicaciones de robótica. Proporciona abstracción de $hardware$, controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, administración de paquetes y más. ROS es completamente de código abierto (BSD) y es gratuito para que otros lo usen, lo modifiquen y lo comercialicen. [1]

El objetivo principal es permitir a los desarrolladores de $software$ crear aplicaciones para robot más capaces de forma rápida y sencilla en una plataforma común.

Particularmente, se utiliza una herramienta de ïntroducción"llamada: $turtlesim$. $Turtlesim$ es una forma sencilla de aprender lo básico de ROS desde un paquete ya probado e implementado. La simulación consiste en una ventana gráfica que muestra un robot con forma de tortuga. El color de fondo para el mundo de la tortuga se puede cambiar utilizando el Servidor de parámetros. La tortuga se puede mover en la pantalla mediante comandos ROS o usando el teclado.

A continuación se muestra una imagen de cómo se ve la herramienta introductoria:

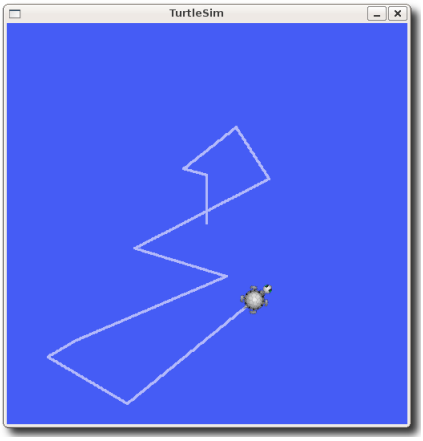

Figura 1. Tortuga que aparece al correr *Turtlesim*

Con toda la teoría presentada en la práctica proporcionada por el departamento, suponemos debería ser suficiente para que los programas salgan bien en esta práctica, contamos con las limitaciones que se hablaron en la introducción: es difícil de aprender para principiantes y el control de versiones puede representar un problema para el programador desprevenido.

III. Desarrollo

III-A. Parte 1: ROS

Para esta parte se necesitaba implementar dos nodos: un suscriptor y un publicador que se comunicarían entre ellos. Cuando el primero recibe un mensaje del segundo, debía imprimir una respuesta en la terminal.

Para resolver esta parte, simplemente fue necesario descargar el repositorio donde se había proveído el código de inicio necesario y modificar las partes relevantes para lograr el comportamiento deseado. Dentro del repositorio, se encontró que el paquete que se debía ejecutar se llamaba `ejemplo`. Entonces, los nodos que se podían correr eran: `ejemplo_pub_node` y `ejemplot_sub_node`. Esto significa que los archivos a modificar eran: `ejemplo_pub.cpp` para el caso del publicador y `ejemplo_sub.cpp` para el caso del suscriptor. Las modificaciones necesarias fueron los tipos de mensajes que se estaban publicando y recibiendo. Mientras que originalmente éstos eran de tipo $Int32$, se necesitaba que fueran de tipo $String$. Entonces, además de cambiar el tipo de mensaje, fue necesario incluir la biblioteca de este tipo de mensajes en las primeras líneas de ambos documentos. También el tipo de publicador tuvo que ser cambiado (todos los aspectos de cada nodo estaban configurados para recibir un mensaje de tipo $Int32$) junto con la función de $callback$ del suscriptor.

Después, fue necesario compilar el paquete (con el comando `catkin_make`) para después correr ambos nodos del paquete. Para esto, en una ventana de la terminal se corrió el comando `roscore`. En otra, se usó el comando `rosrun ejemplo ejemplo_pub_node` mientras que, en una ventana más de la terminal, se corrió el comando `rosrun ejemplo_sub_node`.

III-B. Parte 2: Turtlesim

Para la segunda parte del laboratorio, se pidió utilizar el paquete *Turtlesim* - un tutorial de ROS - para mover una tortuga utilizando los números:

- Si se presiona la tecla correspondiente al número 8, la tortuga avanza hacia adelante.
- Al presionar la tecla que corresponde al número 4, la tortuga gira sobre su propio eje hacia la izquierda.
- Cuando se oprime la tecla que muestra el número 6, la tortuga gira sobre su propio eje hacia la derecha.
- En cuanto es oprimida la tecla del número 2, la tortuga se mueve hacia atrás.

Estos objetivos se pudieron lograr sin mucho esfuerzo: resulta que, dentro de los paquetes de tutoriales de ROS, ya se incluyen los documentos necesarios para correr el nodo de *Turtlesim*. El comando para correrlo es (con `roscore` corriendo en otra terminal): `rosrun turtlesim turtlesim_node`. Al correr este comando, la tortuga que se muestra en la figura 1 aparecerá en el simulador, con un fondo de color.

En cuanto al control de la tortuga, ya existe también un nodo dentro de este repositorio (el repositorio es `ros_tutorials/turtlesim`) que permite controlar la tortuga con comandos del teclado. El nombre del nodo es `teleop_turtle_key_node` y su código está en el archivo `teleop_turtle_key.cpp` ubicado en la carpeta `tutorials`. Al analizar el código de este programa, se encontró que está originalmente programado para que el control de la tortuga se haga por medio de las flechas del teclado. Para cambiar esto para que sean los números los que mueven a la tortuga en vez de las flechas, simplemente había que reemplazar los códigos ASCII de las teclas definidas en la primer parte del documento como `KEYCODE_R`, `KEYCODE_L`, `KEYCODE_U` y `KEYCODE_D`. Más abajo en el código, queda claro que estas teclas corresponden a los comandos de *right*, *left*, *up* y *down*. Entonces, se reemplazó el código ASCII presente en el código original con el de las teclas deseadas para lograr el desempeño requerido.

III-C. Parte 3: ROS y Arduino

Esta parte requería que, al enviar un comando desde un nodo que estaba corriendo en la computadora, se encienda un LED conectado al microcontrolador Arduino durante tres segundos. Esto significaba que había que realizar y correr un solo nodo publicador y suscriptor en el microcontrolador y otro en la computadora a la que éste estaba conectado.

Antes que nada, fue necesario seguir las instrucciones proporcionadas en la práctica para instalar las dependencias de ROS para correrlas en el microcontrolador. Se corrió una serie de comandos que construyen las bibliotecas requeridas para utilizar ROS en un Arduino. También fue necesario descargar un repositorio de comunicaciones seriales de ROS.

Una vez terminada la instalación, se prosiguió a escribir el código para resolver el problema.

Una complicación con esta parte de la práctica fue que existe un cambio de sintaxis no trivial entre ROS para C++ y ROS para Arduino. Comandos como `ros::NodeHandle.initNode()` existen para Arduino pero no para C++, y los comandos de este tipo más bien se llevan a cabo desde ROS (por ejemplo, en vez de escribir `nh.initNode()`, en C++ se escribió `ros::init(argc,argv,"NOMBRE DEL NODO")`). Además de esta complicación, también se encontró que la versión de `serial_node` que se tenía estaba diseñada para Python 3, pero la computadora que se utilizó estaba utilizando Python 2.7. La única repercusión de esto fue que un parámetro en particular cambiaba de nombre: para instanciar un objeto `Serial`, se utilizaba el parámetro `write_timeout`, pero esto generaba un error (mencionaba que este parámetro no existía). Por lo tanto, después de buscar en internet preguntas acerca de problemas similares, se encontró que el nombre de este parámetro había cambiado en una actualización de `writeTimeout`. Cuando se puso el nombre original del parámetro en el código original, el error desapareció y se logró prender el LED.

III-D. Parte 4: ROS en múltiples computadoras

Para esta última parte de la práctica, se pedía correr Turtlesim en una computadora y controlar el nodo desde otra.

La solución consistió simplemente en exportar las variables de ambiente necesarias (`ROS_HOSTNAME` y `ROS_MASTER_URI`) y después correr *Turtlesim* en una computadora y *teleop_turtle_key* en otra, que ya habían sido implementados. En este caso, la computadora en la que se corrió *roscore* tenía una dirección IP que terminaba en 94, y en la que se ejecutó el nodo para controlar la tortuga tenía una direción IP que terminaba en 93. Por lo tanto, mientras que en la primera la variable de ambiente `ROS_HOSTNAME` fue la dirección IP que terminaba con 94 y en la segunda fue la dirección IP que terminaba en 93, en la segunda se exportó la variable de ambiente `ROS_MASTER_URI` con el valor `http://[PRINCIPIO DE IP].93:11311/`

Después, se corrió en la primer computadora en comando `roscore` y, en otra terminal, el comando `rosrun turtlesim turtlesim_node`. En la segunda computadora se corrió (después de haber exportado las variables de ambiente descritas en el párrafo anterior) el comando `rosrun turtlesim teleop_turtle_key_node`. Así, se podía ver cómo desde una computadora se oprimían teclas que lograban que la tortuga se mueva en la dirección especificada.

IV. Resultados

IV-A. Primera parte de la práctica. ROS

La primera parte de la práctica fue relativamente simple y se pudo completar sin mayores contratiempos. Se implementaron dos nodos exitosamente de manera que se comunicaran entre ellos, el primero logrando recibir la señal del segundo.

El primer nodo, el publicador, emitió una señal de tipo String que fue recibida por el segundo nodo, el suscriptor. Esto se hizo a través de tres ventanas diferentes, donde en cada una se ejecutaba un nodo y en la tercera el comando `roscore`.

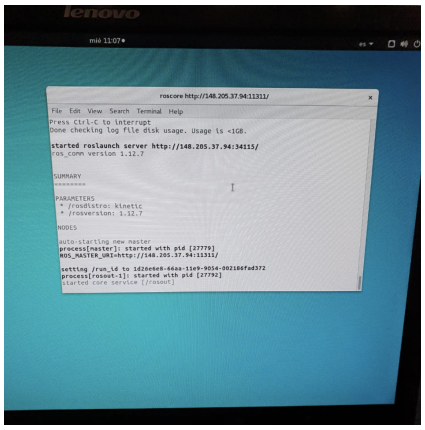

Figura 2. Computadora corriendo los comandos.

IV-B. Segunda parte de la práctica. `Turtlesim`

En la segunda parte se alacnazaron los obejtivos de manera muy sencilla. Se utilizó el paquete `Turtlesim` para mover una tortuga, ilustrada en la figura 1, utilizando números en lugar de las flechas del teclado.

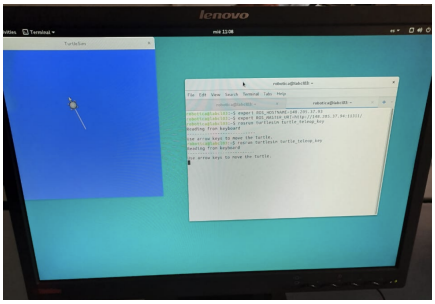

Figura 3. Computadora corriendo `Turtlesim` con las modificaciones.

Para conseguirlo se modificaron los códigos ASCII para lograr que la tortuga se desempeñara con las teclas deseadas.

IV-C. Tercera parte de la práctica. ROS y Arduino

De manera parecida a la primera parte, en la parte 3 se utilizaban publicadores y sucriptores, en este caso para que un LED, conectado al microcontrolador Arduino, se encendiera.

Fue posible alcanzar los objetivos, y prender el LED, pero no sin antes enfrentarnos a las comunicaciones seriales de ROS en las cuales existían diferencias considerables entre su implementación en C++ y Arduino.

IV-D. Cuarta parte de la práctica. ROS en múltiples computadoras

Tras realizar las partes anteriores de la práctica fue muy sencilla implementar la cuarta. La solución La solución consistió

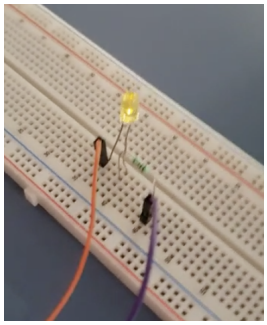

Figura 4. Circuito mostrando el LED

simplemente en exportar las variables de ambiente necesarias, (`ROS_HOSTNAME` y `ROS_MASTER_URI`), y después correr los nodos necesarios previamente implementados.

V. Conclusiones

V-A. *Humberto Martínez*

Como conclusión, podemos decir que, aunque ROS no es exactamente un sistema operativo, ni tampoco es exclusivo para robots, su uso en el ámbito de la robótica se ha vuelto cada vez más común últimamente, y, por lo tanto, su conocimiento se está volviendo más valioso con el tiempo para aquellos que piensan dedicarse a aplicaciones de robótica con fines académicos o comerciales.

La complejidad de ROS puede llevarse al extremo, y, aunque hay ocasiones en las que los programas que se están utilizando resultan muy simples y muy fáciles de comprender, la escalabilidad de ROS no garantiza que éste siempre sea el caso. Por lo tanto, vale la pena cuidar que las herramientas que se utilizan estén bien organizadas y que la documentación esté lo más completa posible, dado que las repercusiones de la desorganización al acomodar los paquetes puede ser catastrófica.

Los objetivos de la práctica indica claramente que se busca aprender a utilizar ROS, pero de ninguna manera parece que el objetivo sea volverse experto en ROS en una sola práctica. Sin embargo, se podría considerar que el objetivo fue logrado, porque el equipo logró familiarizarse con ROS en un nivel suficiente para utilizarlo en algunas aplicaciones de robótica y para integrar su uso con $hardware$.

V-B. *Juan Carlos Garduño*

De primera mano,sentí que esta práctica estaba más enfocada a mis compañeros que estudian Ingeniería Mecatrónica; sin embargo, con el desarrollo y la evolución de la práctica me di cuenta que en realidad no es así y la importancia que tiene la práctica.

Investigando y trabajando con ROS empecé a pensar la buena combinación que resultaría implementar sistemas que utilicen ROS y sistemas de telecomunicaciones. Un ejemplo muy burdo: se puede generar una arquitectura de comunicación para robots de rescate, en donde el acceso a humanos es casi imposible. Y en este ejemplo ROS jugaría un papel muy importante, ya que,este permite la comunicación basada en mensajes, la incorporación de nuevos subscriptores a la red y la transmisión en tiempo real. Entonces, ahora que la práctica finalizó, pienso que la práctica no esta sólo enfocada a los estudiantes de Ingeniería en Mecatrónica, sino también a los estudiantes de Ingeniería en Telecomunicaciones.

Aunque no se estudia ROS en la clase de teoría, esta práctica sirve para darnos una pequeña pero buena introducción a ROS. Investigando lo que es ROS y qué involucra me doy cuenta del poder que tiene y las múltiples aplicaciones que tiene en el mundo comercial, industrial y académico.

También, me siento satisfecho con el trabajo de mi equipo porque creo que nos hemos complementado bastante bien y nuestros horarios se asemejan para poder trabajar fuera de horas de clase.

V-C. *Sebastián Aranda*

Me quedé satisfecho con la práctica ya que me pareció útil para aprender sobre ROS, que no es algo que hayamos realmente cubierto durante la clase de teoría.

Es interesante observar que se ha vuelto mucho más difícil la elaboración de las prácticas, algo que no parece que se le atribuya necesariamente a la teoría detrás de la práctica como sí parece ser culpa de la implementación de ellos. En las últimas ocasiones la diferencia entre la teoría y la práctica se ha incrementado y la dificultad añadida ha provocado que tengamos que trabajar fuera del horario de clase, haciendo toda el proceso del laboratorio todavía más tedioso.

A final de cuentas creo que es útil el laboratorio porque tiene muchos aspectos que se asemejan más al mundo laboral que nos espera como ingenieros y la entrega de proyectos y elaboración práctica de los mismos.

VI. Rol o Papel

VI-A. *Humberto Martínez*

En la primera sección de la práctica, Humberto tomó posesión del teclado para realizar el código necesario en la práctica. En lo que los demás compañeras trataban de resolver los problemas que el equipo venía arrastrando con los $op-amps$ de la práctica pasada (*Motor DC & Control*).

En las demás secciones, se encargó del código, mientras sus compañeros colaboraban con dudas de código que iban surgiendo.

En cuanto al reporte Humberto elaboró o colaboró las siguientes secciones:

- Introducción.
- Desarrollo.
- Sus respectivas conclusiones.

VI-B. *Juan Carlos Garduño*

En la práctica, Juan Carlos se encargó junto con Sebastián a ir alambrando los componentes necesarios. En las primera parte de la práctica, iba dictando algunas secciones del código a Humberto que había investigado y podían ser de utilidad para la compilación del programa. De igual manera, iba revisando

que la sintaxis de Humberto en el código, fuera la correcta. En cuanto al reporte Juan Carlos elaboró las siguientes secciones:

- Marco teórico.
- Resumen.
- Sus respectivas conclusiones.

VI-C. Sebastián Aranda

En la práctica, Sebastián se encargo del alambrado junto con Juan Carlos, así como de ir colaborando con los demás integrantes en la elaboración del código. En cuanto al reporte él elaboro las secciones siguientes:

- Resultados.
- Rol o Papel.
- Sus respectivas conclusiones.

REFERENCIAS

[1] Willow Garage. (2011). ROS. abril 24,2019, de Willow Garage Sitio web: http://www.willowgarage.com/pages/software/ros-platform

[2] Cplusplus. (2019). C++. abril 24,2019, de cplusplus Sitio web: http://www.cplusplus.com/

[3] DIE. (2019). Linux man pages. abril 24,2019, de die.net Sitio web: https://linux.die.net/man/

[4] Gary. (2018). ROS Cheat Sheet. abril 24,2019, de Departamento Académico de Sistemas Digitales, ITAM Sitio web: https://github.com/garygra/ROS_cheat_sheet/blob/master/main.pdf

[5] ROS.Official Page (2019). abril 20,2019, de ROS Sitio web: http://www.ros.org/

[6] ROS Wiki. (2019). turtlesim. abril 24,2019, de ROS Sitio web: http://wiki.ros.org/turtlesim