Práctica 5
Principios de Mecatrónica

Jorge Alejandro Ramírez Gallardo
Ingeniería Mecatrónica e Ingeniería Industrial
Instituto Tecnológico Autónomo de México
Ciudad de México
Email: ramirez-alejandro.g@outlook.com

Ricardo Edward Meadowcroft
Ingeniería en Computación y Licenciatura en Matemáticas
Instituto Tecnológico Autónomo de México
Ciudad de México
Email: rmeadowc@itam.mx

***Abstract*—En la presente práctica se realizará una introducción a la creación de paquetes, información y comunicaciones entre nodos y arranque de ROS y al uso de TURTLESIM como herramientas para la ejecución de sistemas en constante comunicación. Adicionalmente, se explicarán herramientas como rqt_plot y rqt_graph para obtener información de un nodo y su comportamiento.**

INTRODUCCIÓN

La descripción y desarrollo de tecnologías de comunicaciones ha permitido que diferentes sistemas puedan interactuar entre sí a través de la información que reciben o que envían. En ese sentido, dentro de las tecnologías más conocidas para la comunicación encontramos a ROS: una nube de comunicación entre sistemas cuya base radica en la identificación de los diferentes puertos de entrada y de salida como nodos.

En el presente documento ahondaremos en la implementación de algunas herramientas en ROS dentro de una máquina virtual con un sistema operativo Ubuntu Xenial (y algunos otros pasos iniciales) que permita al lector utilizar algunas funcionalidades que posee ROS.

MARCO TEÓRICO

A. Sobre los nodos de ROS

La arquitectura de ROS está basada principalmente en la interacción entre **nodos**, siendo estos procesos que llevan a cabo ciertas computaciones con un nombre que permite referencia a él dentro del sistema de ROS. [1]

Para un sistema complejo como un robot, ROS esta hecho para que cada nodo lleve a cabo una tarea determinada relacionado con un elemento específico de su funcionalidad, por ejemplo, uno que maneje la visión computacional, otro que maneje el sistema de control que mueve los motores, otro que determine el camino que debe llevar a cabo el robot, etc.

Para llevar a cabo la interacción entre tales nodos, una herramienta importante existente en ROS es la de los **tópicos**. En estos, nodos que producen información de una utilidad especifica publican tal información en un **mensaje**, que contiene una lista de estructuras de datos nombrados, al un tópico nombrado que sirve tal tema. Nodos que deben usar información de tal tema en su computación se pueden entonces subscribir al mismo tópico para recibir todos los mensajes enviados, sin tener que importarle cuales son los nodos específicos de dónde se origina, de dónde puede haber entonces varios publicadores y suscriptores siempre que trabajen sobre mensajes que llevan un mismo formato.[1]

El modelo anterior de tópicos, siendo unidireccional desde los publicadores hacia los subscriptores, permite realizar una variedad de comunicaciones útiles, pero no es apropiado para comunicar entre nodos donde uno debe primero solicitar la información al otro antes de que este realice la acción para mandar un mensaje en respuesta. Para esto, se pueden usar **servicios**, que permite definir un par de mensajes que funcionen como solicitud y respuesta.

La estructura de ROS permite entonces una gran extensibilidad en la funcionalidad de los nodos, y un conjunto de nodos con las relaciones de tópicos y servicios que lo conectan, formando una estructura de grafo, junto con su configuración y datos, etc., puede unirse dentro de un **paquete**. Así, se puede reunir a un conjunto de acciones que llevan a cabo una funcionalidades relacionadas de manera modular, siendo el paquete el elemento más pequeño que se puede construir para su ejecución. [1]

B. Gráficas informativas: rqt_graph y rqt_plot

Para poder visualizar la estructura que tiene un sistema en ROS, que es un grafo con los nodos, y las aristas dirigidas entre ellas que son los tópicos que los relacionan, ROS tiene disponible la funcionalidad en un plugin gráfico llamado rqt_graph. En este, los nodos son representados como círculos nombrados que son entonces conectadas por flechas, que representan como los tópicos por flechas que van desde los publicadores a subscriptores.

Por otro lado, si se quiere visualizar el funcionamiento que tiene tal sistema, una de las formas de hacerlo para, por ejemplo, cuando se quiere saber el movimiento que simula un sistema en coordenadas de espacio, es usar rqt_plot. Aquí, si al comando se le dan como argumentos las estructuras de datos numéricas que están en uno o varios tópicos, mostrará una gráfica de linea que demuestra cómo el valor de un dato cambia conforme varía el valor de otro.

DESARROLLO

Para el desarrollo de la práctica correspondiente se requiere una computadora con sistema operativo Ubuntu Xenial (16.04

LTS) y con ROS Kinetic Kame instalado. La primera sección ahondará en la implementación de los conocimientos teóricos sobre la creación y el comportamiento de los nodos en ROS. La segunda sección abordará el uso de TURTLESIM como herramienta para las simulación dentro de la robótica y se explorarán algunos comandos y funcionalidades visibles a través de este.

C. Configuración del paquete en ROS

Para poder iniciar la comunicación dentro de ROS se debe de crear un paquete. Cada paquete permite la ejecución de ciertas tareas de comunicación o de implementación de código. Para ello, cada paquete debe de tener su propio directorio, además de dos archivos: uno que provea meta información del paquete y sus dependencias (con extensión .xml) y otro que contenga la entrada al sistema de compilación CMake para crear paquetes de software (CMakeLists.txt).

Para iniciar, utilizamos el comando **roscore** en una nueva terminal para iniciar ROS. En otra terminal,se debe de refrescar el acceso a la configuración del entorno con el comando **source /opt/ros/kinetic/setup.bash**. La carpeta del paquete debe estar contenida en nuestro directorio de trabajo correspondiente (en nuestro caso, catkin_ws, subdirectorio src).

Creamos el paquete con el comando **catkin_create _pkg nombreDelPaquete std_msgs rospy roscpp** (lo último establece las relaciones de dependencia necesarias para que el paquete incorpore lo necesario para correr programas en Python o C). Con el comando **cd /catkin_ws** podemos dirigirnos a nuestro directorio de trabajo y escribir **catkin_make** para construir los paquetes en el área de trabajo.

Nuestro paquete contendrá la carpeta **src**. Dentro de esta carpeta se colocan los códigos ejemplo para publicar y suscribirse y los documentos mencionados con anterioridad se colocan dentro de la carpeta del paquete (en nuestro caso, **ejemplo**). Utilizamos **catkin_make** para rehacer el paquete y poder así emplearlo con la comunicación entre nodos.[2]

D. Nodos en ROS

Con ROS corriendo, insertamos el comando **rosrun ejemplo ejemplo_pub_node** para inicializar el nodo relacionado con la publicación de información. La terminal mostrará la solicitud de información, como se muestra en la Figura 1.

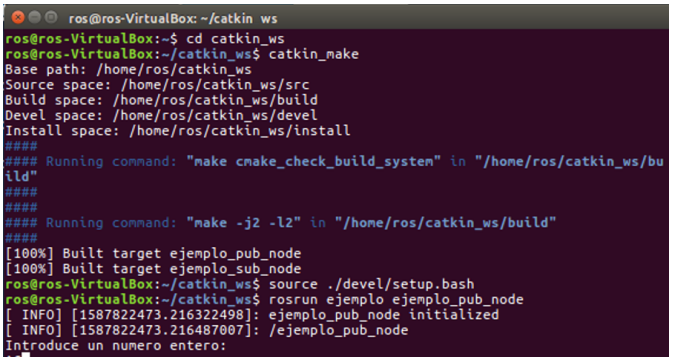

Figura 1. Ejecución del nodo que publica.

El comando **rosrun ejemplo ejemplo_sub_node** en una nueva terminal para inicializar la suscripción de información, como muestra la Figura 2.

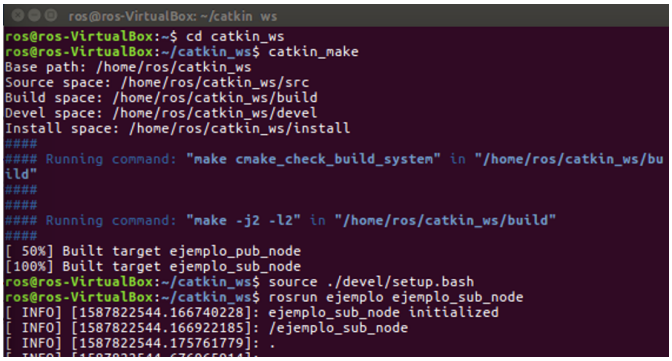

Figura 2. Ejecución del nodo suscriptor.

Note que en ambas ventanas se deben de correr los comandos ejecutados previamente para que puedan correr los archivos correspondientes. Cuando se envía información por medio de la terminal del nodo que publica, el nodo suscriptor recibe la información, como lo muestran las figuras 4 y 5 de la sección de Resultados.

Note que esta implementación puede adaptarse para enviar cualquier tipo de información, incluyendo texto, como lo muestra la Figura 6. Para ello, dentro del código en C++ deben de incorporarse los mensajes de ROS tipo String con el comando **#include <std_msgs/String.h>**. A su vez, dentro del nodo que publica se deben de actualizar el tipo de mensaje a String en las líneas **ros::Publisher pub = nh.advertise<std_msgs::String> ("/msg_ejemplo", 1);** y **std_msgs::String msg;**. También el nodo suscriptor debe ser editado incluyendo el primer comando del párrafo y cambiando el método get_msg por el siguiente: **void chatterCallback(const std_msgs::String::ConstPtr& msg){ ROS_INFO("I heard: [%s]", msg->data.c_str()); }**.

E. TURTLESIM

Figura 3. Ejecución de TURTLESIM

Para continuar con la revisión de algunas herramientas dentro de ROS, ejecutaremos a TURTLESIM. Para ello, se deberá de insertar el comando **rosrun turtlesim turtlesim_node turtle** para ejecutarlo, como se muestra en la Figura 3.

TURTLESIM posee algunos tópicos como *pose* y *cmd_vel*, donde el primero permite visualizar la posición y la orientación, mientras que el segundo tópico contiene la velocidad linear y angular que posee la tortuga.[3] Para visualizar gráficamente los tópicos se actualizan conforme la tortuga realiza movimientos, utilizamos **rqt_graph** y **rqt_plot**.

Considere dos trayectorias: una en línea recta y otra de forma circular. La forma para realizar la trayectoria uno puede efectuarse a través del comando **rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 – '[2.0, 0.0, 0.0]' '[1.0, 0.0, 0.0]**, mientras que, para la trayectoria dos, se emplea el comando **rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 – '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.0]**.

Activamos las gráficas con los comandos **rosrun rqt_graph rqt_graph** y **rosrun rqt_plot rqt_plot** en diferentes terminales. Ambos nos permiten observar la dinámica entre los tópicos, como se explica en la siguiente sección.

Resultados

Los resultados de las secciones C y D se muestran en las siguientes imágenes. Las primeras dos figuras muestran las dos terminales que corrieron los nodos de publicación y de suscripción, respectivamente, comprobando con ello la utilidad de la comunicación entre nodos a través de los tópicos definidos por los scripts en C++, tanto para números como para texto.

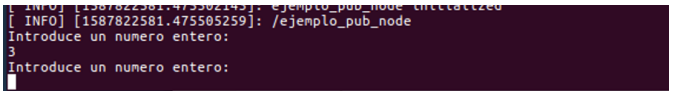

Figura 4. Envío de información.

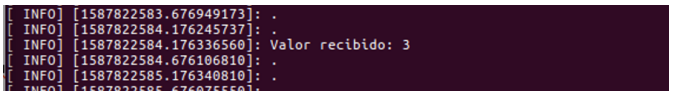

Figura 5. Recepción de información.

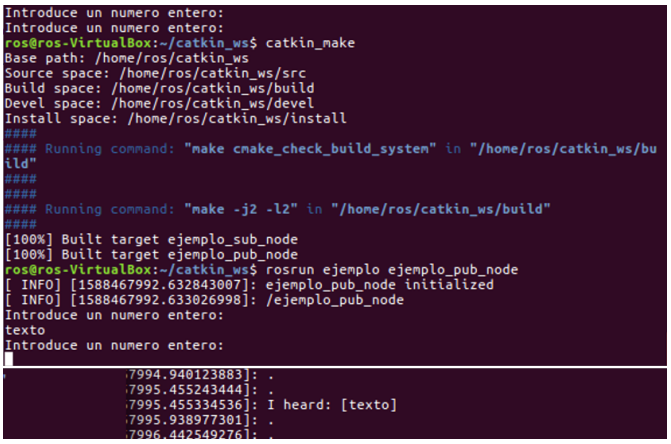

Figura 6. Transmisión del texto.

Por su parte, con relación a la ejecución de TURTLESIM, se incluyen las gráficas de rqt_graph y rqt_plot para cada trayectoria. Observe que, en el caso de la trayectoria lineal, theta y y conservan un valor constante porque la variación sólo se realiza linealmente sobre el eje X hasta que la tortuga se estrella contra la pared de la ventana.

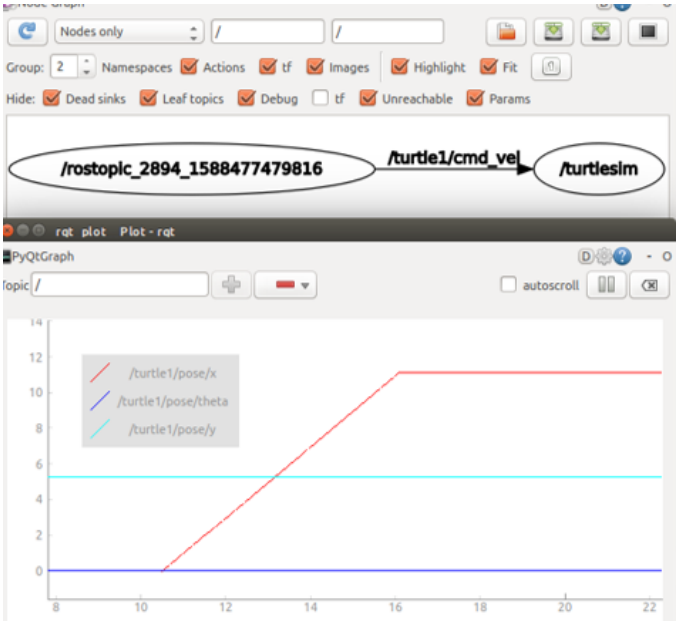

Figura 7. rqt_graph y rqt_plot para trayectoria circular

En contraste, para la trayectoria circular, los valores de x y y corresponden a un comportamiento sinusoidal correspondiente con su posición a lo largo del círculo, mientras que los valores de theta varían linealmente entre 0 y 2pi, como lo muestra la Figura 8.

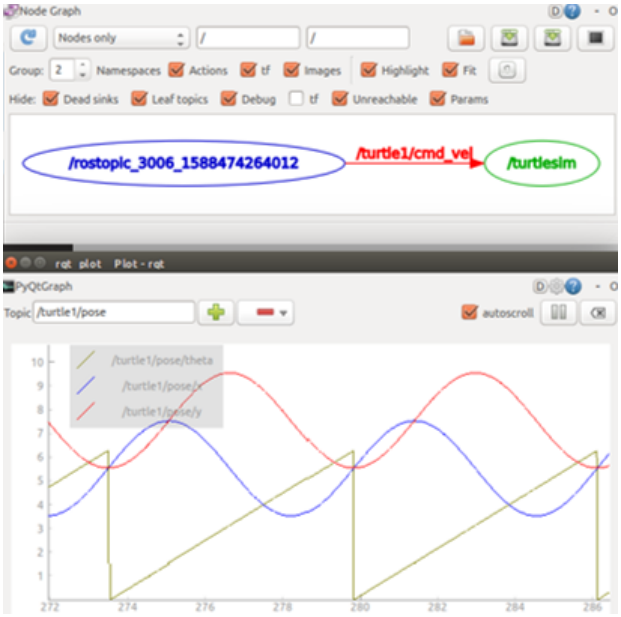

Figura 8. rqt_graph y rqt_plot para trayectoria circular

El último punto a destacar dentro de los resultados es que

ambas figuras contenían dentro de sí la misma imagen para rqt_graph, pues este gráfico se encarga de visualizar en tiempo real la comunicación entre nodos y los tópicos por los cuales se suscriben o publican y, para ambas trayectorias, el movimiento se suscribía a las instrucciones provenientes de un comando.

CONCLUSIONES

Primeramente, el uso de ROS como herramienta para poder interactuar con diferentes componentes y sistemas comunicados a través de los tópicos descritos describe una forma organizada de realizar las comunicaciones entre los nodos.

Adicionalmente, las simplificaciones e introducciones al uso y conocimiento de sistemas que introducen herramientas como TURTLESIM permiten al ávido ingeniero realizar la verificación y prueba de las soluciones a alguna problemática en general que una red de sistemas o elementos comunicándose pueden solventar, teniendo interesantes aplicaciones para el desarrollo de sistemas interactivos que permitan cumplir objetivos medibles con acciones rectificables.

REFERENCES

[1] "Documentation", *ROS Wiki*, 2020, wiki.ros.org

[2] "Writing a Simple Publisher and Subscriber (C++)", *ROS Wiki*, wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29.

[3] "Understanding ROS Topics", *ROS Wiki*, wiki.ros.org/ROS/Tutorials/UnderstandingTopics.