Práctica 5
Robot Operating System

Stephanie Lizeth Malvaes Diaz
Ingeniería en
Instituto Tecnológico Autónomo de México
Email:

Daniela Arely Morales Hernández
Ingeniería en Telecomunicaciones
Instituto Tecnológico Autónomo de México
Email: daniela.morales@itam.mx

***Resumen*—En esta práctica aprendimos cómo utilizar ROS (Robot Operating System) y también cómo conectarlo con Arduino. ROS provee librerías y herramientas para ayudar a los desarrolladores de Software a crear aplicaciones para robots. También repasamos la programación y compilación en C/C++.**

I. Introducción

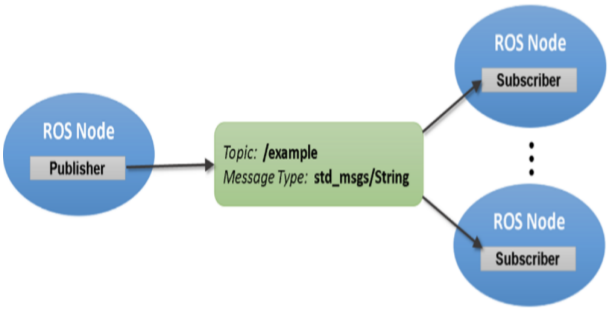
ROS tiene un enfoque distribuido, lo que significa que los nodos que comunica no requieren estar siquiera en la misma computadora. Este aspecto distribuido permite que múltiples dispositivos implementados con tecnologías diversas y con funcionalidades varias logren colaborar. Una de las aplicaciones mas llamativas y obvias de ROS es la coordinación de robots. Tambien se puede utilizar en en sistemas de reconocimiento facial. Con un nodo que publique imagenes (una cámara), otro nodo que ejecute el proceso de analisis y un nodo que consuma los resultados se puede implementar este sistema. Junto con el aspecto modular y distribuido de ROS, los nodos que implementan las diversas tareas pueden no estar dentro de la misma computadora: la camara podría estar posicionada en un lugar fijo y que se encuentre en la misma red de una Raspberry pi que ejecute el proceso de reconocimiento y publique los resultados para que una computadora física los consuma.

II. Marco Teórico

Existen varias maneras de comunicar dos dispositivos y una de ellas es la comunicación indirecta. En este modelo la comunicación entre entidades se realiza a través de un intermediario y sin la necesidad de un acoplamiento directo entre emisores y receptores. De esta manera, el emisor no necesita conocer al receptor, y viceversa. Uno de los modelos de comunicación indirecta es el de publicación-inscripción, en donde la comunicación está basada en eventos. Por un lado los editores, o publishers, publican eventos a un tópico determinado; y por el otro, los subscriptores, o subscribers, se subscriben a los tópicos de su interés y el servicio se encarga de entregarle los eventos que se publiquen a estos tópicos. Además, antes de comenzar a transmitir eventos, los editores anuncian el tópico para así avisar a los subscriptores que este existe. Unas de las características principales de este modelo de comunicación es que la comunicación se realiza de manera asíncrona y que no es necesario que un tópico tenga subscriptores para que un editor pueda publicar eventos al mismo. Es importante mencionar que en este modelo puede haber más de un editor y subscriptor a un tópico.

II-A. ROS

ROS (Robot Operating System), es un meta sistema operativo para robots, es decir, es un intermedio entre un sistema operativo y un middleware. ROS provee los servicios estándar de un sistema operativo como abstracción de hardware, control de dispositivos de bajo nivel, paso de mensajes entre procesos y manejo de paquetes. Pero, también provee funcionalidades de alto nivel como llamadas asíncronas y síncronas, base de datos centralizada, un sistema para congurar robots, entre otras. El objetivo principal de este framework es facilitar la creación de comportamientos de robots que sean complejos y robustos entre diversas plataformas.



ROS está basado en una arquitectura peer-to-peer, acoplada a un sistema de almacenamiento en búffer y un sistema de búsqueda (master), los cuales permiten que cuales quiera dos componentes se comuniquen entre ellos de manera síncrona o asíncrona. La comunicación asíncrona se realiza por medio de un sistema de publicación-inscripción en el que se publican mensajes, que pueden ser de distintos tipos, a diferentes tópicos. Algunos de los conceptos principales de ROS son:

- Computation Graph: Es la red peer-to-peer de procesos ROS, que están trabajando juntos.
- Nodos: Los nodos son procesos que realizan diferentes operaciones computacionales. ROS está diseñado para ser modular, por lo que un sistema se compone de varios nodos.
- Master: Provee un sistema de registro y de búsqueda en la Computation Graph. Esto permite que los nodos

se encuentren entre sí, manden mensajes o invoquen servicios.

- Mensajes:Los nodos se comunican entre sí mediante el paso de mensajes.Un mensaje es una estructura de datos que contiene campos tipados.
- Servidor de parámetros: Permite que los datos se guarden por medio de una llave en una localización central. Es parte del Master.
- Bolsas: Es un formato para guardar y reproducir los mensajes de ROS. Es un mecanismo útil para guardar datos y poder reutilizarlos posteriormente.
- Servicios: Las operaciones tipo request/reply (solicitud/respuesta) son hechas por medio de servicios, denidos por un par de estructuras de mensajes: una para la solicitud y otra para la respuesta.

ROS no está diseñado para un lenguaje en especíco, las conexiones peer-to-peer se realizan en XML-RPC que es un modelo que existe en varios lenguajes. Actualmente existen implementaciones de ROS en Python, C++m Lsip y librerías experimentales en Java y Lua. Además, ROS está diseñado para ser lo más delgado posible, de manera que el código programado para ROS se pueda integrar fácilmente con otros frameworks de software para robots.

II-B. Turtlesim

Una de las herramientas con las que cuenta ROS es Turtlesim. Turtlesim es un simulador que consiste en una ventana gráca que muestra una tortuga-robot. Esta tortuga puede ser controlada mediante mensajes tipo geometry msgs/Twist, los cuales se publican en el tópico /turtleN/cmd vel. En la Fig 2, se puede ver el funcionamiento de un robot-tortuga en Turtlesim. Para esta práctica, casi toda la programación de nodos de ROS se hizo en C++, que es un lenguaje de programación de propósito general basado en C. C++ es bastante popular actualmente; muchos sistemas operativos, drivers, navegadores y juegos están basados en este lenguaje. C++ es un lenguaje de nivel medio, por lo que el código interactúa directamente con el hardware interno de la computadora. Es considerado un lenguaje multi-paradigma, pues soporta diferentes estilos de programación, tales como: orientado a objetos, programación estructurada, ensamblador de alto nivel, entre otros. También se implementó ROS en Arduino, el cual es una plataforma de electrónica open-source. Consiste en una placa física de circuito programable, llamada microcontrolador y un software que se ejecuta en la computadora y que permite escribir y cargar código a la placa física.

Sumamos estos tres valores con la finalidad de ajustar el proceso por medio de un elemento de control. Ajustando estas tres variables en el algoritmo de control del PID, el controlador puede proveer una acción de control diseñado para los requerimientos del proceso en específico. Se pueden usar independientemente una de otra las fases del control. Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control.

III. DESARROLLO

III-A. ROS

Se modicaron los nodos ejemplo pub_node y ejemplo sub_node proporcionados de tal modo que, al recibir un mensaje, el subscriptor respondiera publicando un mensaje del tipo std msgs::String bajo un nuevo tópico. Al recibir la respuesta, el publicador la imprimía en la terminal.

El nodo ejemplo pub node publicó mensajes del tipo std msgs::Int32 bajo el tópico /msg_ejemplo, al cual se subscribió el nodo_ejemlpo_sub_node. Al recibir el mensaje, ejemplo sub_node imprimía el valor recibido y publicaba un mensaje std msgs::String bajo el tópico /msg_response, al cual se subscribió el nodo ejemplo_pub_node. El código del paquete conteniendo los nodos puede ser encontrado bajo la carpeta /src/ejemplo en el repositorio.

III-B. Turtlesim

Se creó un nodo turtle_controller_node para controlar el nodo turtlesim_node de ROS. Dependiendo del caracter recibido, turtle_controller_node mandaba instrucciones de movimiento distintas al Turtlesim. Al recibir el caracter '2', la tortuga giraba en el sentido de las manecillas del reloj, al recibir '4', se movía hacia atrás, al recibir '6', se movía hacia adelante y al recibir '8', giraba en el sentido opuesto a las manecillas del reloj. Además, si se recibía una cadena de caracteres en lugar de un único carácter, el nodo enviaba un comando por cada carácter, ignorando caracteres inválidos. El nodo turtle_controller_node enviaba direcciones al Turtlesim a través de mensajes tipo geometry_msgs::Twist en el tópico /turtle1/cmd vel. Además, se impuso un pequeño retardo entre mensajes para evitar que estos se perdieran en la cola del tópico. El código del paquete conteniendo el nodo controlador puede ser encontrado bajo la carpeta /src/turtle en el repositorio.

III-C. ROS-Arduino

Se creó un programa para el Arduino que, al recibir un mensaje, prendiera un LED por 3 segundos y luego contestara. Luego, este programa se comunicó con ROS utilizando el nodo serial node de la librería rosserial. Por último, se agregó un nodo no arduino, que publicaba mensajes para el Arduino y luego recibía la respuesta.

Para el arduino, se creó un programa que, al recibir mensajes tipo std_msgs::String en el tópico /other_msg, prendía un LED conectado al puerto 13 por 3 segundos y luego respondía con un mensaje tipo std_msgs::String en el topico /arduino msg. Para el publicador, simplemente se modicó el nodo ejemplo pub_node, actualizando los tópicos que utilizaba. Finalmente, se monto un nodo serial_node para la comunicación con el Arduino a través del puerto serial. El código para el Arduino puede ser encontrado bajo la carpeta /arduino_ros, mientras que el del paquete conteniendo el nodo publicador esta bajo la carpeta /src/totalmente_arduino.

IV. Conclusiones

Utilizar ROS facilita la comunicación entre procesos a través del uso de tópicos. Esto permite pasar información entre distintos nodos de manera fácil, haciendo que el diseño del sistema sea mucho mas modular. La transmisión de información entre nodos es fundamental cuando un sistema cuenta con varios componentes. ROS, con sus capacidades de abstracción, permite que los nodos se encuentren física y lógicamente separados y por ende sean distribuidos. Los modelos de comunicación indirecta, como el de publicación-inscripción, son muy útiles al momento de crear sistemas con varios componentes. ROS permite modelar las diferentes acciones/partes de un robot de manera distribuida por medio de nodos y tópicos, lo cual facilita la implementación del sistema.

Referencias

[1] J. O. Gutiérrez García, "Modelos de comunicación indirecta," notas del curso Sistemas Distribuidos, Ene-May 2019.

[2] G. Robots, "Ros – robot operating system," accessed 2019-04-26. [Online]. Available: https://www.generationrobots.com/blog/en/ ros-robot-operating-system-2/