

Reporte de la Práctica 5: ROS

Andrea Marín Alarcón
Luis Felipe Landa Lizarralde
Miguel González Borja



Resumen—La comunicación entre componentes de un robot representa una seria problemática. Cuando un nodo afecta de forma directa la ejecución de las tareas de otro, la transmisión eficaz de mensajes entre dichos nodos es de importancia fundamental. ROS (Robot Operating System) se presenta como la herramienta para salvar estas dificultades. Con esta herramienta logramos crear un programa en donde un suscriptor escucha por mensajes de un publicador; implementamos un simulador en el que se dirige una tortuga por medio del teclado; expandimos este simulador para dirigir a la tortuga desde un teclado en otra computadora y, en conjunto con un Arduino, implementamos un programa que al escuchar un mensaje se prende un LED por un tiempo determinado.

1. INTRODUCCIÓN

Al facilitar la comunicación entre procesos heterogéneos, las aplicaciones posibles se tornan, en esencia, infinitas. Como ROS tiene un enfoque distribuido, los nodos que comunican 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 más llamativas y obvias de ROS es la coordinación de robots. El término coordinar puede adquirir más de un significado en este contexto: coordinar el movimiento de robots para evitar colisiones, coordinar el comportamiento de varios robots para desarrollar una estrategia de equipo, etc. Lo que es particularmente interesante de esta aplicación es que cada robot puede estar utilizando ROS de forma individual, pero el *framework* que provee ROS permite comunicarlos.

Aunque ROS incluya la palabra Robot en su nombre, también se puede utilizar en aplicaciones que no se atribuyen a lo que comúnmente pensamos como robots. Se puede utilizar en sistemas de reconocimiento facial. Con un nodo que publique imágenes (una cámara), otro nodo que ejecute el proceso de análisis 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 cámara 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.

El reporte prosigue con el Marco Teórico en el cual se detallan los aspectos técnicos de las herramientas utilizadas

en la práctica; sigue el Desarrollo, que incluye los pasos y el procedimiento llevado a cabo; el apartado de Resultados exhibe lo que se obtuvo siguiendo el procedimiento descrito en el Desarrollo; las Conclusiones presentan observaciones y análisis por parte de los integrantes del equipo sobre lo sucedido en la práctica; el Rol indica la participación de cada integrante del equipo en la práctica; las Fuentes Consultadas proveen la bibliografía utilizada para llevar a cabo la práctica y elaborar el reporte.

2. 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 [1].

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 suscriptores, 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 suscriptores que éste existe [1]. 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 suscriptores para que un editor pueda publicar eventos a él. Es importante mencionar que en este modelo puede haber más de un editor y suscriptor a un tópico.

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 configurar robots, entre otras [2]. El objetivo principal de este *framework* es facilitar la creación de comportamientos de robots que sean complejos y robustos entre diversas plataformas [3].

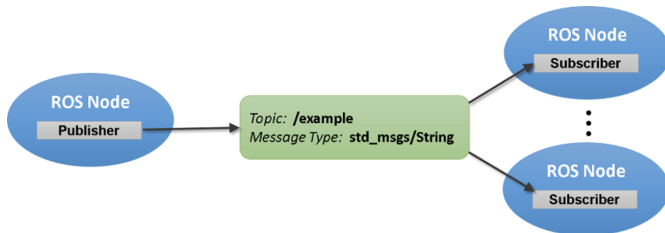


Figura 1: Sistema de publicación-inscripción de ROS

ROS está basado en una arquitectura *peer-to-peer*, acoplada a un sistema de almacenamiento en búfer 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 (Fig. 1)

Algunos de los conceptos principales de ROS son [4]:

- **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, definidos por un par de estructuras de mensajes: una para la solicitud y otra para la respuesta.

ROS no está diseñado para un lenguaje en específico, las conexiones *peer-to-peer* se realizan en XML-RPC que es un modelo que existe en varios lenguajes [2]. Actualmente existen implementaciones de ROS en Python, C++ y Lisp 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.

Una de las herramientas con las que cuenta ROS es *Turtlesim*. *Turtlesim* es un simulador que consiste de una ventana gráfica 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* [5]. 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

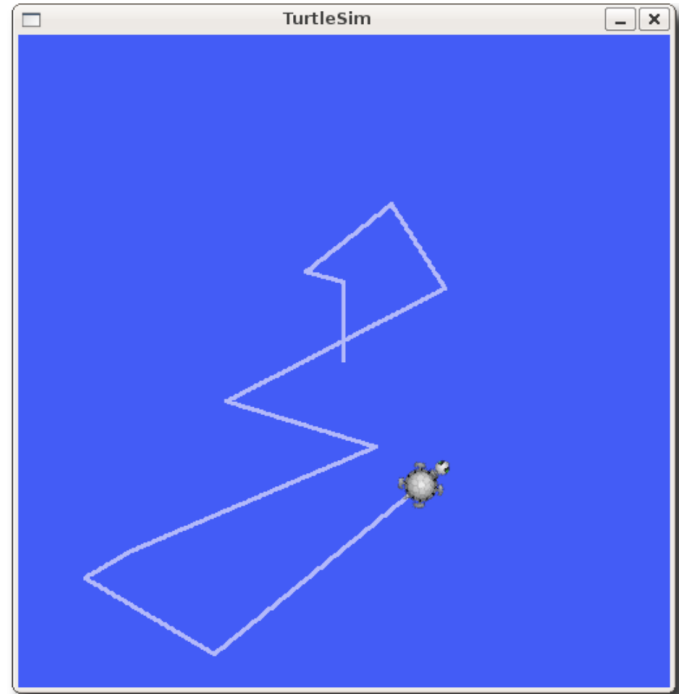


Figura 2: Movimiento del robot.

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 [6] [7].

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 [8].

3. DESARROLLO

3.1. ROS

Se modificaron 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.

3.2. Turtlesim

Se creó un nodo *turtle_controller_node* para controlar el nodo *turtlesim_node* de ROS. Dependiendo del carácter recibido, *turtle_controller_node* mandaba instrucciones de movimiento distintas al *Turtlesim*. Al recibir el carácter '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.

3.3. Arduino-ROS

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.

3.4. ROS en múltiples computadoras

Se monto el *roscore* así como un nodo de *Turtlesim* en una computadora. Luego, desde otra computadora en la misma red, se conecto levanto el nodo *turtle_controller_node*, agregando la dirección IP y puerto en el cual estaba montado el *roscore*. Una vez que se hizo esto, el *Turtlesim* funcionó igual que como funciono en la sección 3.2.

4. RESULTADOS

4.1. ROS

El nodo *ejemplo_pub_node* publicó mensajes del tipo *std_msgs::Int32* bajo el tópico */msg_ejemplo*, al cual se suscribió el nodo *ejemplo_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 suscribió el nodo *ejemplo_pub_node*.

El código del paquete conteniendo los nodos puede ser encontrado bajo la carpeta */src/ejemplo* en el repositorio.

4.2. Turtlesim

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.

4.3. Arduino-ROS

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 tópico */arduino_msg*.

Para el publicador, simplemente se modificó 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*.

4.4. ROS en múltiples computadoras

El *roscore* y el *Turtlesim* se montaron en la máquina con dirección IP 148.205.37.91, con el *roscore* en el puerto 11311. Por otro lado, el controlador se montó en la computadora con IP 148.205.37.92.

5. 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.

Miguel González Borja

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.

Luis Felipe Landa Lizarralde

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.

Andrea Marín Alarcón

6. ROL

Miguel González Borja - Nodos de *Turtlesim*; sección de Desarrollo y Resultados

Luis Felipe Landa Lizarralde - Implementación de programa con ROS y el Arduino; secciones de Abstract e Introducción.

Andrea Marín Alarcón - Implementación del programa ROS-Arduino y ROS en múltiples computadoras; sección de Marco Teórico.

7. FUENTES CONSULTADAS

- [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/>
- [3] O. S. R. Foundation, "About ros," accessed 2019-04-26. [Online]. Available: <http://www.ros.org/about-ros/>
- [4] ROS.org, "Ros concepts," accessed 2019-04-26. [Online]. Available: <http://wiki.ros.org/ROS/Concepts>
- [5] —, "Understanding ros topics," accessed 2019-04-26. [Online]. Available: <http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics>
- [6] S. Downey, "The paradigms of c++," accessed 2019-04-26. [Online]. Available: <https://edn.embarcadero.com/article/10419>
- [7] Programiz, "Learn c++ programming," accessed 2019-04-26. [Online]. Available: <https://www.programiz.com/cpp-programming>
- [8] sparkfun, "What is an arduino?" accessed 2019-04-26. [Online]. Available: <https://learn.sparkfun.com/tutorials/what-is-an-arduino/all>