[[1]](#footnote-1)

Reporte de práctica 5: Robot Operating System

(Abril 29, 2019)

Thomas, Manuel *&* Ruz, Pablo. Principios de Mecatrónica, ITAM

*Abstract*— A lo largo de dos sesiones, se trabajó con ROS bajo distintos esquemas. Se produjeron publicadores y suscriptores para que entre ellos se mandaran mensajes; se crearon otro par de archivos para manejar Turtlesim y finalmente se implementó ROS junto con Arduino.

*Index Terms*—ROS, Arduino, Turtlesim, C, publicador, suscriptor.

# Introducción

Hoy en día, las funciones realizadas con robots son en extremo complejas, dada la cantidad de actividades que estos realizan. Para poder lidiar con todas estas cuestiones, se desarrolló el conjunto MRS (Multi-Robot Systems); una plataforma que permite la implementación de varios robots complejos para desarrollar una tarea conjunta.

ROS es uno de los sistemas de mayor uso y con mayor desarrollo dentro de los MRS. Aunque sus siglas provienen de *Robot Operating System,* se trata en realidad de un meta-sistema operativo, puesto que para su funcionamiento adecuado es necesario montarlo sobre otro sistema operativo basado en UNIX.

El propósito de ROS es facilitar la implementación de cualquier robot, mediante un lenguaje que permite desde la interacción de plataformas hasta la comunicación de procesos. Así mismo, es posible, dada su naturaleza modular, desarrollar paquetes especializados en diversas funciones para alcanzar armar proyectos de importancia alta.

# Marco teórico

ROS es un sistema modular, en donde cada uno de los módulos que lo compone es independiente del otro. Esto implica que tiene que existir comunicación entre ellos para poder llevar a cabo tareas de colaboración y, para esto se utilizan protocolos específicos. Todo esto es posible gracias a los componentes que forman ROS, de los cuales se provee una explicación a continuación.

*Nodo:* como se dijo anteriormente, ROS tiene estructura modular. Un nodo es una especie de módulo que conjunta una serie de funciones que le permiten realizar funciones dentro de sí mismo, pero también comunicarse y colaborar con otros nodos.

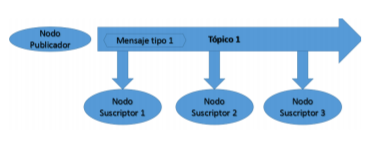


*Fig. 1*

*Estructura básica de un nodo. Obsérvense los diversos componentes que lo forman: publicaciones, suscripciones, prestación y petición de servicios.*

*Topic:* como su nombre lo indica, un “topic” es el nombre que se le otorga a una serie de procesos que permiten identificarlo y, por ende, que exista comunicación entre los nodos. Es decir, cuando un nodo realiza una función determinada, puede publicarlo a un topic y, a su vez, un segundo nodo puede estar suscrito a este topic para así poder recibir el mensaje tras la publicación por parte del nodo uno.

*Mensaje:* un mensaje es el paquete de datos que un nodo publicador envía a través de un tópico para que el suscriptor pueda leerlo.



*Fig. 2*

*Véase que el nodo publicador manda el mensaje a través del tópico uno, de modo que los nodos suscriptores pueden tener acceso a él y, por ende, al mensaje.*

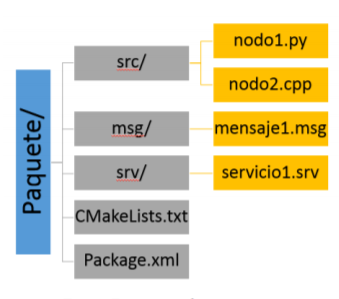
*Servicio:* Algunos nodos dentro de ROS pueden funcionar mediante la estructura cliente-servidor, de modo que uno presta el servicio mientras que todos los demás pueden solicitarlo. Bajo este esquema, el servicio debe de especificar tanto el total como el tipo de datos, según lo muestra la siguiente figura:



*Fig. 3*

*Esquema cliente-servidor. Véase que el nodo proveedor del servicio es el único que lo manda, mientras que los solicitantes realizan una petición y posteriormente reciben una respuesta.*

*Paquete:* el paquete es el contenedor de todos los códigos fuente, de las librerías y de las cabeceras, además de todo lo que pueda necesitarse para ejecutar los trozos de código. La estructura que sigue es similar a la de cualquier otro lenguaje de programación.



*Fig. 4*

*Véase que el paquete contiene los archivos CMakeLists, Package, src y otros para garantizar el funcionamiento.*

.

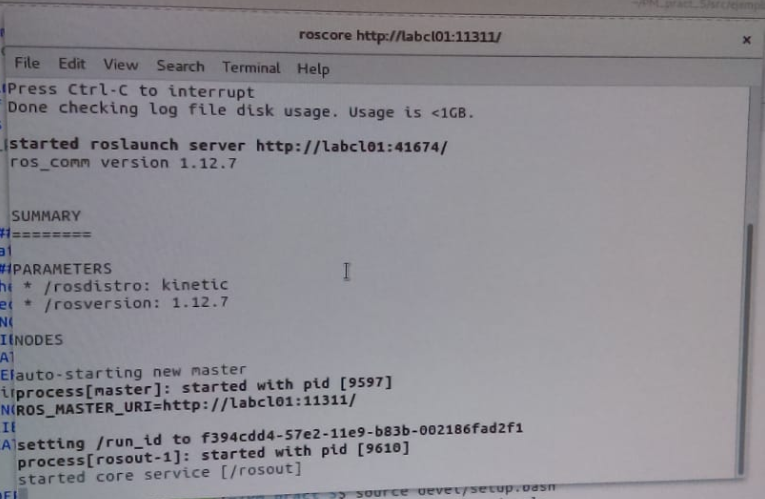
# Desarrollo

PARTE 1

La parte 1 consistió en hacer correr el código provisto en Github y posteriormente en hacer una modificación para mandar un mensaje en String en vez de un número entero. En concreto, primero hubo que inicializar ROS en una terminal y posteriormente inicializar el publicador en una segunda y el suscriptor en una tercera.

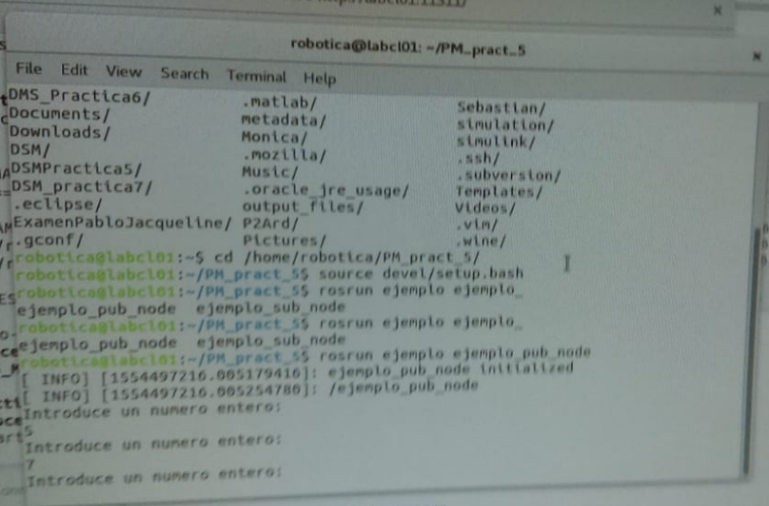
De manera más específica, se necesitaron correr las siguientes líneas de código:

* *$roscore* : este código inicializa el núcleo central de ROS desde el cual se entablará la comunicación con todos los otros nodos.
* *cd repo*: donde “repo” es la dirección del repositorio donde se guardaron los archivos provenientes de Github.
* *$catkin\_make:* crea un nuevo paquete de acuerdo con la dirección anterior.
* *$source devel/setup.bash:* encuentra los ejecutables dentro del paquete para el siguiente comando.
* *$rosrun packacge exec:* este es el código que corre el ejecutable “exec” que está en el paquete “package”. Evidentemente, en el código real, se sustituyeron dichos valores, como lo muestran las figuras que están a continuación.



*Fig. 5*

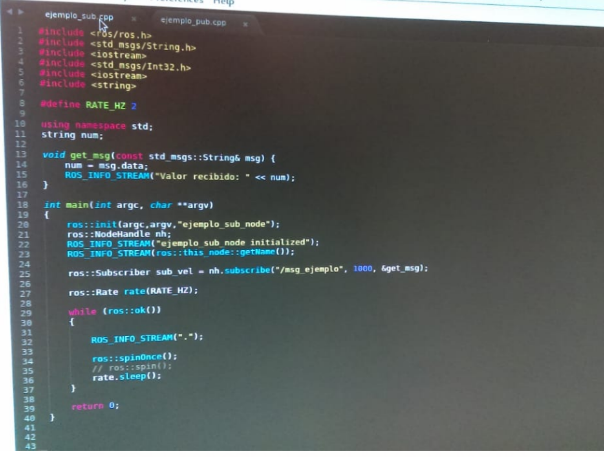
*Código de inicialización de ros en la primera terminal.*

**

*Fig. 6*

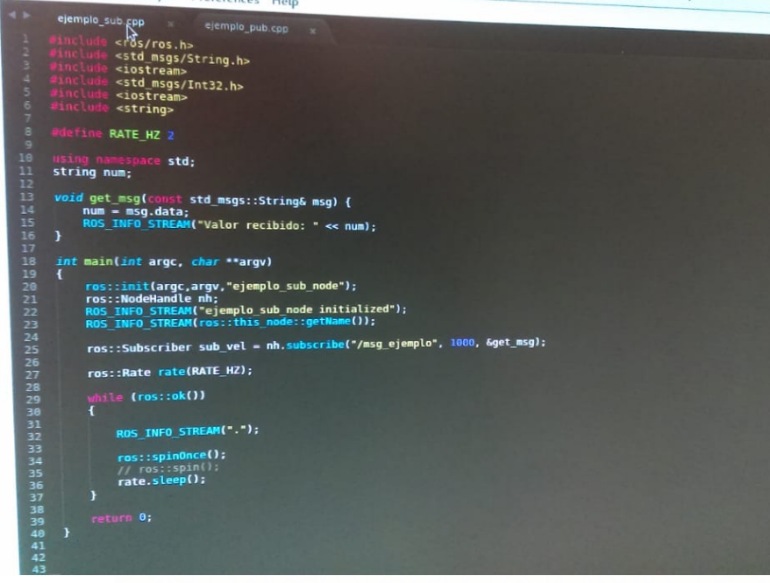
*Código correspondiente a la primera parte. Véase que se ejecutan todas las líneas descritas en los párrafos de arriba, pero con nombres diferentes en los archivos y paquetes.*

Una vez que se ejecutó con éxito el código de ejemplo, se modificó para que la entrada no fuese un número entero, sino un String. Para esto, se usó el editor de texto “Sublime”. Las librerías extra necesarias para la correcta ejecución y las modificaciones tanto al suscriptor como al publicador, así como la comprobación del resultado, se muestran en las siguientes figuras:

**

*Fig. 7*

*Código correspondiente al publicador. Obsérvese la línea 16 del código. En ella se realizó una modificación para que el parámetro no fuese un entero, sino un String, mediante “std\_msgs::String”. Para ello, fue necesario el comando include de la línea 5.*

**

*Fig. 8*

*Código correspondiente al suscriptor. Véase que en la línea 13 se espeficica que el mensaje a recibir está en formato String. Finalmente, en la parte del main es donde se lleva a cabo la acción de suscribirse al tópico que tiene el mismo nombre “msg\_ejemplo” que el puesto en el publicador*

En la sección de resultados se muestra la interacción exitosa entre estos dos archivos.

PARTE 2

La segunda parte de esta práctica fue ejecutar el ejemplo de introducción a ROS llamado Turtlesim., el cual consiste en una simulación de un robot en forma de tortuga que se mueve mediante comandos ROS en una interfaz gráfica que simula el mar. El objetivo era hacer que la tortuga se moviera de la siguiente manera al presionar los botones correspondientes:

* Hacia adelante si se presiona el 8
* Hacia atrás si se presiona el 2
* Girar en sentido de las manecillas del reloj si se presiona el 6
* Girar en sentido opuesto a las manecillas del reloj si se presiona el botón 4

Para ello, haciendo énfasis en que es una simulación de introducción, no tuvimos que programar nada, con la excepción de modificar una sección de código que mencionaremos a continuación.

Primero, en una terminal se ejecuta el comando *roscore* correspondiente a la ejecución de simulaciones en la terminal Linux; luego, en otra terminal se ejecuta el comando *rosrun turtlesim turtlesim\_node*  que llama al nodo de la simulación y muestra la interfaz gráfica de la tortuga.

Ahora, para poder darle instrucciones de movimiento a la tortuga, se recurre en una tercer terminal al comando *rosrun turtlesim teleop\_turtle\_key q*ue ya se encuentra en el repositorio *ros\_tutorials/turtlesim.* Es en este último comando mencionado donde se hacen las modificaciones pertinentes a lo requerido en la práctica, pues por default, los movimientos se hacen al presionar las flechas del teclado. Solo hay que cambiar la configuración en ASCII hexadecimal de las variables *KEYCODE\_R, KEYCODE\_L, KEYCODE\_U y KEYCODE\_D* para que ahora la tortuga se mueva cuando se presionan los números deseados.

PARTE 3.

Si bien, no se pudo concluir adecuadamente esta sección se requería que, al enviar un comando desde un nodo publicador, se encendiera un LED conectado al microcontrolador Arduino durante tres segundos. Para lograrlo, se tenía que crear también un nodo suscriptor que escuchase los comandos enviados por el publicador para poder cumplir con los requerimientos. Nuestro problema radicó en lograr la conexión entre ambos nodos, por lo que el mensaje no se pudo trnasmitir.

PARTE 4.

En esta parte se pretendía ejecutar nuevamente la simulación *Turtlesim*, solo que, en esta ocasión, además de utilizarse múltiples terminales, se usarían dos computadoras; una para correr la interfaz gráfica y otra para controlar los movimientos de la misma. Para ello, fue necesario decirle a ROS en cuales computadoras se encontrarían los procesos mediante las direcciones IPv4 de cada una.

Con el comando *ifconfig* se pudo obtener la dirección IP de cada computadora: 148.205.37.91 y 148.205.37.94 para la máquina máster y la máquina controladora respectivamente. En ambas computadoras se requirió exportar las variables de ambiente *ROS\_HOSTNAME y ROS\_MASTER\_URI* del modo que indicaba la práctica. “En cada terminal que se utilice, utilizar el comando export para asignar valor a la variable de ambiente ROS\_HOSTNAME de la siguiente manera:”

*export ROS\_HOSTNAME=ip\_computadora*

siendo *ip\_computadora* la ip de la propia computadora donde se exportaban las variables de ambiente.

Y para cada terminal utilizada

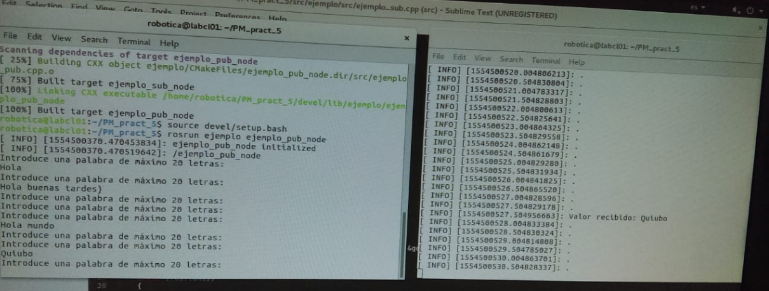
*export ROS\_MASTER\_URI=http://ip\_master:11311/*

donde *ip\_master* es la ip de la computadora master y 11311 es el puerto que ROS utiliza para comunicarse.

Luego de hacer las configuraciones descritas en el párrafo anterior, se corrió, desde la computadora denominada máster, el comando *roscore* en una terminal y en otra el comando rosrun turtlesim turtlesim\_node. En la otra computadora se ejecutó el comando rosrun turtlesim teleop\_turtle\_key para controlar los movimientos del robot. Una vez realizado lo anterior, se llevó a cabo esta parte de manera exitosa.

# Resultados

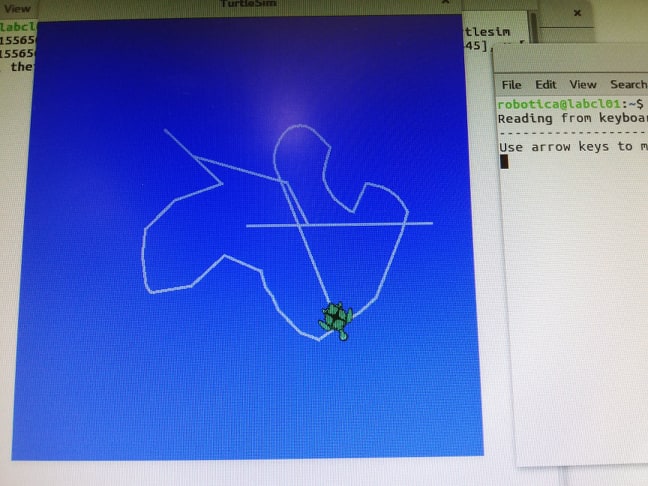
La implementación de toda la primera parte fue exitosa. A continuación se muestra una figura en la cual se aprecia la escritura del mensaje con el publicador para su recibimiento con el suscriptor.



*Fig.9*

*Véase que el mensaje “Quiubo” es correctamente recibido.*

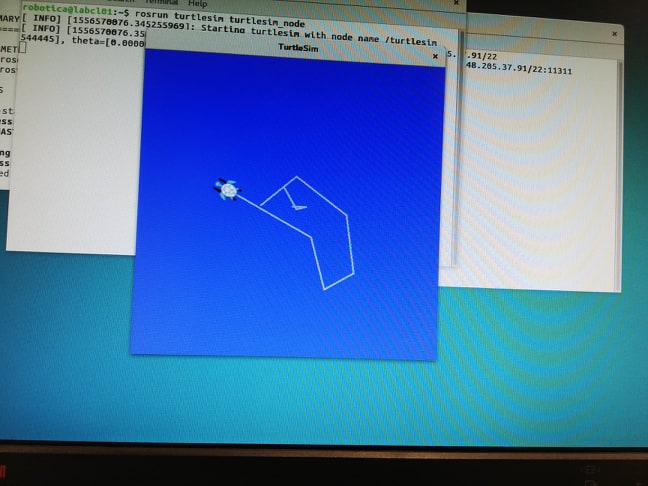
Para la segunda parte, luego de probar un poco las instrucciones de movimiento, esta fue la trayectoria seguida por la tortuga:



*Fig. 10*

*Trayectoria uno.*

Ahora, para la parte cuatro el resultado fue prácticamente el mismo, ya que, a pesar de que las instrucciones vinieran de otro ordenador, los comandos eran los mismos y el funcionamiento era idéntico. A continuación la trayectoria:



*Fig. 11*

*Trayectoria dos*

# Conclusiones

Pablo A. Ruz Salmones: ROS es un lenguaje que en definitiva será útil para el proyecto final. Además, logramos tener una familiarización mayor con Linux y las conexiones existentes entre ROS y Arduino. Finalmente, también es importante decir que hubo partes de la misma que resultaron complicadas y que nos llevaron más tiempo del que deberíamos de haber tardado.

Victor M. Thomas Ruiz: Esta práctica resultó muy interesante, pues es una pequeña muestra de lo que será el proyecto final. El hecho de habernos introducido a ROS da pie a la posibilidad de conocer más este sistema operativo y poder implementarlo a sistemas mecatrónicos, que es sin duda el objetivo. Durante el desarrollo de la práctica nos pudimos familiarizar más con el uso de la terminal en Linux y aprendimos a usar los comandos roscore y rosrun, que nos ayudarían a entender el funcionamiento de los publicadores y suscriptores, así como implementar una conexión entre ROS y Arduino. Por tales motivos considero que llevar a cabo la presente práctica nos fue de gran ayuda para nuestro aprendizaje.

# Roles

Pablo A. Ruz Salmones: tanto la práctica como el reporte fueron realizados en equipo. Hubo muy poca diferencia en las labores que realizamos, y logramos que la distribución del trabajo, tanto en el reporte escrito como en las prácticas, fuera adecuado.

Manuel Thomas: En esta ocasión, nuevamente, tanto Pablo como yo trabajamos de igual modo en ambas partes de la práctica, tanto en la sesión de laboratorio como en la elaboración del reporte escrito. Logramos una mejor distribución del trabajo en donde ambos nos desenvolvimos de igual modo.

# Referencias

[I] “Robot Operating System - ROS”. http://www.robotica.itam.mx/; Eagle Knights. Disponible en: <http://www.robotica.itam.mx/documents/pm_practica5.pdf>

[II] “Turtlesim Tutorials.” ROS.org. Disponible en: <http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line>

[III] “ROS tutorials” ROS.org. Disponible en: <http://wiki.ros.org/ROS/Tutorials>

[IV] “Arduino IDE Setup”. ROS.org. Disponible en: <http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup>

[V] “ROS. Sistema operativo para robótica; nociones y aplicaciones”. Núñez, T., León, C. & Cárdenas, P. Universidad de Pamplona. Disponible en: <https://www.researchgate.net/publication/267394769_ROS_SISTEMA_OPERATIVO_PARA_ROBOTICA_NOCIONES_Y_APLICACIONES/download>

1. [↑](#footnote-ref-1)