TALLER

Robot

Operating

System

https://www.ros.org/

INTRODUCCIÓN

ROS es una plataforma software para robótica de código abierto, que se emplea tanto para robots comerciales como para investigación. Proporcionando ciertas características para evitar que cualquiera que empiece a desarrollar el software de un robot, tenga que empezar de cero. Como veremos, en si ROS pese a su nombre, no es un sistema operativo, sino un meta sistema operativo que provee ciertas funcionalidades. Por este motivo ha de instalarse sobre otro sistema operativo y así ROS extiende su funcionalidad adaptándolo a aplicaciones robóticas. Para esto dispone de herramientas de desarrollo, visualización, depuración, etc.
Es por eso que ROS y Linux van de la mano, cada versión de ROS está asociada a una distribución de Linux, más concretamente, a una versión de Ubuntu que es la distribución que soporta completamente ROS. Por lo que debemos tener unas habilidades mínimas en GNU/Linux para poder trabajar con ROS. A nivel de programación, en ROS, principalmente se emplean dos lenguajes C++ y Python. Por su sencillez y claridad de código emplearemos Python en esta guía para poder centrarnos en otros aspectos.

Se facilita una máquina virtual con Linux+ROS, más concretamente Ubuntu 20.04 LTS + ROS Noetic
Password: huro
https://drive.google.com/file/d/1vlOCSgBbAXNiOJ2qMqSrETCAPGj4XBV5/view?usp=sharing

Otras maquinas virtuales de ROS Industrial
Password: rosindustrial
https://rosi-images.datasys.swri.edu/

Además, en este enlace puedes encontrar una guía de cómo realizar la instalación http://wiki.ros.org/noetic/Installation/Ubuntu.
El objetivo es que el alumno entienda los conceptos básicos de la plataforma para robótica ROS.

PYTHON

El lenguaje de programación Python.
Python es un lenguaje de programación de uso general que es ampliamente utilizado en áreas como la robótica, visón artificial, inteligencia artificial, etc. Es un lenguaje interpretado, de tipado dinámico, fuertemente tipado, orientado a objetos y multiplataforma.

Lenguaje interpretado.

Al ser un lenguaje interpretado se ejecuta utilizando un programa intermedio llamado interprete, en vez de usar un compilador para generar un código máquina que sea capaz de entender el ordenador. La ventaja de los programas compilados es que su ejecución es más rápida, sin embargo, un lenguaje interpretado es más portable y flexible. De todos modos Python tiene parte de las ventajas del código compilado, dado que se trata de un lenguaje semi interpretado esto quiere decir que el código se traduce la primera vez que se ejecuta a un pseudo código máquina (como java) llamado bytecode generando archivos .pyc o .pyo (bytecode optimizado) que son los que se ejecutaran en las siguientes ocasiones.

El intérprete de Python.

El intérprete de Python está disponible en múltiples plataformas Windows, GNU/Linux, MacOS, Unix, etc. Por lo que si no utilizamos librerías especificas nuestro código podrá ejecutarse en cualquier plataforma.

De tipado dinámico.

Quiere decir que una variable no tiene tipo hasta que se le asigna un valor, a partir de este momento la variable será de ese tipo. Es decir, si a una variable le asignamos un entero la variable será de tipo entero y si le asignamos una cadena será de tipo cadena.

Fuertemente tipado.

No se permite a una variable tratarla como si fuese de un tipo distinto sin una conversión explicita de tipos.

Orientado a objetos.

Esto quiere decir que cumple con el paradigma de programación orientado a objetos, donde elementos significativos del mundo real son modelados como clases y objetos.

Fuentes de información

Encontraremos multitud de ejemplos, tutoriales, guías, etc. De Python en la red sin perder de vista el sitio oficial https://www.python.org/ donde podremos encontrar tutoriales, referencias del lenguaje, etc.

Nuestro primer programa en Python.

Vamos a empezar con el famoso programa “Hola mundo” en Python disponemos de dos opciones a la hora de ejecutar nuestro condigo una será introducir las instrucciones directamente en el intérprete de Python y la segunda es escribiendo nuestro condigo en un archivo de código fuente que después ejecutaremos.
Comencemos con la forma más sencilla que consisten en ir introduciendo las instrucciones en el intérprete de comandos. Para lo que vamos a abrir una terminal como vimos en la primera parte y ejecutar la siguiente instrucción:

```
$python3
```

Esto abrirá el intérprete de comandos de Python que será algo parecido a

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

En este punto Podemos introducir nuestro hola mundo que será

```
>>>print(‘Hola Mundo’)
```

Y la respuesta ha de ser

```
Hola Mundo
>>>
```

Para salir del interprete de comandos

```
>>>exit()
$
```

Para probar la segunda forma de ejecutar Código Python vamos a crear en nuestro editor de textos favorito (Gedit, Nano, Atom) la misma línea que introducimos antes y vamos a guardar el archivo como HolaMundo.py.

```
#!/usr/bin/env python3
print('Hola Mundo')
```

Una vez creado nuestro archivo de código fuente vamos a ejecutar nuestro programa y lo primero que tendremos que hacer es asignarle permisos de ejecución.

```
$ chmod +x tuScript.py
```

Ahora podemos ejecutar nuestro script con

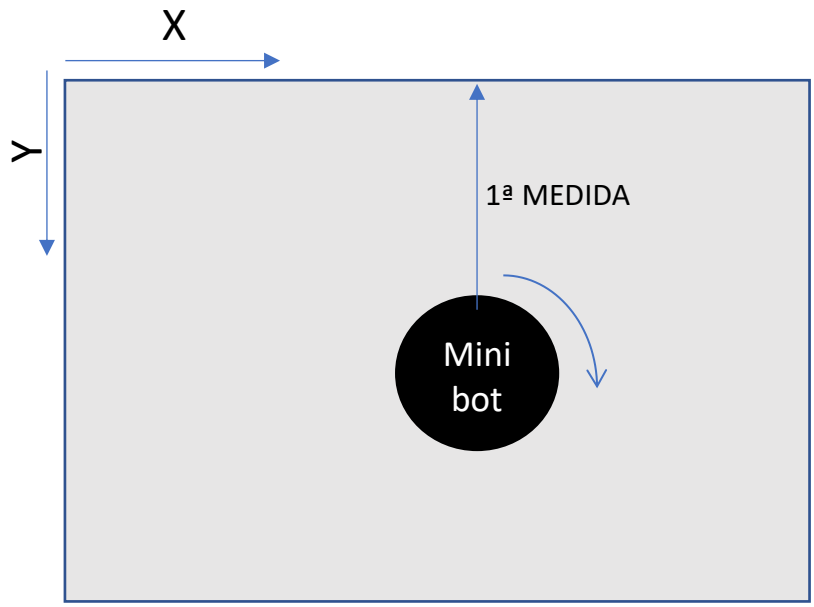
```
$ ./HolaMundo.py
Hola Mundo
```

Repaso de la documentación.

Queda fuera de la presente guía de prácticas realizar una guía de programación de Python, el alumno puede visitar la web oficial donde encontrará un completo tutorial del lenguaje https://docs.python.org/3/tutorial/index.html , también se adjunta con la practica tanto el original como una traducción al español del tutorial de Python de Guido Van Rossum(creador de Python).
Se recomienda realizar una lectura de los aspectos más básicos del lenguaje, tipos de datos, sentencias de control, comentarios y funciones. En clase se explicarán los aspectos de Python relacionados con la programación orientada a objetos.

Ejercicio 1

El segundo programa que vamos a realizar nos servirá para afianzar los conocimientos adquiridos. En este caso el alumno tiene que desarrollar un programa en Python que se llamará **minibot.py.** Este simulará un pequeño robot móvil que se encuentra en un punto de una habitación de paredes paralelas dos a dos, saluda diciendo que es Minibot y que si queremos escanear la habitación. En caso de que contestásemos que sí, nos pedirá cuatro medidas del láser. Una vez hemos introducido las cuatro medidas mostrará como resultado cuál es el largo y ancho de la habitación, la superficie total y la posición del robot en coordenadas X, Y. En caso de que le contestemos que no, Minibot se despide y termina el programa. Se ha de entregar el código fuente debidamente comentado.



ROS

ROS Arquitectura y conceptos.

Para entender mejor cual es la arquitectura de ROS igual que hicimos antes, vamos a ejecutar un pequeño ejemplo que nos ayudará a la hora de explicar ciertos conceptos.
Vamos a abrir una terminal e introduciremos el siguiente comando:

```
$roscore
```

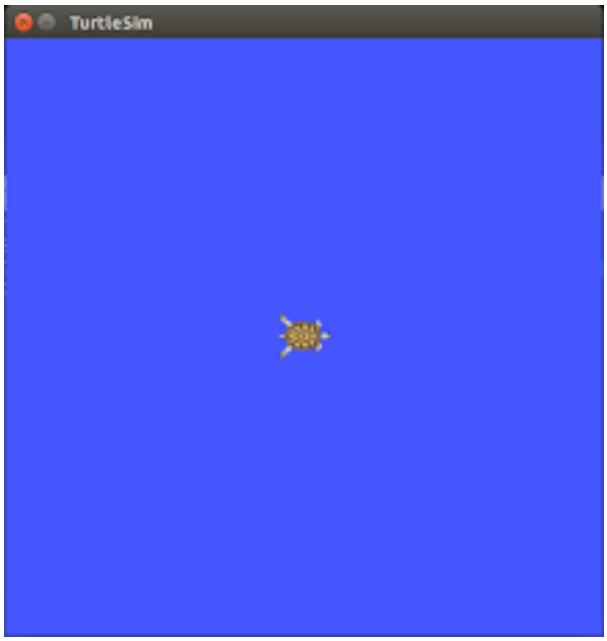
Si todo es correcto acto seguido tiene que arrancar en nodo principal de ROS mostrando …

```
... logging to /home/huro/.ros/log/49d6af12-f1be-11e9-8925-080027131460/roslaunch-mayr-
6214.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mayr:40629/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [6225]
ROS_MASTER_URI=http://mayr:11311/

setting /run_id to 49d6af12-f1be-11e9-8925-080027131460
process[rosout-1]: started with pid [6238]
started core service [/rosout]
```

En estos momentos está corriendo el nodo maestro, sin cerrar esta terminal abriremos otra nueva e introduciremos:

```
$ rosrun turtlesim turtlesim_node
```

Una vez más si todo es correcto aparecerá una ventana con una tortuga en el centro.



Y este será el “Hola Mundo” de ROS, pero vamos a aprovechar para explicar la arquitectura de ROS en base a este ejemplo. ¿Qué es lo que ha sucedido? Primero hemos ejecutado una instrucción que ha iniciado el nodo maestro. Éste tiene que estar siempre ejecutándose mientras queramos que funcione nuestro robot. El maestro queda a la escucha hasta que otro nodo se inicia, que es lo que hemos hecho en la segunda terminal. Cuando se inicia un nodo cualquiera, se registra en el nodo maestro indicando toda la información que puede resultar útil para el resto de los nodos. Como hemos comentado antes, la piedra angular de ROS es facilitar la comunicación entre nodos y esto es así porque dentro del software de control de un robot recibiremos información de múltiples sensores, aplicaremos un control y enviaremos información a los actuadores. ¿Quiere decir esto que todas las comunicaciones pasan por el nodo maestro? No, el nodo maestro no está en medio de todas las comunicaciones pudiendo convertirse en un cuello de botella, sino que hace la función de una telefonista. Inicialmente contactas con el nodo maestro para conocer dónde se encuentra el nodo con el que vamos a comunicar y una vez establecida la comunicación está se realiza entre nodos. Otra gran ventaja que radica en esta arquitectura es que la aplicación puede estar distribuida en distintas máquinas siempre que no exista más de una en la que se ejecuta el nodo maestro.
Resumiendo, ROS estará compuesto por los siguientes elementos:

Master

El maestro será donde se registran los nodos y donde acuden a buscar a otros nodos.

```
$ roscore
```

Nodes

Son los procesos que realizan las distintas tareas necesarias para nuestro robot, como por ejemplo leer información de un sensor de distancias laser o enviar una referencia de velocidad a una rueda de nuestro robot.

```
$ rosrun package_name node_name
```

Ejecuta node_name del paquete package_name.

```
$ rosnode list
```

Muestra un listado de los nodos activos.

```
$ rosnode info node_name
```

Muestra información del nodo node_name.

Parameter Server

El servidor de parámetros permite el almacenamiento de valores en base a una clave de forma centralizada y común a todos los nodos. El “Parameter Server” utiliza YAML como leguaje de marcado.

```
$ rosparam list
```

Muestra un listado de los parámetros.

```
$ rosparam set param_name value
```

Guardar un valor en los parámetros

```
$ rosparam get param_name
```

Nos muestra en valor de un parámetro.

Messages

Los nodos se comunican entre ellos pasándose mensajes. Un mensaje es simplemente la definición de una estructura de datos para modelar la información que compartirá. La descripción se guarda en un archivo .msg.

Topics

Una de las formas de comunicarse entre nodos es mediante “Topics”. Un Topic es el nombre que recibe el canal de comunicación. Los nodos pueden publicar o subscribirse a un Topic, normalmente suele ser una arquitectura 1 publica N suscriben.

```
$ rostopic list
```

Muestra una lista de los Topics activos.

```
$ rostopic echo /Topic
```

Se suscribe y muestra el contenido de Topic.

```
$ rostopic info /Topic
```

Muestra información del Topic.

Service

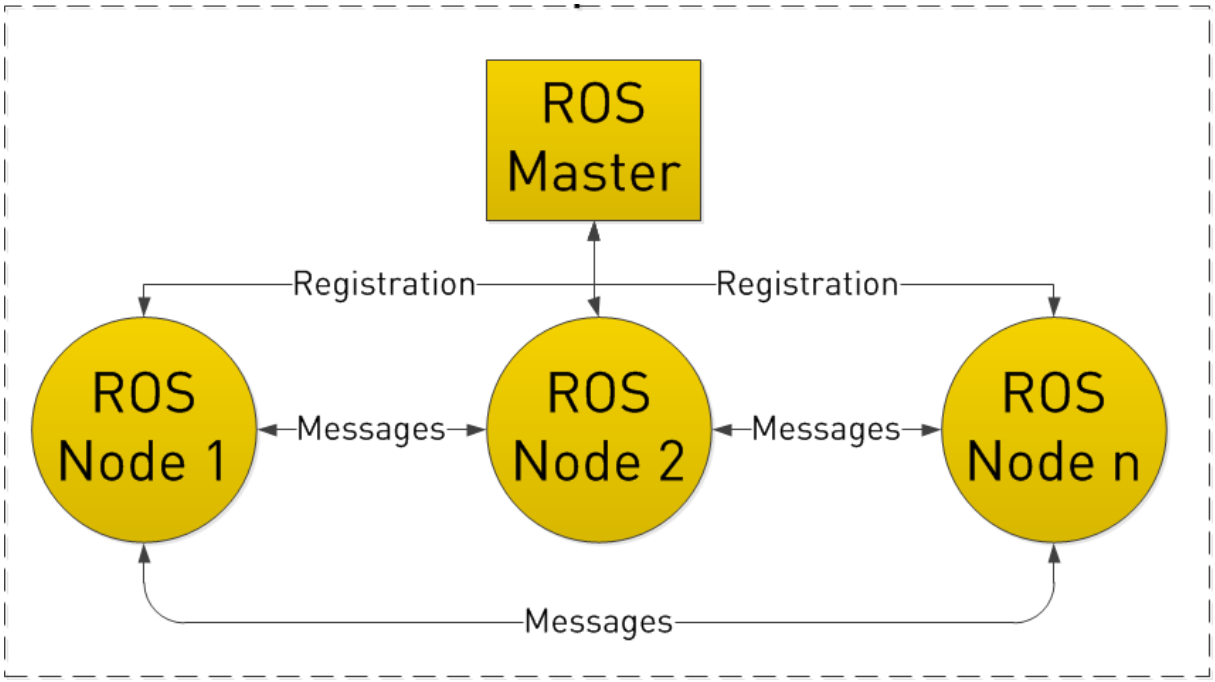
Los servicios son la otra forma que tiene ROS de comunicar entre nodos. Pero en los servicios el nodo puede enviar una petición y recibir una respuesta.

```
$ rosservice list
```

Lista los servicios activos.

```
$ rosservice type service_name
```

Muestra de qué tipo es el servicio.



Hemos visto cuales son los principales aspectos de la arquitectura de ROS en su funcionamiento. A nivel de desarrollo en ROS el código se organiza en paquetes y metapaquetes. Un paquete puede contener uno o varios nodos, más el software necesario que haga fácil de utilizar estos módulos por terceros. Además, debería cumplir la norma de implementar la suficiente funcionalidad como para ser útil, pero evitar ser demasiado pesado. La creación y compilación de estos paquetes se gestiona con la herramienta Catkin. Catkin son una serie de macros de compilación a bajo nivel para ROS. Para poder trabajar en ROS con Catkin tenemos que crear un espacio de trabajo, que será el que albergue los paquetes que estamos desarrollando . Para crear un espacio de trabajo ejecutaremos estos comandos;

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Esto tiene que haber creado la siguiente estructura de directorios:

Catkin_ws/
- build/
- devel/
- src/
- install

De todas estás carpetas ahora mismo nos vamos a centrar en “src” que es donde crearemos todos nuestros paquetes. Así, tras asegurarnos que estamos dentro de “catkin_ws/src/” ejecutaremos la siguiente instrucción:

```
$ catkin_create_pkg mi_primer_package rospy
```

Esto creará dentro de nuestra carpeta src un nuevo paquete con ciertos archivos. Para verificar que todo esta correcto vamos a crear nuestro primer script de Python y lo haremos, cómo no, dentro de la carpeta src de nuestro recién creado paquete. El contenido de nuestro script será:

```
#! /usr/bin/env python

import rospy

rospy.init_node('SoyNodo')
print "Hola mi nombre es SoyNodo, encantado de conocerte"
```

Nota.-Recuerda asignar permisos de ejecución al script. (simple.py)

Ahora tenemos que crear el launch_file. Un launch_file es una herramienta que empleamos para cuando tenemos que iniciar nodos con parámetros o de forma coordinada. Por lo que creamos una carpeta llamada launch dentro de la carpeta de nuestro paquete al mismo nivel que el src donde hemos puesto nuestro script de Python.

```
$ roscd mi_primer_package
$ mkdir launch
```

Dentro crearemos un archivo con el nombre mi_primer_package_launch_file.launch con el siguiente contenido:

```
<launch>
    <!—Mi primer Package launch file -->
    <node pkg="mi_primer_package" type="simple.py" name="SoyNodo" output="screen">
    </node>
</launch>
```

Como último paso vamos a lanzar nuestro nodo con la instrucción

```
$ roslaunch mi_primer_package mi_primer_package_launch_file.launch
```

Si todo es correcto en nuestra terminal debe aparecer algo parecido a lo siguiente:

```
catkin_ws$ roslaunch mi_primer_package mi_primer_package_launch_file.launch
... logging to /home/user/.ros/log/1f10268c-f36c-11e9-957b-0a26c7f36644/roslaunch-
rosdscomputer-25371.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://rosdscomputer:34223/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    SoyNodo (mi_primer_package/simple.py)

ROS_MASTER_URI=http://master:11311

process[SoyNodo-1]: started with pid [25510]
Hola mi nombre es SoyNodo, encantado de conocerte
```

EJERCICIO 2

Para verificar que el alumno ha entendido todos los pasos para crear paquetes, nodos y ejecutarlos, accede a la siguiente dirección y realiza el tutorial de “Simple Publisher and Subscriber”
http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29
Para que sea una versión mejorada, el alumno ha de personalizar los mensajes añadiéndole un toque de originalidad al ejemplo modificando los mensajes. Ha de entregarse la carpeta del paquete completa con todo lo necesario para poder ejecutarse.