UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERIA

IMPI INSTITUTO DE INGENIERIA MECANICA Y PRODUCCION INDUSTRIAL

Introducción a ROS

Fundamentos de Robótica Industrial

¿Qué es ROS?

- Es un ambiente de desarrollo de código abierto para sistemas robóticos.
- Proporciona un conjunto de herramientas y librerías que simplifican considerablemente la creación de aplicaciones complejas para robots mediante la reutilización de código
- Busca unificar (estandarizar) la forma de trabajo en el área
- Promueve el intercambio entre quienes lo utilizan
- Facilita la interconexión entre componentes y diversos robots
- Historia:
    - Iniciativa del Stanford Artificial Intelligence Laboratory (2007) y su desarrollo fue continuado por Willow Garage.
    - Desde el 2013 es gestionado por OSRF (Open Source Robotics Foundation).

ROS Industrial

- Versión utilizada en la Industria
- Depende del ROS-I Consortium
- Se sustenta en ROS core y es totalmente compatible
- Se enfoca en la interoperabilidad y en la calidad del código y confiabilidad

¿Para qué sirve?

- Hace posible la interoperabilidad en automatización
- Provee librerías de gran capacidad compartidas de forma gratuita
- Crea un conducto en que la investigación académica penetra rápidamente en la industria

Ventajas

Reutilización de software

- Procesamiento de imágenes (image_pipeline)
- Transformación de coordenadas (tf)
- Interfaz gráfica (rviz, rqt_plot, dashboard)
- Navegación (navigation)
  - Path planning
  - SLAM
  - Mapping
- Modularidad: Todos los elementos están conectados por un sistema de mensajes distribuido lo que permite que en caso de tener problemas con algún componente puntual no caiga todo el sistema

Ventajas

Simplifica la instalación de Software: Dependencias de paquetes son resueltas a comandos “apt” que las instalan automáticamente

Formaliza el ciclo de desarrollo:

- Grabar datos a procesar
- Diseñar los algoritmos sobre los mismos datos
- Grabar nuevos datos
- Probar y depurar los algoritmos
- Probar el algoritmo en línea en el robot

Ventajas

Reproducibilidad de experimentos:

- Si se publica un artículo, se publica el software y los datos
- Cualquier revisor puede rápidamente instalar el software y probarlo sobre los datos

Especialización: Muchos aspectos de la robótica quedan “resueltos” y un investigador se puede concentrar en su área de conocimiento

Desventajas

Matar un mosquito con un cañón:

- La resolución de dependencias hacen que instalemos mucho software para resolver problemas simples
- A medida que aumenta el software que instalamos, aumentan las probabilidades de fallo de la instalación

Curva de aprendizaje empinada:

- Conceptos como programar en Python o C++, Remote procedure calls (RPC), “build system” software (Make, CMake, catkin), etc nos complican un poco el arranque
- Falta de GUI

Instalación

- El simple hecho de instalarlo y que funcione nos da una idea de a qué nos enfrentamos
- En EVA encuentran una guía de instalación de ROS
- http://wiki.ros.org/
  - Instalación: http://wiki.ros.org/ROS/Installation
  - Tutoriales: http://wiki.ros.org/ROS/Tutorials
- Cheat Sheet: https://github.com/ros/cheatsheet/releases

Conceptos Principales: Nodos

Equivalente a un módulo

- Programa independiente en ejecución (p.e. driver de sensores, driver de actuadores, cartógrafo, planificador, ...)
- Se compilan, gestionan y ejecutan de forma individual.
- Se programan utilizando una biblioteca cliente de ROS:
  - roscpp en c++
  - rospy en python
- Los nodos pueden publicar o suscribirse a un tópico.
- Pueden usar o proveer servicios.

Tópicos o temas

Un tópico es un nombre para un flujo de mensajes con un tipo de datos determinado

- p.e., datos desde un sensor laser podrían ser enviados a un tópico llamado scan, con mensajes de tipo LaserScan

• Los nodos se comunican con otros nodos mediante el envío de mensajes a tópicos.

• Comunicación asíncrona.

Mensajes

Estructura de datos estrictamente tipada para comunicación entre nodos

- Ejemplos
  - String
  - Twist es usado para expresar comandos de velocidad
    - Vector3 linear
    - Vector3 angular
  - Vector3 es otro tipo de mensaje definido como:
    - float64 x
    - float64 y
    - float64 z

Servicios

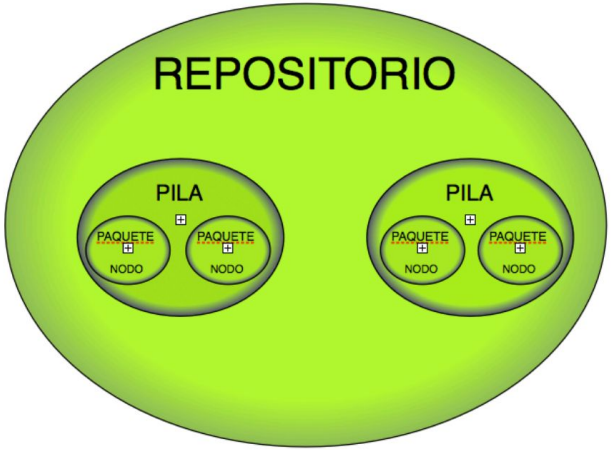
Transacción síncrona entre nodos (RPC, llamada a un procedimiento remoto)

- Modelo cliente/servidor: 1-to-1 request-response
  - Un nodo llama a un procedimiento que ejecuta en otro nodo
- Usos:
  - iniciar una funcionalidad o comportamiento
- Ejemplo
  - map_server/static_map, devuelve el mapa actual que está utilizando el robot para navegar

Paquetes

Unidad de compilación en ROS

- Contiene la definición de uno o más programas nodos
- Contiene información para facilitar la instalación
  - Dependencia de otros paquetes ROS
  - Dependencia de bibliotecas externas
- Contiene información para facilitar compilación
  - Definición de compilación en un entorno cmake (catkin_make)



Instalación de ROS

- Seguir tutorial de ROS
- Verificar ejecutando roscore
- Ver ejemplo teleop
  - rosrun turtlesim turtle_teleop_key
  - rostopic echo /turtle1/cmd_vel

Comandos Básicos

- roscore
- rosrun
- rosnode
- rostopic

Roscore

roscore es el primer comando a ejecutar cuando se comienza a utilizar ROS.

```
$ roscore
```

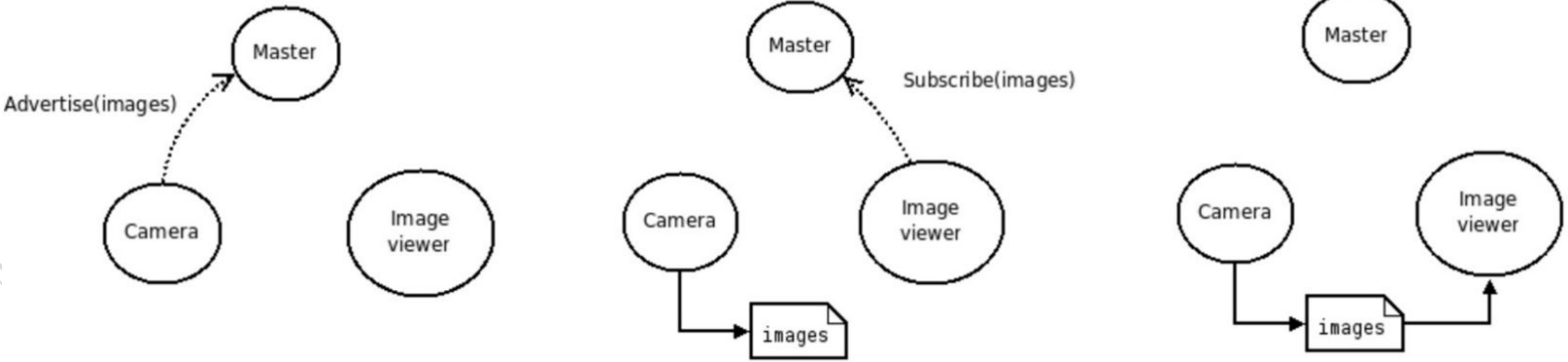
roscore levanta:

- Un master
- Un servidor de parámetros
- Un nodo rosout para logging

ROS Master

Proporciona información de conectividad a los nodos de forma que puedan intercambiar información

- Cada nodo se conecta al master al inicio para registrar los detalles de los mensajes que publica y los tópicos a los cuales se suscribe.
- Cuando un nuevo nodo se crea, el master le proporciona la información necesaria para realizar una comunicación directa peer-to-peer con otros nodos que comparten sus mismos tópicos.



rosrun

rosrun permite levantar un nodo.

- Forma de uso:

  $ rosrun <package> <executable>

- Ejemplo:

  $ rosrun uvc_camera uvc_camera_node

  $ rosrun image_view image_view image:=/image_raw

Ejemplo

En tres terminales separadas ejecutar los siguientes comandos:

```
$ roscore
```

```
$ rosrun turtlesim turtle_teleop_key
```

```
$ rosrun turtlesim turtlesim_node
```

rosnode

Muestra información de los nodos y permite gestionarlos.

• Opciones:

- list: lista los nodos activos
- ping: chequea la conectividad con un nodo
- info: muestra información del nodo
- kill: mata un nodo
- machine: lista los nodos ejecutando en una determinada máquina

rostopic

Permite obtener información de los tópicos y publicar en ellos.

• Opciones:

- list: lista los tópicos activos
- echo: imprime los mensajes que se publican en un tópico
- info: imprime información acerca del tópico
- type: imprime el tipo de mensaje que maneja el tópico
- pub: publica en el tópico

Ejemplo

Mostrar mensajes que llegan a un tópico

```
$ rostopic type /turtle1/pose

$ rostopic echo /turtle1/pose
```

- Para hacer que la tortuga se mueva hacia adelante a una velocidad de 0.1m/s.

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist'{linear: {x: 0.1, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

- O

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist'{linear: {x: 0.1}}'
```

roslaunch

Herramienta para levantar varios nodos y setear parámetros.

- roslaunch opera sobre un archivo launch (XML).

  $ roslaunch PACKAGE LAUNCH_FILE

- roslaunch automáticamente ejecuta roscore si es necesario.
- Ejemplo

  $ roslaunch turtle_tf turtle_tf_demo.launch

Soporte para hardware

Robots

– https://robots.ros.org/

– Ejemplos: ABB, Fanuk, Motoman, Roomba, Etc.

• Sensores

– https://wiki.ros.org/Sensors

– Ejemplos: Lidar, Camaras 2D y 3D, IMU, GPS.

• Actuadores

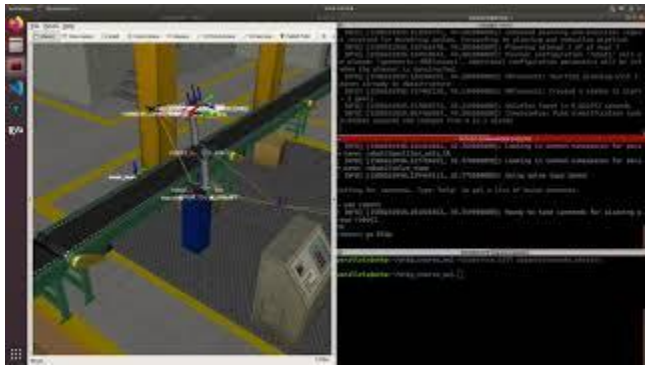
– https://wiki.ros.org/Motor%20Controller%20Drivers

– Ejemplos: Robotis Dynamixel, VESC.

Visualizador de ROS (RViz)

Proporciona visualización 3D de sensores y robots (URDF).

- Permite visualizar la información en un sistema de coordenadas común.
- Herramienta imprescindible para debug.



FIN!