Fundamentos de la Robótica Autónoma

Introducción a ROS

Facultad de Ingeniería
Instituto de Computación

mina
network management | artificial intelligence

Contenido

- ¿Qué es ROS?
- Ventajas
- Desventajas
- Nodos
- Tópicos
- Comandos

¿Qué es ROS?

- Robotic Operating System (ROS)
  - Es un ambiente de desarrollo para robótica que promueve la re-utilización de componentes.
  - Es un conjunto de bibliotecas y herramientas para el desarrollo de aplicaciones robóticas.
- Nace a partir de la ausencia de estándares para la robótica.
- Historia:
  - Es una 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: Five Years, https://youtu.be/PGaXiLZD2KQ
ROS: Ten Years, https://vimeo.com/245826128

¿Qué es ROS?

No es un sistema operativo:

- No maneja asignación de recursos como memoria o utilización de CPU.
- No mantiene contacto directo con el hardware.

Pero:

- Administra la ejecución de procesos.
- Administra la comunicación entre procesos y entre máquinas.
- Provee mecanismos de logging, depuración y accounting.

Ventajas

Reutilización de software

- Procesamiento de imágenes (**image_pipeline**)
- Transformación de coordenadas (**tf**)
- Interfaz gráfica (**rviz**, **rqt_plot**, **dashboard**)
- Logging (**rosout**)
- Navegación (**navigation**)
  - Path planning
  - SLAM
  - Mapping
- Abstracción de sensores (***_cam**, **openni_launch**)
- Abstracción de control (http://wiki.ros.org/Robots)
- Abstracción de software (**ar_pose**, **bfl**, **opencv_bridge**)

Ventajas

Simplifica la instalación de software

- Dependencias de paquetes son resueltas a comandos “apt” que las instalan automáticamente
- Herramientas como **rosinstall_generator** y **rospack** permiten determinar todo el software necesario para utilizar cualquier subporción de ROS

Ventajas

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 linea 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

Desventajas

Curva de aprendizaje de instalación:

- Fuera del amigable entorno de Ubuntu+i386, hay que compilar ROS desde los fuentes y resolver las dependencias de manera “semi-automática”
- Obtener una primera instalación funcional o agregar funcionalidades puede consumir tiempo

Desventajas

Demasiada abstracción:

- Se pierde la noción del desperdicio de recursos
  - Envío de mensajes: latencia, throughput y copia de datos
  - Niveles de abstracción: wrappers de python a funciones optimizadas en c++
- A veces no se usa bien una herramienta por no saber lo que se está haciendo

Wiki ROS

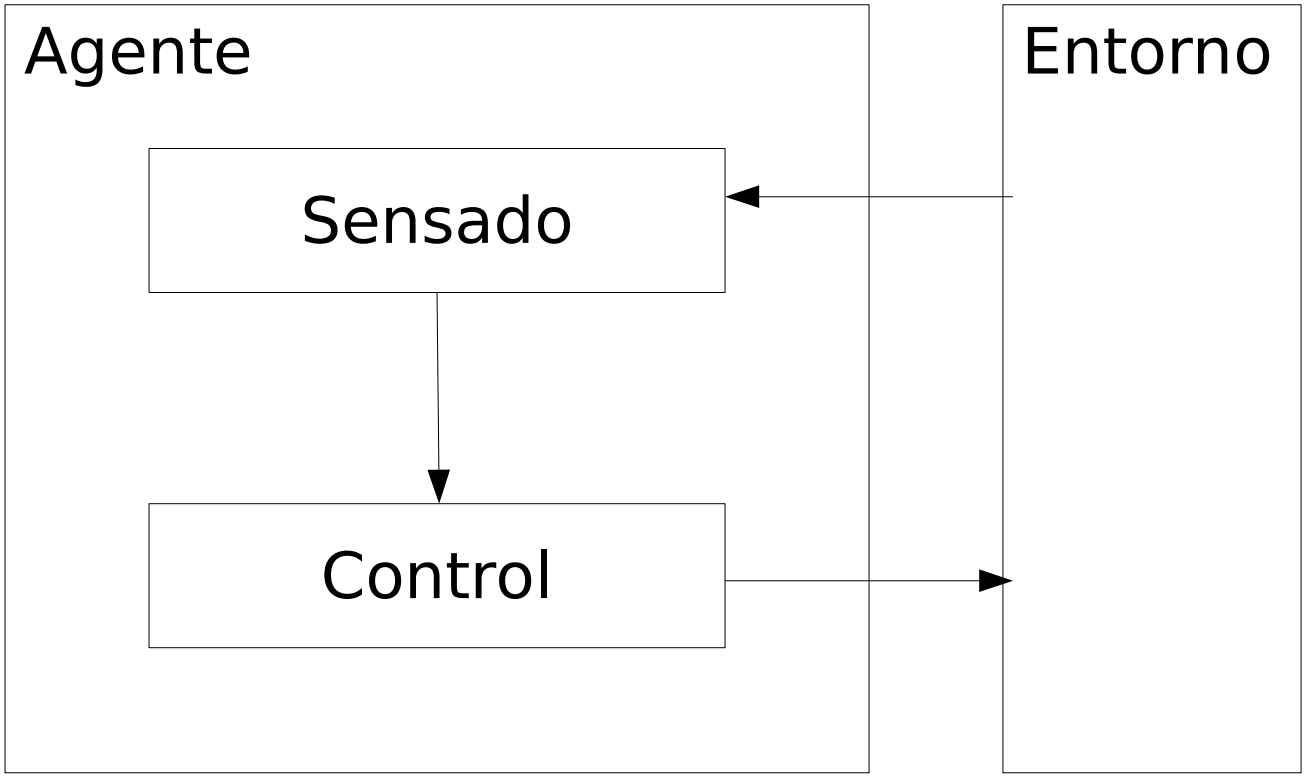
- http://wiki.ros.org/
  - Instalación: http://wiki.ros.org/ROS/Installation
  - Tutoriales: http://wiki.ros.org/ROS/Tutorials
- Cheat Sheet: https://github.com/ros/cheatsheet/releases

Conceptos de ROS

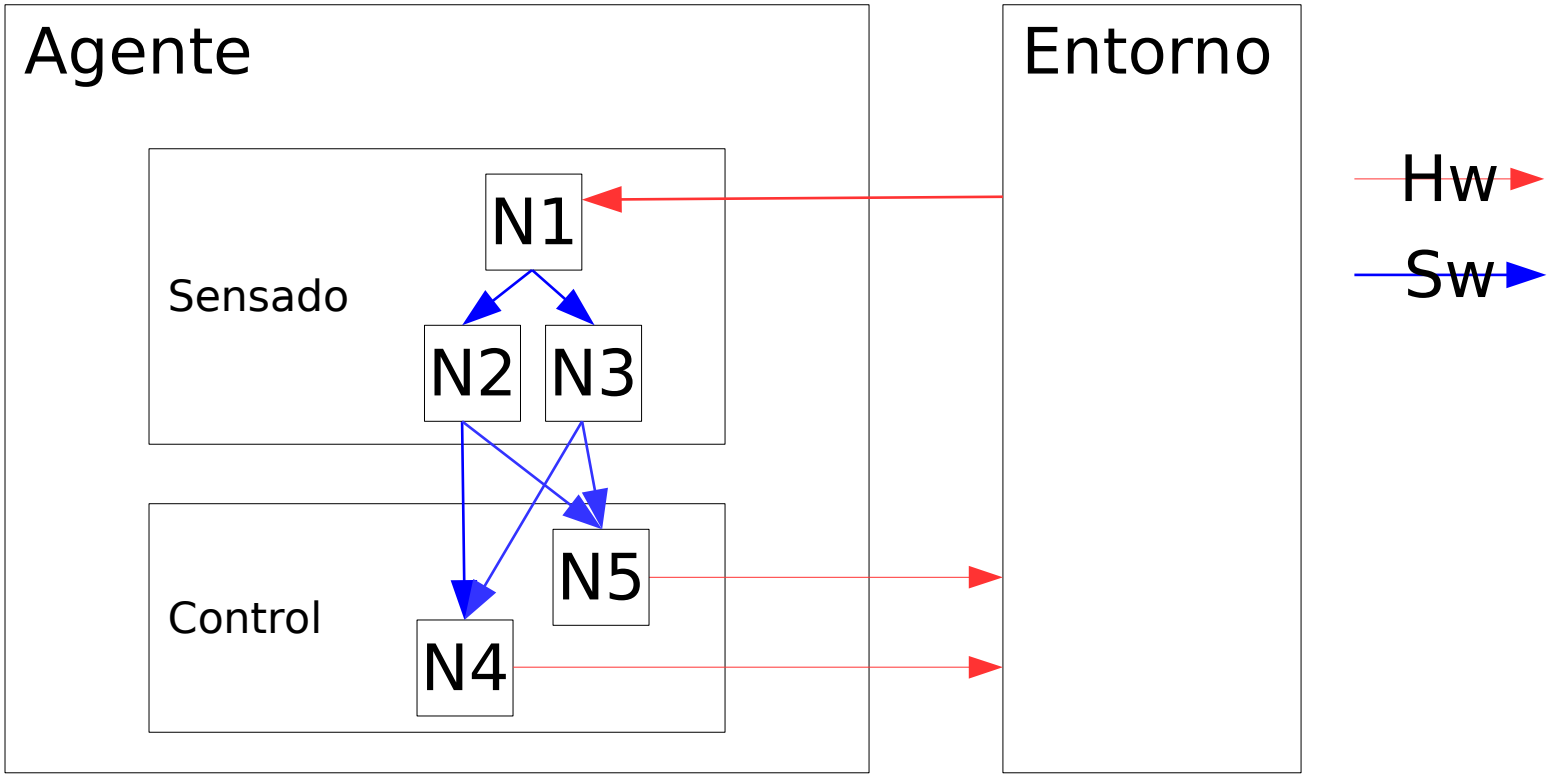
Nodos

- 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.

Programa Agente



Programa Agente ROS



Tópicos

- 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 en envío de mensajes a tópicos.
- Comunicación asíncrona.
- El modelo utilizado es broadcast 1-to-N.

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

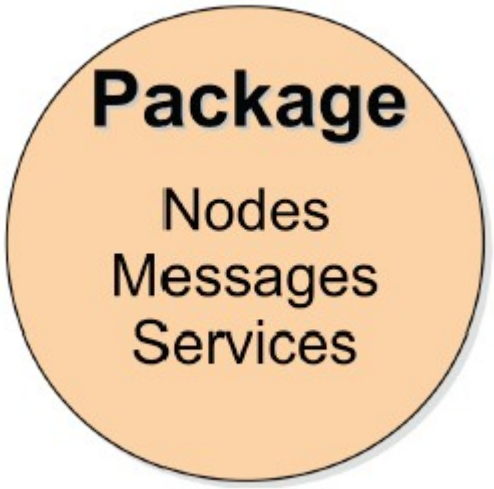
    float64 z

Servicios

- Transacción síncrona entre nodos (RPC)
- Modelo cliente/servidor: 1-to-1 request-response
- Usos:
- – realizar cómputo remoto
- – 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**)



Comprobar Instalación

- Abrir una terminal
- Ingresar '**export | grep ROS_MASTER_URI**' y presionar enter.

Ejemplo:

declare -x ROS_MASTER_URI="http://localhost:11311
"

Comprobar Instalación

- En una terminal ingresar '**roscore**' y presionar enter.

Ejemplo

...
SUMMARY
========

PARAMETERS
* * /rosdistro: jade*
* * /rosversion: 1.11.16*

NODES

auto-starting new master
process[master]: started with pid [22277]
...

Comandos básicos

- roscore
- rosrun
- rosnode
- rostopic

roscore

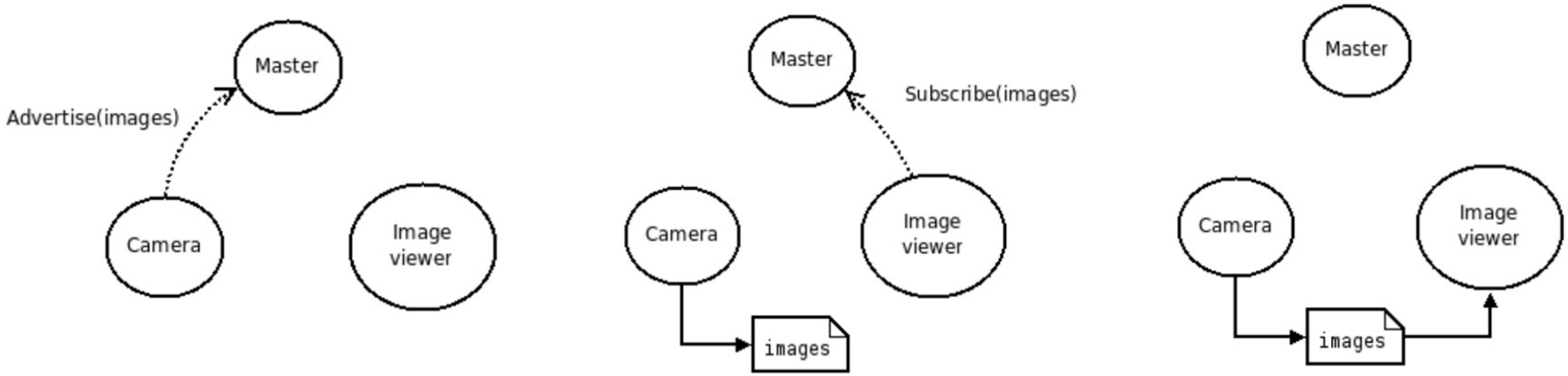
- roscore es el primer comando a ejecutar cuando se comienza a utilizar ROS.

```
$ 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:

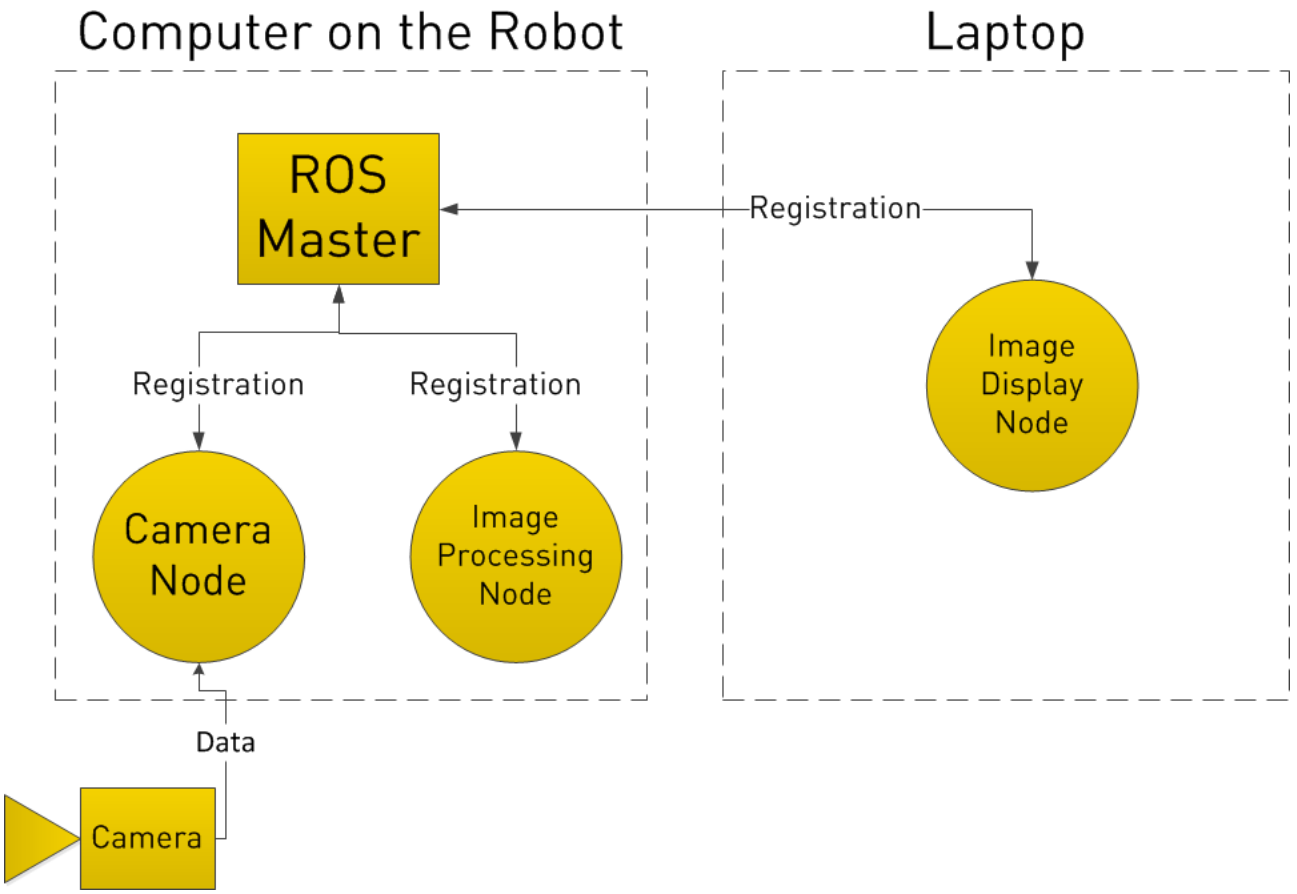
```
$ rosrun <package> <executable>
```

- Ejemplo:

```
$ rosrun uvc_camera uvc_camera_node
```

```
$ rosrun image_view image_view image:=/image_raw
```

Ejemplo


Computer on the Robot
Laptop
ROS Master
Registration
Registration
Registration
Image Display Node
Camera Node
Image Processing Node
Data
Camera

Demo

- En tres terminales separadas ejecutar los siguientes comandos:

```
$ roscore
```

```
$ rosrun turtlesim turtlesim_node
```

```
$ rosrun turtlesim turtle_teleop_key
```

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

Demo

- 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}}'
```

Demo (cont.)

- Mostrar servicios

```
$ ros ...
```

Nodo publicador

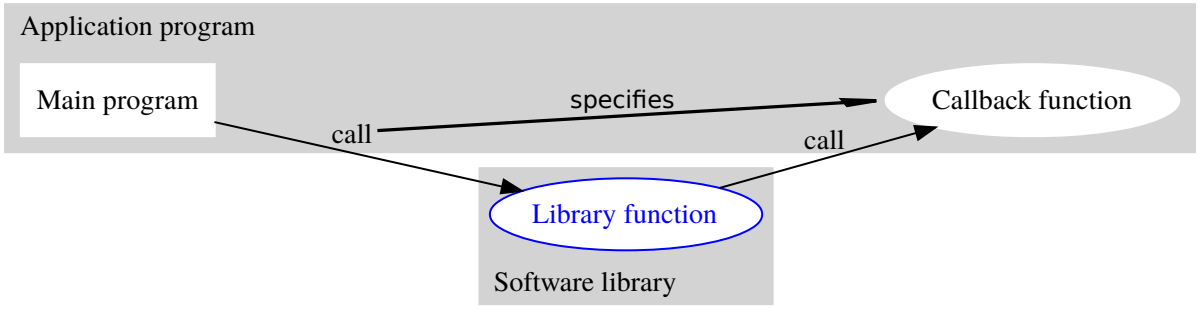
```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Nodo publicador - Explicado

- rospy.Publisher(name, type, size)
  - Registra un tópico en el nodo maestro.
  - size, limita los mensajes encolados en caso de suscriptores lentos.
- rospy.init_node(name, anonymous)
  - Los nombres deben ser únicos. Cuando el nombre del nodo no deba ser único puede invocarse con anonymous en True.
- rospy.rate(freq)
  - Crea un objeto de tipo Rate. Es útil para controlar la frecuencia de ejecución de un loop.
  - El cuerpo del loop debe demorar menos que el período.
- rospy.is_shutdown()
  - Chequea si el programa debe terminar (p.e. Ctrl-C).
- pub.publish(msg)
- rate.sleep()

Callback

- Un callback es un un fragmento de código pasado como argumento a otro código.
- La ejecución puede ser síncrona (bloqueante) o asíncrona (diferida).
- Los callbacks asíncronos se utilizan para manejar eventos o entrada/salida.
- Los callbacks asíncronos se apoyan en hilos de ejecución (thread) o interrupciones para realizar el call-back.



Callbacks en Python

- Ejemplo de callback síncrono

```
>>> def get_square(val):
...     """The callback."""
...     return val ** 2
...
>>> def caller(func, val):
...     return func(val)
...
>>> caller(get_square, 5)
25
```

- Ejemplo de callback asíncrono … recepción de mensajes en un tópico.

Nodo suscriptor

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

Demo

- Levanto un nodo publicado

```
$ rosrun demos talker.py
```

- Levanto un suscriptor

```
$ rosrun demos listener.py
```

- Crear un grafo que muestra lo que sucede en el sistema

```
$ rosrun rqt_graph rqt_graph # o simplemente rqt_graph
```

Demo

- Los nombre deben ser únicos. En caso de existir un nodo con el mismo nombre roscore instará al más viejo a cerrarse.

```
$ rosrun demos talker.py __name:=talker1
$ rosrun demos talker.py __name:=talker2
```

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: Darin Op, Heron, Nao, Pioneer, Turtlebots, Heron.
- Sensores
  - https://wiki.ros.org/Sensors
  - Ejemplos: Lidares, Camaras 2D y 3D, IMU, GPS.
- Actuadores
  - https://wiki.ros.org/Motor%20Controller%20Drivers
  - Ejemplos: Robotis Dynamixel, VESC.

Motores Dynamixel

- Instalación del paquete

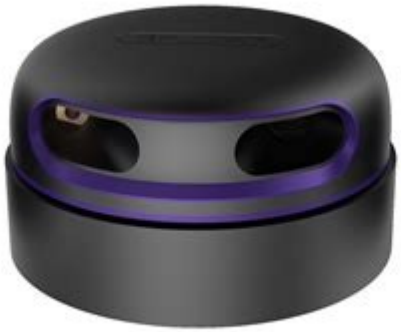
```
$ sudo apt-get install ros-$ROS_VERSION-dynamixel-motor
```

Sensor LIDAR

- Instalación del paquete

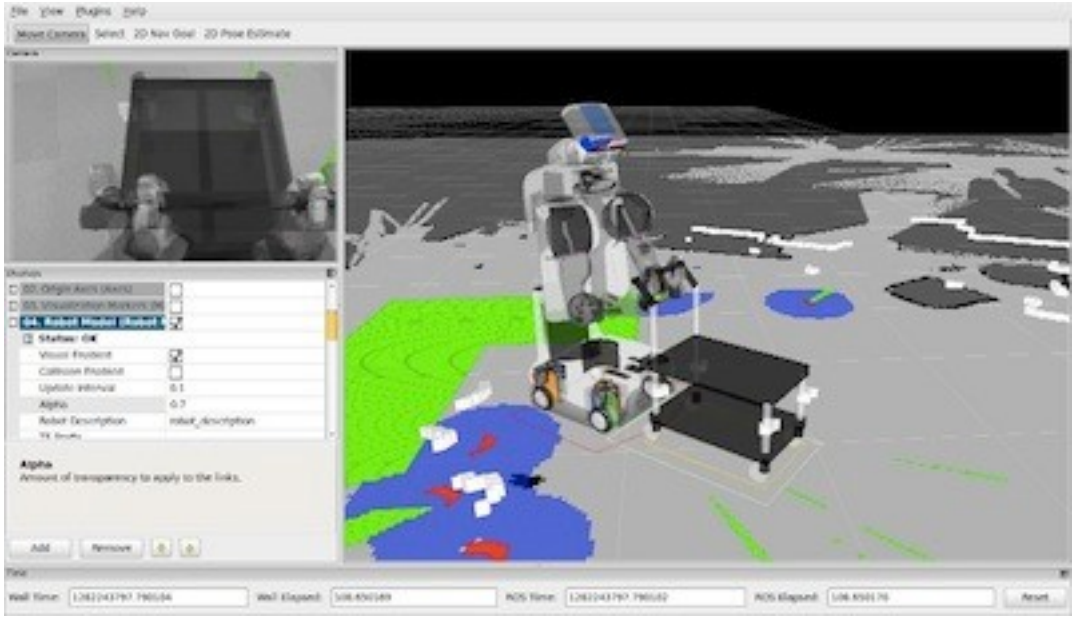
```
$ sudo apt-get install ros-$ROS_VERSION-rplidar-ros
```

```
$ sudo apt-get install ros-$ROS_VERSION-urg-node
```



Visualizador de ROS (rviz)

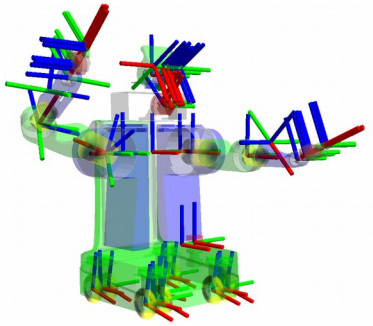
- Proporciona visualización 3D de sensores y robots (URDF).
- Permite visualizar la información en un sistema de coordenadas común.
- Herramienta imprescindible para debug.
- https://youtu.be/i--Sd4xH9ZE



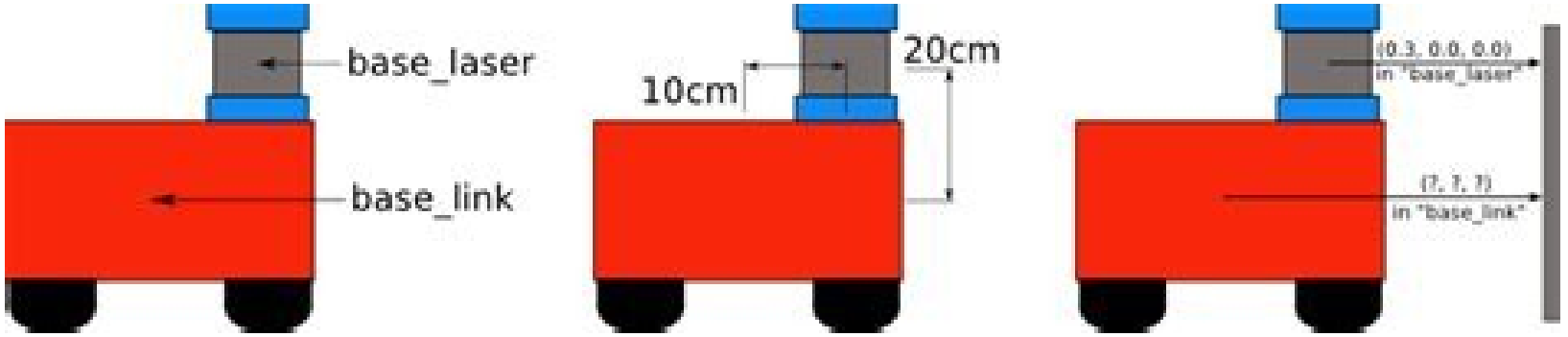
La familia rqt

- ROS utiliza un conjunto de herramientas generales que pueden componerse o configurarse a través de plugins para generar interfaces personalizadas.
- Algunas herramientas proporcionadas son:
  - rqt_console
  - rqt_graph
  - rqt_plot
  - rqt_topic
  - rqt_bag
  - ...

tf



- Un robot dispone de varios ejes de coordenada (p.e. mundo, base móvil, cabeza y pinza).
- La biblioteca tf fue diseñada para proporcionar una manera estándar mantener los ejes de coordenadas y de transformar información.
- Elementos principales: Broadcaster y Listener.
- Robustez (diferentes frecuencias, latencia y perdida de paquetes).



Tf demo

- Demo

```
$ roslaunch turtle_tf turtle_tf_demo.launch
```

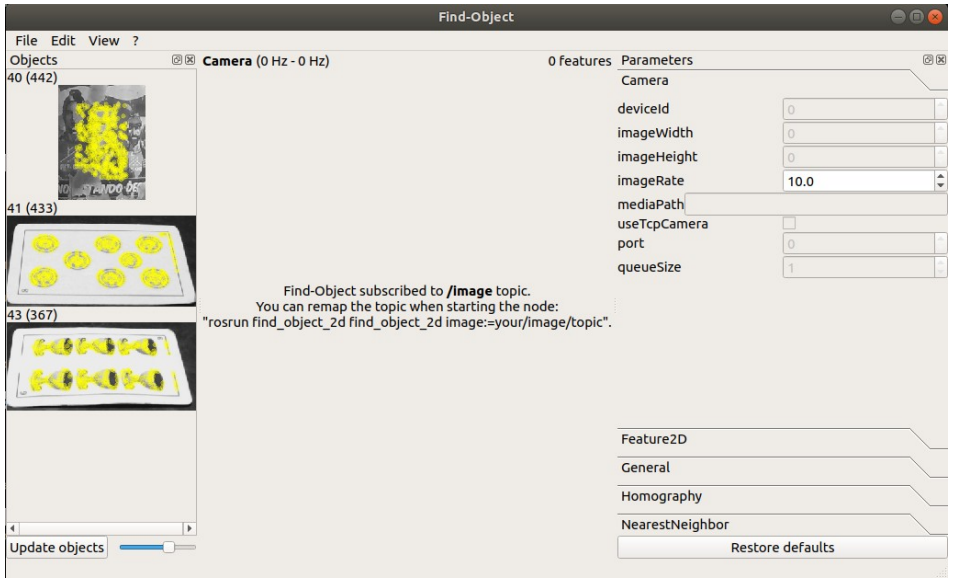
- El launch levanta:
  - turtlesim
  - dos tf broadcasters
  - tf listener
  - teleop

Paquete find object

- Paquete para experimentar con algoritmos de visión por computador.
- Dispone de un conjunto de algoritmos para evaluar y calibrar.

```
$ sudo apt-get install ros-$ROS_VERSION-find-object-2d
```

```
$ roslaunch find_object_2d find_object_2d_gui.launch
```



Objects to Turtle

```
def new_object(msg):
    cmd = Twist()
    cmd.linear.x = 1
    if len(msg.data)>0:
        turtle_controller.publish(cmd)

if __name__ == '__main__':
    rospy.init_node("object2turtle")

    turtle_controller = rospy.Publisher('/turtle1/cmd_vel', Twist)
    rospy.Subscriber("/objects", Float32MultiArray, new_object)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()
```

```
$ rosrun ...
```

rosbag

- **rosbag** permite grabar y reproducir mensajes publicados en los tópicos
- ¿Para qué sirve?
- Ejecutar el publicador y, en otra terminal, ejecutar los comandos:
  - rosbag record -a -o chatter
  - terminarlo luego de unos segundos (ctrl+c)
  - rosbag play chater*
  - ver la salida de rostopic echo /chatter

Demo Bag

- Grabo en un bagfile la información de tópicos de interes

```
$ rosbag record /objects -o OBJ
```

- Mostrar información grabada

```
$ rosbag play OBJ_ ...
```

Demo Integración

- Levanto un nodo de comportamiento

```
$ rosrun demos obj2turtle.py
```

- Usar rosbag para grabar y reproducir datos

```
$ rosbag record /objects …
…
Ctrl+C
…
$ rosbag play filename
```

Bag to Turtle

```
$ rosnode list
$ rosnode kill …
$ rosbag play filename
```

Recapitulando

- ROS: plataforma para la programación de agentes robóticos utilizando pequeños programas desacoplados.
- ROS promueve la reutilización de software, la abstracción, la reproducibilidad de experimentos y un mejor ciclo de desarrollo
- Los nodos son la unidad de ejecución
- Estos pueden comunicarse mediante mensajes enviados a un tópico
- Los nodos se organizan en paquetes, y los paquetes en meta-paquetes o stacks

Recapitulando

- Un nodo puede programarse en python o c++
- Luego de llamadas a funciones de inicialización, se puede:
  - Publicar en un tópico de forma periódica
  - Establecer una función “callback” para el manejo de nuevos mensajes
- Herramientas:
  - **rostopic** nos permite analizar los tópicos actuales
  - **rosbag** nos permite grabar y reproducir datos
  - **roslaunch** nos permite lanzar varios nodos con un solo comando

Referencias

- ROS, ros.org.
- http://wiki.ros.org/ROS/Tutorials
- http://answers.ros.org/questions/ (recomendable usar google como motor de busqueda)

Preguntas

¿?