Robot Operating System (ROS)

Robótica

Alberto Díaz y Raúl Lara

Curso 2022/2023

Departamento de Sistemas Informáticos

¿Qué es *Robot Operating System* (ROS)?

Pues, aunque se denomine *Robot Operating System*:

- Ni es un sistema operativo, ni es exclusivo para robots

Es un **framework** y un **middleware** para desarrollar aplicaciones distribuidas

- **Framework**: Establece las prácticas y conceptos con los que trabajar
- **Middleware**: Sirve de intermediario de comunicación entre componentes
- Es *Open Software*, licenciado bajo la BSD 3-Clause

Incluye además un sistema de gestión de paquetes para desarrollar y desplegar software con facilidad

- En C++ y Python

¿Para qué? Podemos programarlos desde cero

Claro que sí, pero una vez tenemos el hardware:

- Hay que desarrollar drivers para cada uno de los sensores y actuadores
- Hay que desarrollar el framework de comunicaciones
    - Que soporte, además los diferentes protocolos de diferentes hardwares
- Escribir también el código asociado a la percepción
- Si es móvil, también los algoritmos de navegación y path planning
- Ojo, no olvidemos tampoco el mecanismo para sacar los *logs*
- Ah, y la gestión de errores

Ya entiendo, *no reinventar la rueda*

Exacto; tradicionalmente el desarrollo de un robot era una tarea muy tediosa

- En esencia se construían desde cero prácticamente todos sus componentes

Con ROS se intenta minimizar ese efecto de reinventar la rueda; para ello:

- Se incluyen múltiples librerías de componentes de uso típico
- Se ofrece una infraestructura de comunicación *language agnostic*
    - ¡Incluso entre componentes de una misma aplicación!

Versiones

En la actualidad coexisten dos versiones independientes en desarrollo

1. ROS, la versión original
    - Bastante extendida, aunque en desuso
2. ROS2, la sucesora
    - Soporte desde 0 para Python 3.X
    - Nuevas funcionalidades y mejoras en la funcionalidades existentes

¿Cuál debemos usar?

- ROS2 siempre que sea posible
- Cuando no, intentar migrar la aplicación existente a ROS2, y entonces ROS2

Instalación de ROS2

¿Qué distribución elegir?

La lista se encuentra en https://index.ros.org/doc/ros2/Releases/

- Órden alfabético $\equiv$ órden cronológico
- Para elegir (si el proyecto no depende de una versión en concreto):
  - Comprobar la *End of Life* (EOL)
  - Comprobar si es *Long Term Support*
  - Comprobar el sistema operativo sobre el que funciona

Nosotros usaremos **Humble Hawksbill** sobre **Ubuntu GNU/Linux 22.04**

- **Una máquina virtual sería una buena tarea opcional...**

Nodos y la CLI

Antes de nada

Las aplicaciones en ROS se componen de nodos principalmente

- Se puede pensar en ellos como **procesos independientes**
- Se agrupan en **paquetes**

¿Paquetes?

- **Componentes** de nuestro programa; incluyen los fuentes de este
- Se encuentran en el directorio de instalación de ROS o en nuestro *workspace*

¿*Workspace*?

- Espacio de trabajo (**directorio**) con las aplicaciones a ejecutar

Creación de un espacio de trabajo

El desarrollo suele ser un proceso tedioso, porque implica muchas tareas:

- Crear y gestionar paquetes
- Gestionar las dependencias de componentes
- Compilar paquetes
- Desplegar

Ójala existiese una herramienta para gestionar los espacios de trabajo

`colcon`

Herramienta para la gestión de los espacios de trabajo

- Está creada específicamente para ROS2
- Pero no viene instalada por defecto

Instalación (como superusuario)

```
$ apt install python3-colcon-common-extensions
```

Para habilitar el autocompletado (recomendable añadir al `~/.bashrc`)

```
$ source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

Ahora sí, creación de un espacio de trabajo

Los pasos a realizar son los siguientes:

1. Creamos un directorio para nuestro *workspace* (e.g. bajo `$HOME/ros_ws`):
2. Accedemos al espacio de trabajo y creamos un nuevo directorio llamado `src`:
   - Aquí se almacenará todo el código fuente de nuestros componentes
3. Creamos nuestro espacio de trabajo, usando la herramienta `colcon`

   ```
   $ colcon build
   ```

   - Creará directorios `install/` y `logs/` si no existen
   - También los ficheros de configuración de *workspace* si no existen
   - Construirá todos los componentes (paquetes) de nuestra aplicación
4. Cargamos `setup.bash` del directorio `install/`, creado tras `build`:
   - Suele ser útil añadirlo al `~/.bashrc`

Creación de un paquete

Los pasos a realizar son los siguientes

1. Vamos al directorio `src/` de nuestro *workspace*
2. Ejecutamos el comando para la creación de paquetes

```
$ ros2 pkg create super_pkg --build-type ament_python --dependencies rclpy
```

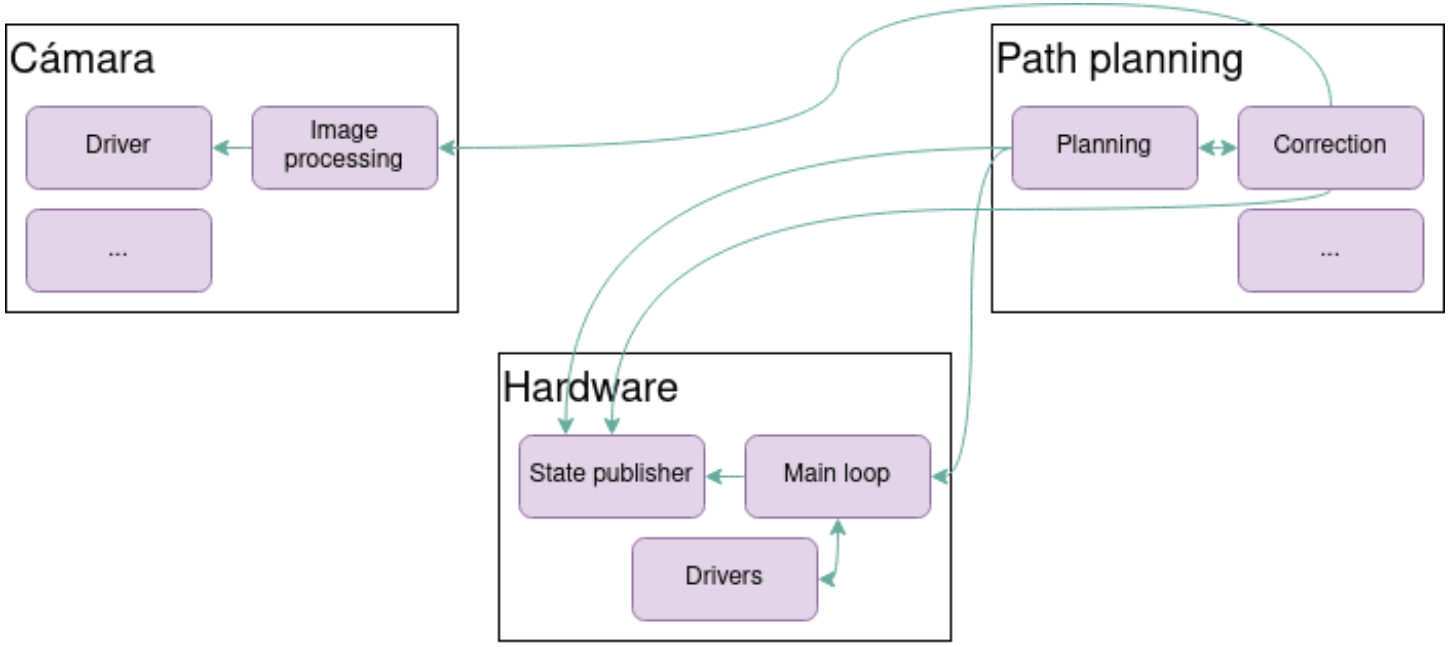
   - Esto creará un paquete llamado `super_pkg` ...
   - ... usando el sistema *ament* para la creación de paquetes ...
   - ... de tipo `python`
   - ... dependiente de la librería `rclpy`

`rclpy` es la librería base de ROS y la usaremos prácticamente siempre

¿Y qué es un nodo?

Son el componente principal de nuestras aplicaciones

- Un único nodo debería tener (idealmente) un único propósito
- Se comunican entre sí a través de la infraestructura de mensajería de ROS



Fuentes de un nodo básico en Python

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node

def main(args=None):
    try:
        rclpy.init(args=args)
        node = Node('py_test')
        node.get_logger().info('Hello, 🤖!')
        rclpy.spin(node)
    finally:
        rclpy.shutdown()

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

Configuración de la instalación

Fichero `setup.cfg`

```
[develop]
script-dir=$base/lib/NOMBRE_DEL_PAQUETE
[install]
install-scripts=$base/lib/NOMBRE_DEL_PAQUETE
```

Fichero `setup.py`

```
from setuptools import setup
#...
setup(
    # ...
    entry_points={'console_scripts': ["exec_name = PAQUETE.NODO:main"],},
)
```

Plantilla de un nodo como clase

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__():
        super().__init__('py_test')
        self.get_logger().info('Initializing py_test')

def main(args=None):
    try:
        rclpy.init(args=args)
        node = MyNode()
        rclpy.spin(node)
    finally:
        rclpy.shutdown()

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

Un nodo que hace algo (pero poco)

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node

class MyNode(Node):

    def __init__(self):
        super().__init__("py_test")
        self.i = 0
        self.create_timer(0.5, self.timer_callback)

    def timer_callback(self):
        self.i += 1
        self.get_logger().info("Hello, world! {self.i}")

def main(args=None):
    try:
        rclpy.init(args=args)
        node = MyNode()
        rclpy.spin(node)
    finally:
        rclpy.shutdown()

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

Recapitulando

Hemos visto qué son espacios de trabajo, paquetes y nodos

- Sabemos crear el espacio de trabajo de nuestro robot (`colcon`)
- Sabemos construir paquetes que contendrán el software de nuestra aplicación

Los nodos son subprogramas existentes dentro de nuestra aplicación

- Cada uno es **responsable de una y solo una funcionalidad**
- Se comunican a través de *topics*, *servicios* y *parámetros* (para que haga poso)

Los nombres del fuente, el instalado y el nodo no tienen por qué coincidir

- Sí, hemos aprendido también a instalar los paquetes
- Y a lanzarlos

```
$ ros2 run <paquete> <executable>
```

Topics y mensajes

Servicios

Launchers

Parametros

¡GRACIAS!