Introducción a ROS2

Fundamentos de Robótica Autónoma

MINA - Facultad de Ingeniería - Udelar

Contenido

1. Qué es ROS2
2. Modelo de Software
3. Modelado del Robot
4. Comunicaciones
5. Funcionalidades
6. Despliegue

Qué es ROS2

Qué es ROS2

- Robotic Operating System (ROS)
  - Es un ambiente de desarrollo para robótica que promueve la reutilización 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).
  - ROS2 disponible desde 2018 (ROS1 EOL: 2025)

Qué es ROS2

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, “accounting”...
- Provee abstracciones para interactuar con el robot

Qué es ROS2

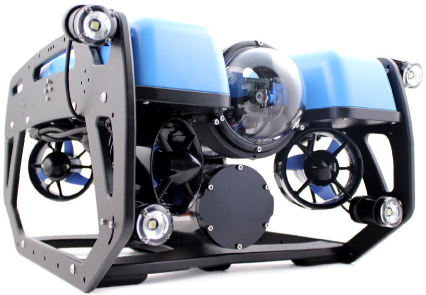
Ventajas:

- Reutilización de software
  - Funcionalidades complejas o tediosas
  - Interoperación y estándares
- Reproducibilidad
- Especialización
- Organiza dependencias y despliegue
- Formaliza ciclo de desarrollo
  - Grabar - implementar - simular - desplegar
- Versionado

Desventajas

- Resolución de dependencias trae “*bloat*”
- Gran número de cosas que pueden fallar
- Fuera de sistemas “tier 1” (Ubuntu, x86, ARM64) todo se complica
- Curva de aprendizaje: empezar es dificil (cuidado con documentación de ROS1!)
- Excesos de abstracción traen ineficiencias
- Versionado

Ejemplos de Robots que usan ROS2



Blue Robotics

Freefly

HIWONDER

Mitsubishi

Modelo de software

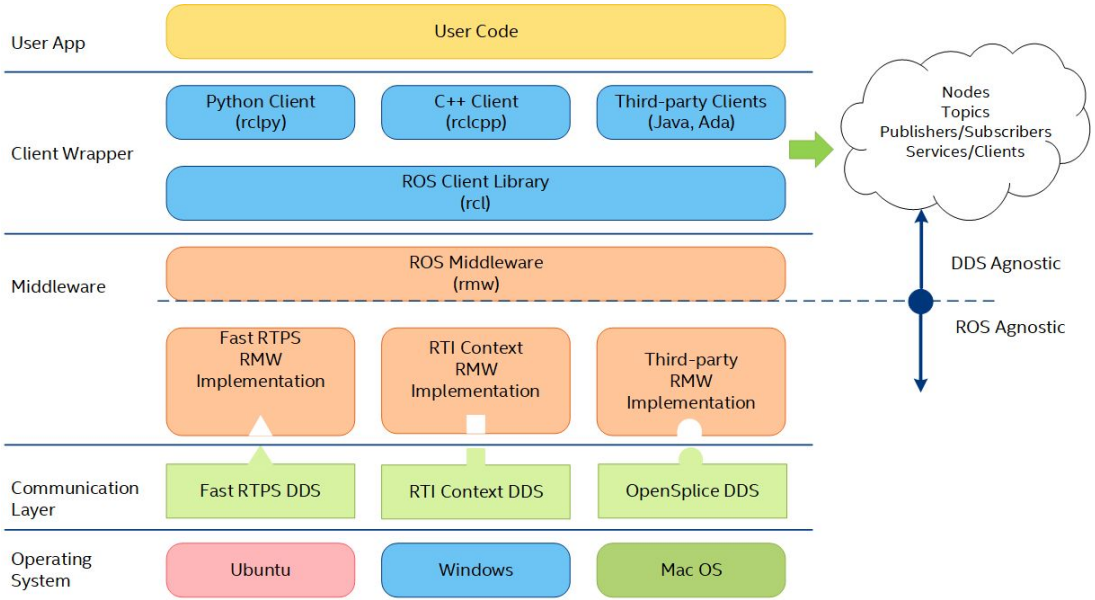
Usuarios de ROS2

Ejemplos de usuarios de ROS2

- Desarrollador de Hardware: crea un brazo articulado, quiere que sea fácil de integrar en las aplicaciones de sus clientes
- Desarrollador de software: crea un nuevo sistema de combina información de localización de múltiples fuentes y genera una estimación de la ubicación real. Quiere que sea fácil incorporar datos de nuevos sensores
- Integrador: está creando un robot para una aplicación novedosa en el agro. Quiere combinar una plataforma robótica comercial con unos sensores comprados y otros de desarrollo propio, y además necesita un sistema de localización confiable

Sistema distribuido

- Nodos: procesos que linkean biblioteca de ROS2
  - Lenguajes nativos: C, C++, Python.
- Comunicación entre procesos:
  - Abstraída y distribuida en la red (RMW)
  - Cada nodo tiene un nombre
- Nodos se comunican con otros en el mismo host, mismo robot, u otros robots
- Los nodos puede ser configurados (*parameters*) remotamente



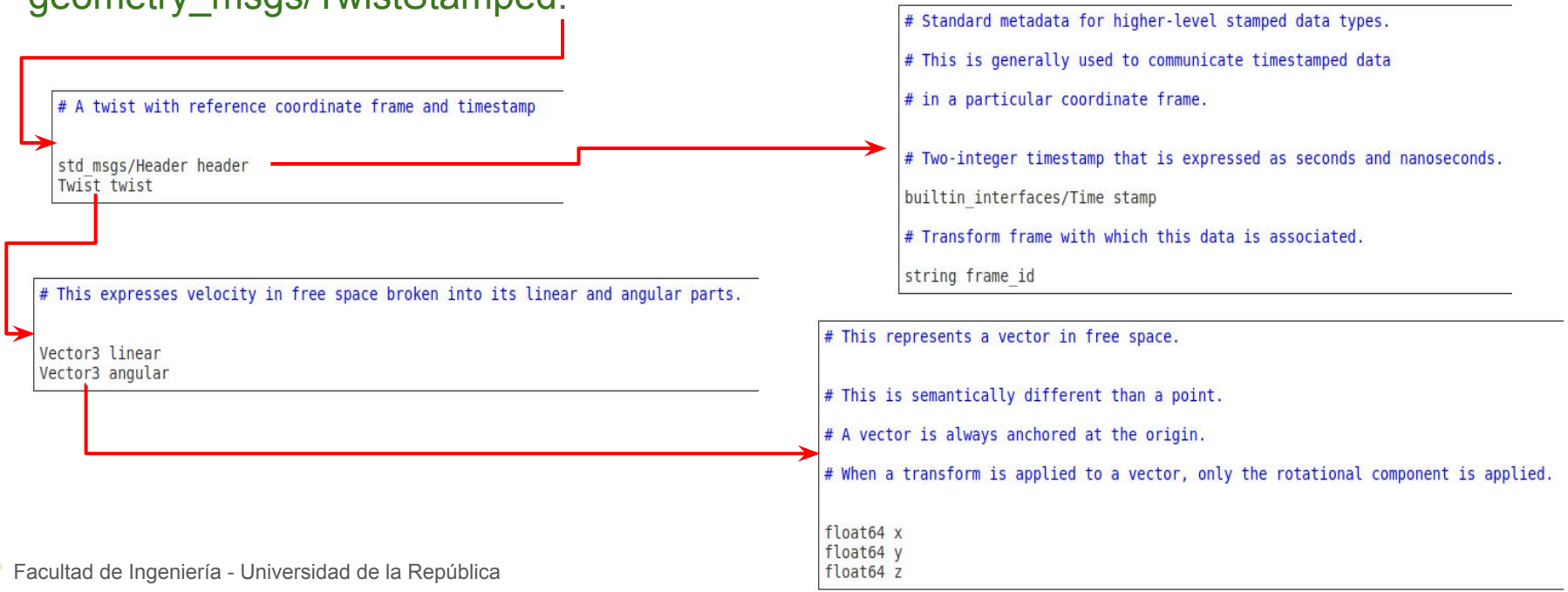
Sistema distribuido: nodos

Tutorial francesca

- `ros2 node list`

Sistema distribuido

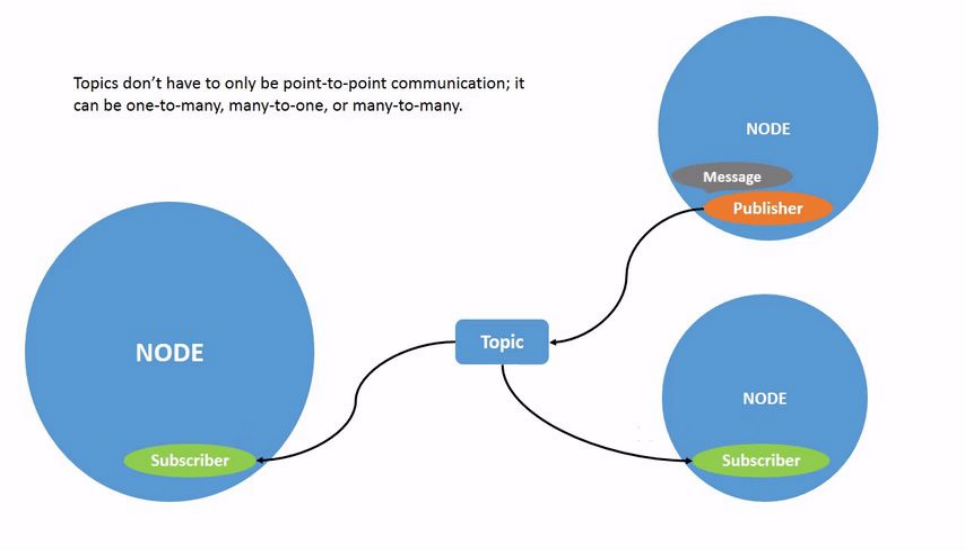
Las comunicaciones son fuertemente tipadas. Podemos definir tipos o usar los pre-definidos. Ejemplo: los comandos de movimiento, se declaran de tipo geometry_msgs/TwistStamped:



Sistema distribuido: tópicos

Topics

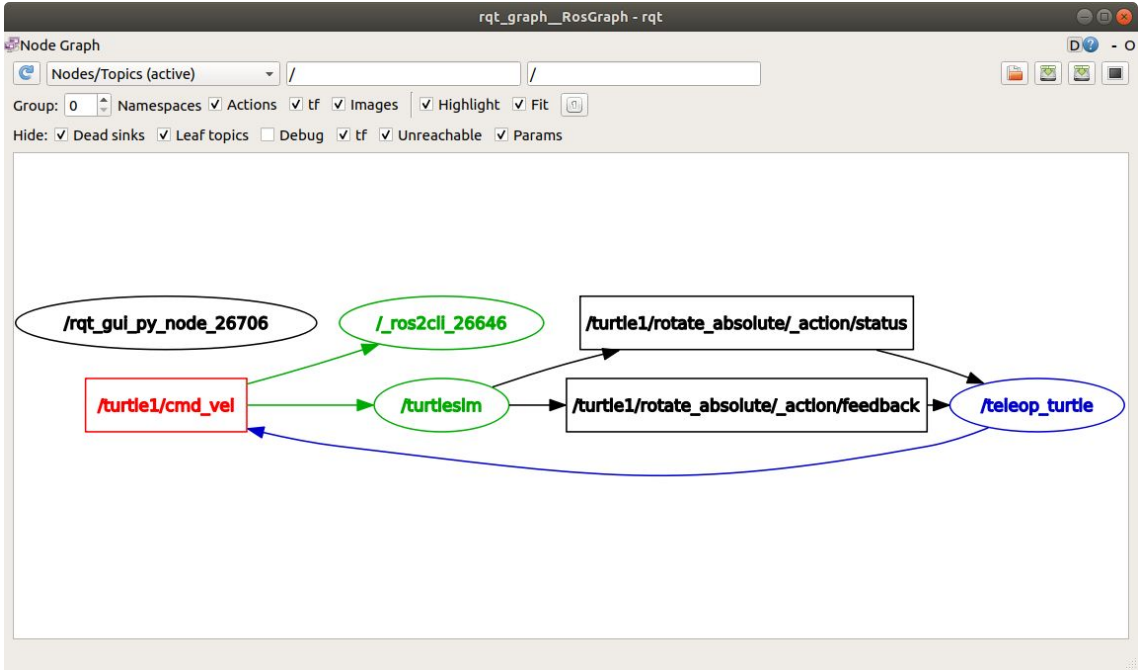
- Mensajería publish-subscribe
- Identificados por un nombre, e.g. /imu/raw
- Tiene un tipo asociado
- Soporta varios modos QoS
  - Reliability
  - Durability
  - Lifespan…



> https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools.html

Sistema distribuido: tópicos

```
$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
$ ros2 topic echo /turtle1/cmd_vel
Linear:
  x: 2.0
  y: 0.0
  z: 0.0
Angular:
  x: 0.0
  y: 0.0
  z: 0.0
^C
$ ros2 topic hz /turtle1/cmd_vel
average rate: 1.000
min: 1.000s max: 1.000s std dev:
0.00001s window: 2
```



Sistema distribuido: tópicos

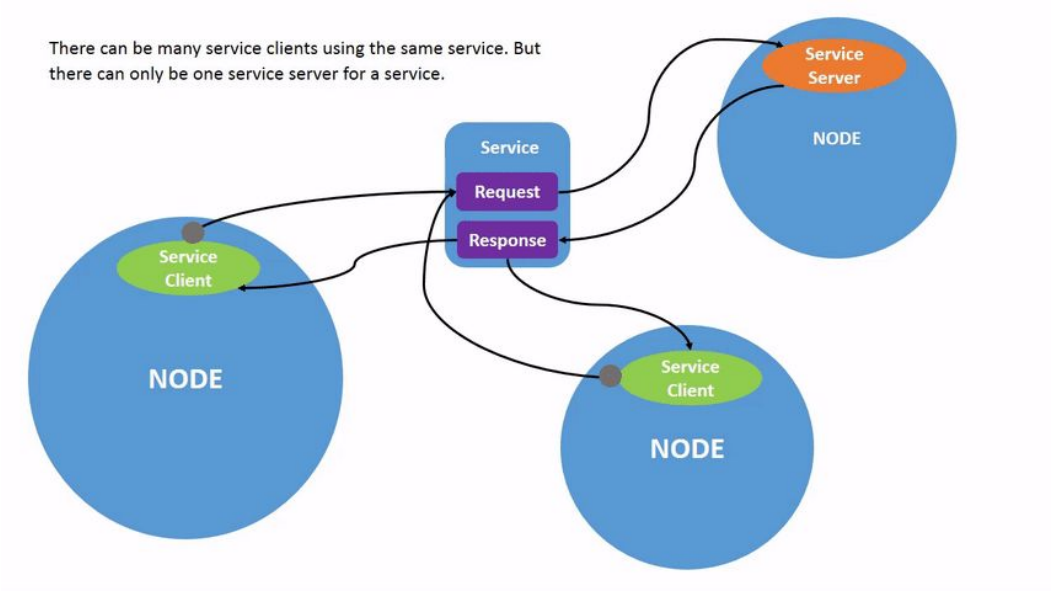
Tutorial francesca

- `ros2 topic list`
- `ros2 topic list -t`
- `ros2 topic hz …`
- `ros2 topic echo …`
- `rqt_graph`

Sistema distribuido: servicios

Service

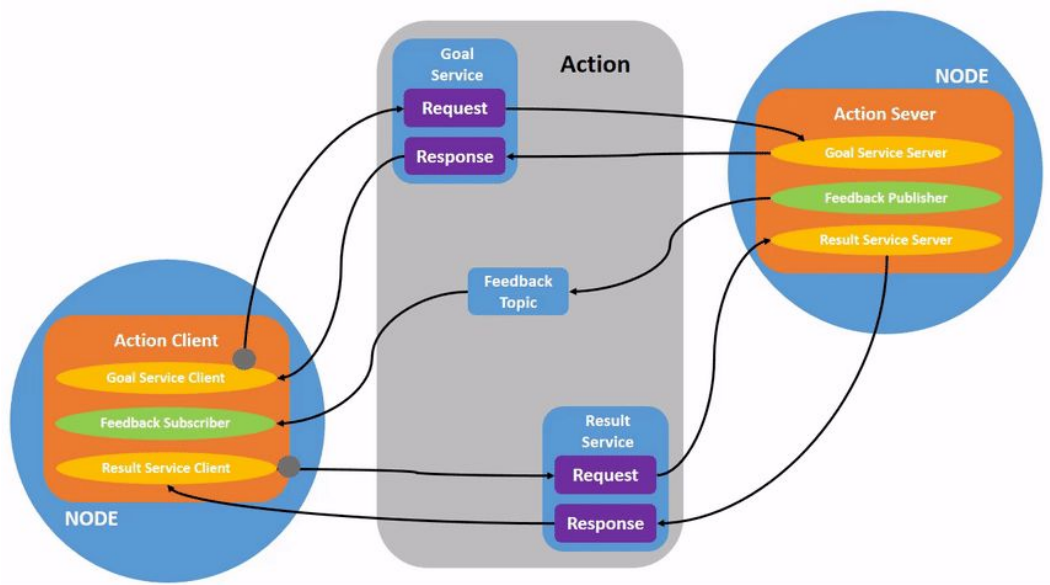
- Call-response
- Identificados por un nombre, e.g. /robot/reset
- Tiene parámetros de entrada y salida tipados



Sistema distribuido: acciones

Actions

- Procesos *long-running*
- Durante la ejecución informa de estado mediante tópicos
- Tiene parámetros de entrada y salida tipados
- Puede ser cancelado



Modelo del Robot

Modelo del Robot

Los nodos para interactuar pueden depender de visión unificada del robot:

- ¿Qué se supone tengo instalado?
- ¿Dónde están ubicados los sensores?
- ¿Cuántas articulaciones tiene el brazo?
- ¿Cómo está orientado el robot en el espacio?

ROS2 se basa en dos subsistemas relacionados para el modelado:

- Sistema de transformaciones y sistemas de coordenadas (“marcos de referencia”) en el espacio: tf2
- Formato de representación del robot: URDF / SDF

Transform System: tf2

Cuando describimos un robot hay muchos marcos de referencia involucrados:

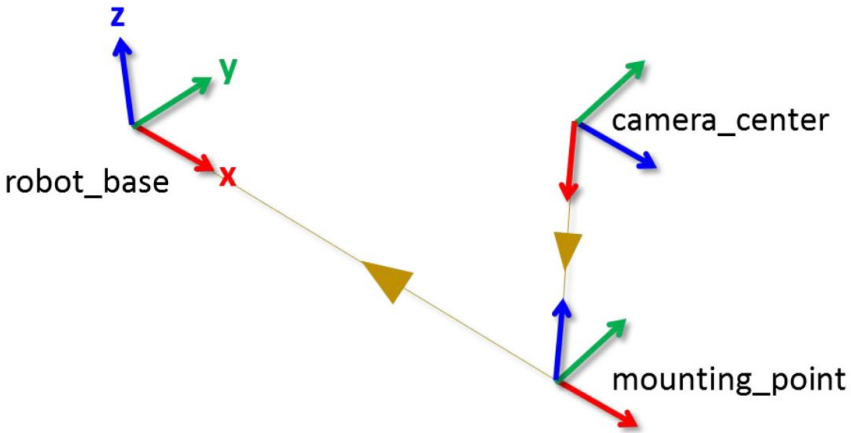
- El ángulo que giró una rueda: respecto a su eje (*rad*)
- La dirección en la que apuntan las ruedas: respecto al robot (*rad*)
- La ubicación de la cámara: respecto al robot (*m*)
- Cuánto el robot cree que se movió (odometría, imu): respecto a si mismo? (*m*)
- La ubicación del robot: uno o más mapas globales, de varias escalas (*m*, *lat/long*)
- Posición de un píxel en una imagen: respecto a la imagen (*pixels*)

> https://www.ros.org/reps/rep-0103.html
> https://www.ros.org/reps/rep-0105.html
> https://articulatedrobotics.xyz/tutorials/ready-for-ros/tf/

Transform System: tf2

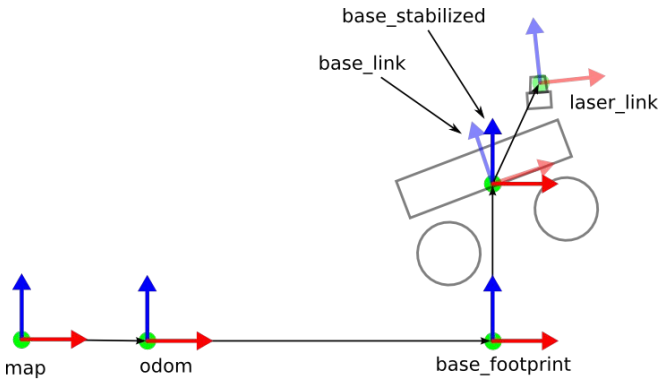
ROS2 organiza los sistemas de coordenadas (*frames*) vinculándolos mediante transformaciones

- Un nombre para un sistema de ejes en el espacio
- Una transformada desde otro marco de referencia (arborescente)



Transform System: tf2

Localización de un robot:



Arquitectura de un robot humanoide

map and odom

See REP 105 [1] for `map` and `odom` frames definition.

Frame Hierarchy

- **base_link**
  - base_footprint
  - ... - (l|r)_ankle - (l_r)_sole - (l|r)_toe
  - **... - torso**
    - ... - gaze
    - ... - (l|r)_wrist - (l|r)_gripper

> https://www.ros.org/reps/rep-0105.html
> https://www.ros.org/reps/rep-0120.html
> http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot

Transform System: tf2

Tutorial francesca

- Frames como tópicos:
  - Broadcast / listens
  - `/tf` y `/tf_static`

    `ros2 run tf2_ros tf2_echo [source_frame] [target_frame]`
- `ros2 run tf2_tools view_frames`
- `rviz`

Unified Robot Description Format (URDF)

Archivo que describe la geometría y organización del robot

- XML + xacro (macros)
- Centraliza todo lo que sabemos del robot
- Puede contener información necesaria para simular (forma, momentos de inercia)
- Arbol de *links* unidos por *joints* (si, como *frames* y *transforms*)
- El fabricantes de un robots provee su URDF
- Sólo con el URDF ya podemos *hacer cosas*

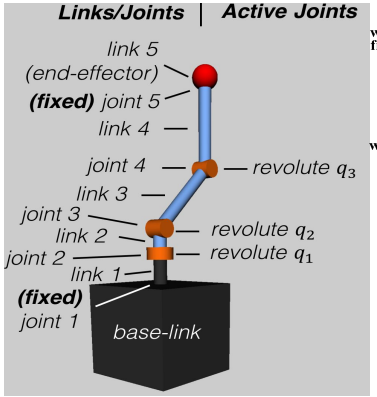
Unified Robot Description Format (URDF)

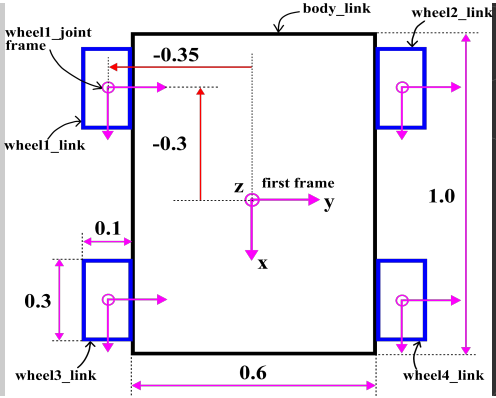
Link: una parte del robot:

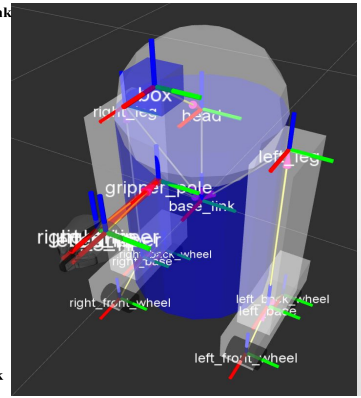
- Tiene un *frame* propio asociado
- Parte física: Rueda, Chassis, Cámara, Antebrazo
- Referencia útil: Posición de una IMU, *base_footprint*

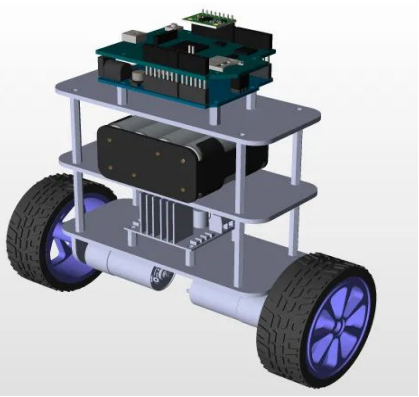
Joint: conecta dos Links

- Contiene una *transform* (por lo tanto implica un arbol de *Links*)
- Uno de varios tipos: *fixed*, *revolute*, *continuous*, *prismatic*, *floating*, *planar*









Unified Robot Description Format (URDF)

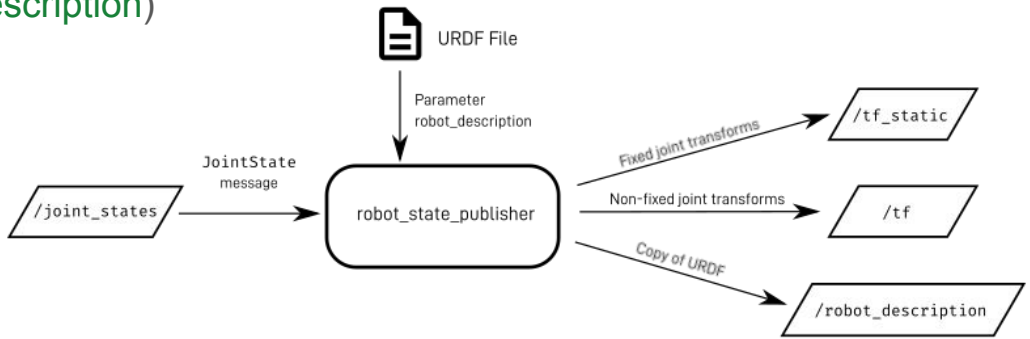
Tutorial francesca

- Mirar urdf y xacro
- rviz

Unified Robot Description Format (URDF)

Los joint móviles (todos excepto *fixed*) tienen grados de libertad. De donde se fija?

- La configuración de los *joints* se publica en el tópico /joint_states
- ¿QUIÉN HACE QUÉ?
- El nodo robot_state_publisher:
  - Lee el URDF (parámetro robot_description)
  - Escucha /joint_states
  - Publica /tf y /robot_description

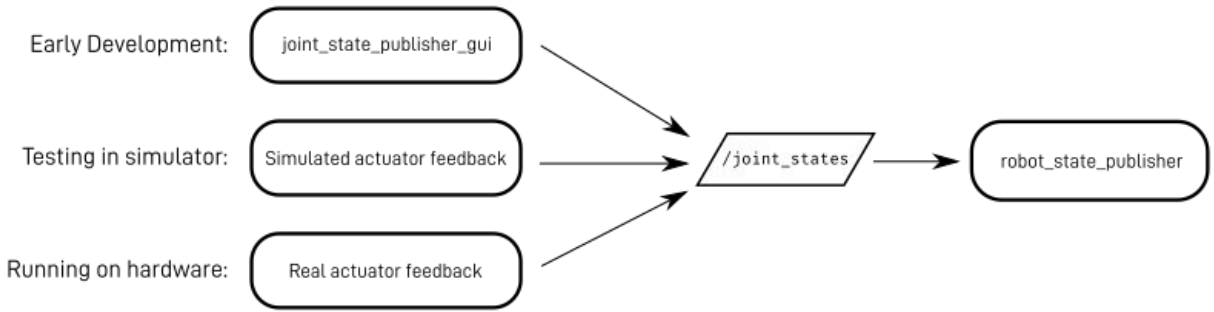


> https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/

Unified Robot Description Format (URDF)

Tutorial francesca

- Rviz + `joint_state_publisher`
- Discutir simulación
- Discutir encoders



Comunicaciones

Implementaciones de RMW

Las comunicaciones en ROS1 eran cliente-servidor. ROS2 rediseño para mayor flexibilidad:

- ROS Middleware interface (RMW), una abstracción de las comunicaciones entre procesos. Implementaciones de RMW:
  - Sobre Data Distribution Service (DDS)
  - Zenoh
- Adaptadores a Data Distribution Service (DDS)
  - Middleware definido por Object-Management Group
  - Completamente distribuido, comunicaciones peer-to-peer
  - Descubrimiento automático de participantes (basado en UDP multicast)
  - Fácil de usar out-of-the-box, problemas de escalabilidad y flexibilidad
- Zenoh
  - Mucho más flexible, permite topologías interesantes.

Múltiples robots en la misma red

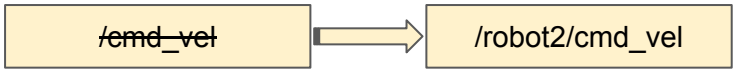
Varios nodos compartiendo una red pueden usar tópicos y servicios que se llamen igual. ¿Cómo separarlos?

Namespaces

Agregar un identificador al inicio de cada nombre de tópico:

```
$ ros2 run <package> <node> --ros-args --remap __ns:=/robot1
```


```
$ ros2 run <package> <node> --ros-args --remap __ns:=/robot2
```



Domains

Atributo manejado internamente por ROS2. Se lee desde una variable de ambiente:

```
$ export ROS_DOMAIN_ID=1
$ ros2 run <package> <node>
```

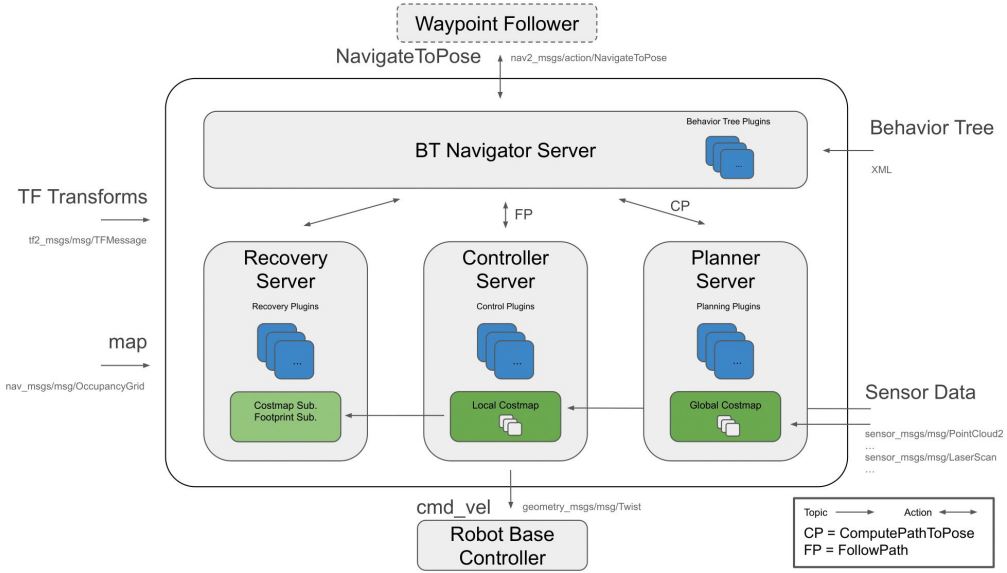
```
$ EXPORT ROS_DOMAIN_ID=2
$ ros2 run <package> <node>
```

Funcionalidades

Funcionalidades

Nav2: Navegación

- Provee
  - SLAM
  - Planificadores
  - Gestión de fallos
  - Optimización y seguimiento de trayectorias
  - Control de *ciclo de vida*
- Basado en
  - *Action Servers*
  - *Behavior Trees*
  - *Costmaps*

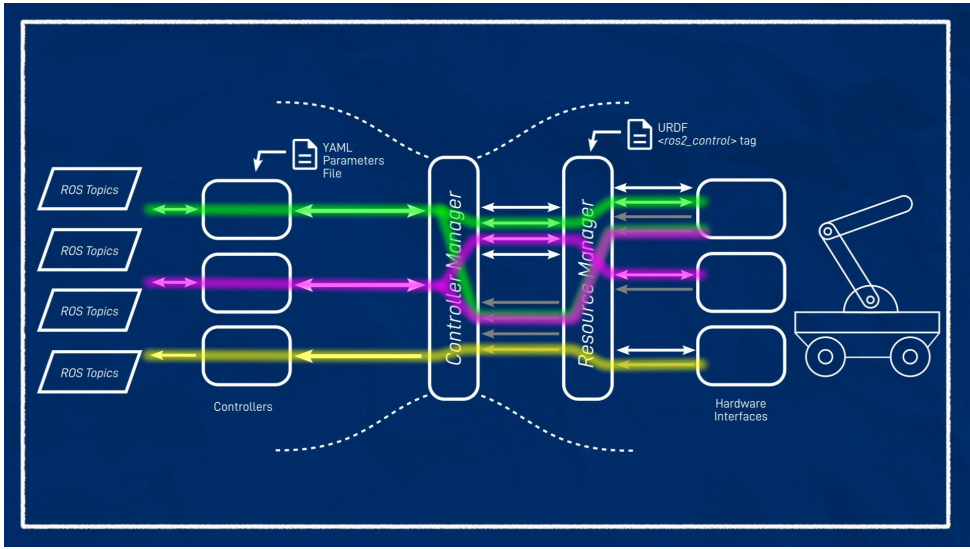


> https://docs.nav2.org

Funcionalidades

Arquitectura de control de tiempo real: Ros2_control

- La separación controlador/hardware ya es implementable con tópicos, pero latencia o ancho de banda puede ser un problema en algunos problemas de control (e.g. PID a 500Hz)
- Ros2_control introduce el concepto de *hardware interface*, que vive *dentro* de un nodo (ros2_control_node)
  - Por ejemplo GPIO, I2C, CAN…
  - Sirve tanto para actuadores como sensores
  - Se define en el URDF + archivos de configuración



> https://control.ros.org

> https://articulatedrobotics.xyz/tutorials/mobile-robot/applications/ros2_control-concepts

Funcionalidades

Integración con simuladores

- Se puede correr la aplicación contra un simulador: *gz*
  - El entorno es provisto por el simulador
  - Fabricantes de sensores y actuadores proveen la versión simulada de sus dispositivos, de idéntica interfaz que los reales
  - use_sim_time
- Otras facilidades relacionadas
  - *rosbag*: grabar toda la información de una ejecución para reproducirla después.
- Fundamental para el desarrollo de robots:
  - Grabar - implementar - simular - desplegar

Despliegue

Versionado

- Una nueva versión cada 12 meses.
  - Años pares: asociada a una distribución Ubuntu LTS, 5 años de soporte.
  - Años impares: asociada a Ubuntu LTS anterior, soporte 1.5 años.
- Historia reciente:
  - May 2021: **G**alactic Turtle: non-LTS release, supported for 1.5 years *(EOL)*
  - May 2022: **H**umble Turtle: LTS release, supported for 5 years
  - May 2023: **I**ron Turtle: non-LTS release, supported for 1.5 years *(EOL)*
  - May 2024: **J**azzy Turtle: LTS release, supported for 5 years ← estamos acá
  - May 2025: **K**ilted Turtle: non-LTS release, supported for 1.5 years
- Además hay una versión *Rolling* donde se hace el desarrollo.

> https://www.ros.org/reps/rep-2000.html

Soporte

Una plataforma se define como:

Versión de OS + Arquitectura + Implementatción de RMW.

- **Tier 1**: Soporte completo. Integración contínua, tests unitarios, etc. Bugs tienen prioridad.
  - Ejemplo para Jazzy: Ubuntu 24.04 + [amd64 o arm64] + [Fast-DDS o Cyclone-DDS] (zenoh va a ser Tier1 en Kilted)
- **Tier 2**: Pueden faltar compilados. Integración testeada ~semanalmente. Fuentes y resultados de tests disponibles. Bugs atacados “según disponibilidad”
  - Ejemplo para Jazzy: RHEL9
- **Tier 3**: Provista por la comunidad.
  - Ejemplo para Jazzy: arm32

> https://www.ros.org/reps/rep-2000.html

Instalación

Disponible como paquetes de Ubuntu. Ejemplo:

```
sudo apt install ros-jazzy-desktop ros-jazzy-camera-ros
```

Variantes:

- `ros_core`: soporte de comunicaciones, nada de GUI
- `ros_base`: `ros_core`+bibliotecas comunes, nada de GUI
- `desktop` (recomendado): `ros_base`+visualización y tutoriales
- `perception`: incluye bibliotecas de percepción
- `simulation`: Integración con *gz*
- `desktop_full`: “batteries included” para completar tutoriales.

> https://www.ros.org/reps/rep-2001.html

Instalación

Cosas molestas del proceso de instalación:

- Infinitas dependencias, algunas misteriosas
- Atado a una versión de Ubuntu, con su configuración (permisos, udev, …)
- Todo se debe repetir en cada instancia de robot
- Los riesgos de actualizar algo

Buena práctica: usar virtualización, tipicamente Docker

- Crear y configurar una vez, levantar en todos lados

> https://docs.ros.org/en/jazzy/How-To-Guides/Setup-ROS-2-with-VSCode-and-Docker-Container.html

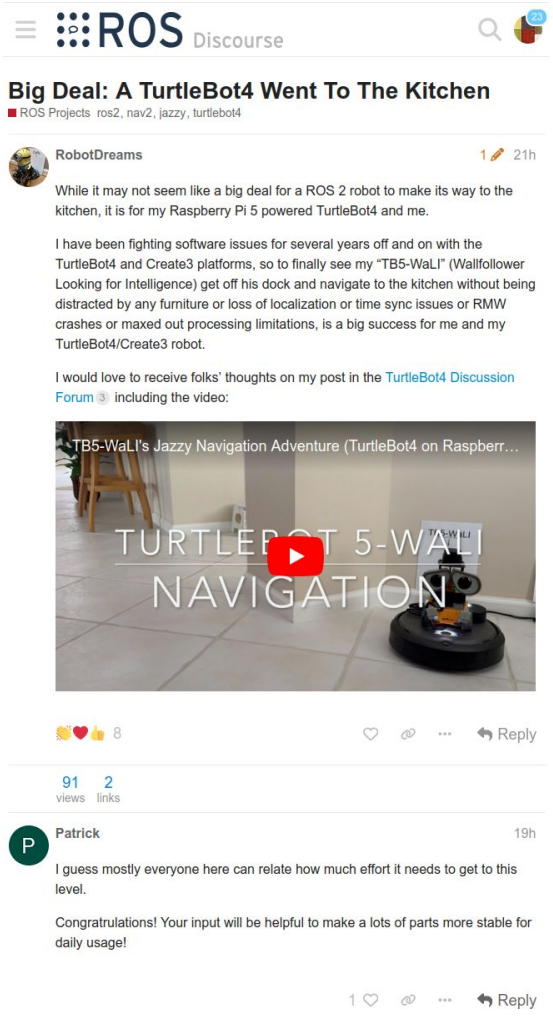
Estructura de un proyecto

```
├── LICENSE
├── README.md
└── src                                                The source code for the project
    └── francesca_pkg                                   The main package
        ├── bringup                                       Stuff necessary to start the system
        │   ├── CMakeLists.txt
        │   ├── config                                    Configuration files
        │   │   ├── francesca_controllers.yaml              ros2_control configuration file
        │   │   ├── francesca.rviz                          rviz configuration file
        │   │   ├── gz_bridge.yaml                          gz <-> ros2 translation specification
        │   │   └── joystick.yaml                           confiration used by the joystick.launch.py
        │   ├── launch                                    Launch files
        │   │   ├── francesca_gz.launch.py                  Start francesca inside simulator
        │   │   ├── francesca.launch.py                     Start real francesca
        │   │   └── joystick.launch.py                      Start a joystick remote control
        │   └── package.xml
        ├── description                                   Stuff describing the robot
        │   ├── CMakeLists.txt
        │   ├── models                                    URDF models
        │   │   └── francesca                               the robot
        │   │       ├── butia.xacro                           macro for generating the base
        │   │       ├── camera.xacro                          macro for generating the camera
        │   │       ├── dimensions_macros.xacro               macro storing the dimensions of stuff
        │   │       ├── francesca.urdf.xacro                  main urdf for francesca (also a macro)
        │   │       ├── inertial_macros.xacro                 macro with utility functions for intertia moments
        │   │       ├── lidar.xacro                           macro for generating the lidar
        │   │       ├── materials_macros.xacro                macro with utility functions for colors and stuff
        │   │       └── ros2_control.xacro                    macro for generating the ros_control description
        │   └── package.xml
        ├── gazebo                                        Gazebo configuration
        │   └── worlds                                    Contains the world specification files (SDFs)
        │       └── francesca_world.sdf                     a simple world with a box and a cylinder
        ├── LICENSE
        └── README.md
```

Cosas útiles

Cosas útiles

- https://docs.ros.org/en/jazzy/index.html
  Documentación principal
- https://robotics.stackexchange.com/
  Comunidad: preguntas
- https://discourse.ros.org/
  Comunidad: foros, discusiones
- https://articulatedrobotics.xyz/
  Blog, tutoriales



Fin