

MOC - ROS2 - Conceptos básicos (Programación y simulación de robots)

- relacionado
 - **ROS2 - Conceptos Adicionales** -> Ya linkeados en este documento
 - **CONECTAR AL ROBOT FÍSICO (ROS1)**
 - **_Archivado - Más cosas extra ROS**

Conceptos básicos

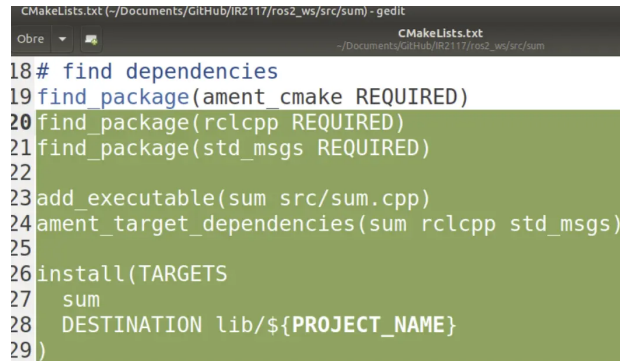
- **Sources**
 - `~/IR2117/directorio_ws`
 - `source /opt/ros/foxy/setup.bash`
 - `source install/setup.bash` (solo si ya se ha hecho el colcon build, para encontrar los paquetes compilados y programas listos para ejecutar)
 - `export ROS_LOCALHOST_ONLY=1`
 - Para no tener problemas entre ordenadores en espacios de trabajo compartidos (como clase). (por defecto 0)
- Otras preparaciones:
 - **Simuladores**
 - `rosdep install -i --from-path src --rosdistro foxy -y`
 - Verificar *dependencias*
- CREACIÓN PAQUETE
 - **sources**
 - `~/IR2117/directorio_ws/src`
 - `ros2 pkg create paquete --build-type ament_cmake --dependencies rclcpp`
 - Crear paquete
 - `--build-type`
 - `ament_python` indica que escribiremos los programas con python
 - `--dependencies`
 - automáticamente añade las dependencias a package.xml, para no hacerlo manualmente
 - **IMPORTANTE:** Si no usamos esto, pues añadimos en package.xml lo siguiente por ejemplo
 - `<exec_depend>rclpy</exec_depend>`
- **Compile**
 - `~/IR2117/directorio_ws`
 - **sources**
 - `colcon build --packages-select paquete --symlink-install`
 - `--packages-select paquete`

- `--symlink-install` -> Se usa para paquetes de python, permite modificar programas y ejecutarlos con los nuevos cambios sin tener que volver a compilar.
- `source install/setup.bash` para que el terminal encuentre los nuevos paquetes compilados

• CONFIG FILES

• CMakeLists.txt

- Como compilar y donde instalar el código



```

CMakeLists.txt (-/Documents/GitHub/IR2117/ros2_ws/src/sum) - gedit
Obre  CMakeLists.txt
~/Documents/GitHub/IR2117/ros2_ws/src/sum

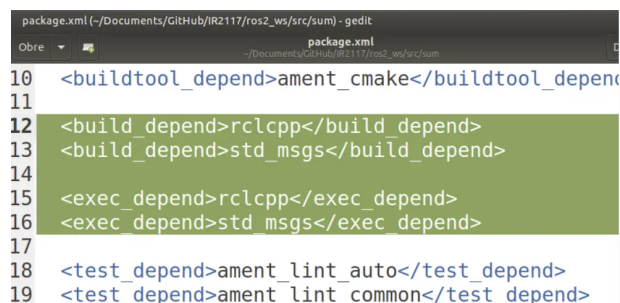
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 find_package(rclcpp REQUIRED)
21 find_package(std_msgs REQUIRED)
22
23 add_executable(sum src/sum.cpp)
24 ament_target_dependencies(sum rclcpp std_msgs)
25
26 install(TARGETS
27     sum
28     DESTINATION lib/${PROJECT_NAME})
29 )

```

- `ament_target_dependencies`:
 - Links a library with the executable
- `install`
 - 2 arguments, targets and destination
 - destination example
 - `ros2_ws/src/examples/install/examples/lib/examples`

• package.xml

- Contiene dependencias



```

package.xml (-/Documents/GitHub/IR2117/ros2_ws/src/sum) - gedit
Obre  package.xml
~/Documents/GitHub/IR2117/ros2_ws/src/sum

10 <buildtool_depend>ament_cmake</buildtool_depend>
11
12 <build_depend>rclcpp</build_depend>
13 <build_depend>std_msgs</build_depend>
14
15 <exec_depend>rclcpp</exec_depend>
16 <exec_depend>std_msgs</exec_depend>
17
18 <test_depend>ament_lint_auto</test_depend>
19 <test_depend>ament_lint_common</test_depend>

```

- Build: Dependencias para compilar
- Execution: Dependencias para correr el programa

• Including

- Main ROS client `#include "rclcpp/rclcpp.hpp"`
- INCLUDING messages
 - `#include "std_msgs/msg/string.hpp"`
 - `#include "std_msgs/msg/int32.hpp"`
 - `#include "std_msgs/msg/float32.hpp"`

• Launch Files

• makefile

• Example basic programs

- Initialization and shutdown
 - `rclcpp::init(argc, argv);`

- `rclcpp::shutdown();`
 - causes all nodes and their constituent parts to become invalid
 - `rclcpp::ok()`
 - test whether or not shutdown has been called
- Creating nodes
- ROS2 - Conceptos Adicionales > PUBLISHERS
- ROS2 - Conceptos Adicionales > SUBSCRIBERS
- Messages
 - Importing vs creating
 - `std_msgs/msg/string.hpp`
 - `std_msgs::msg::String mensaje_creado`
 - Editing data or using it
 - `mensaje_creado.data = "Texto"`
 - They also have smart pointer definitions
 - `std::shared_ptr<package_name::msg::Foo>`
- Running programs

run Allows to run an executable in an arbitrary package without having to 'cd' there first.

Usage:

```
$ ros2 run <package> <executable>
```

Example:

```
$ ros2 run demo_node_cpp talker
```

 - Argumentos
 - `myprog arg1 arg2 arg3 ---> argv[0] = myprog ---> argv[1] = arg1`
 - `int argc`: nº de parámetros
 - Bash

`char** argv/char* argv[]`: (Doble puntero o array de punteros) se puede iterar

`std::cin.eof()` puede servir para indicar cuando queremos parar de dar argumentos en terminal con ctrl+d.

`ls > fichero.txt` → la salida de ls te la escribe en un fichero (> redirección)

`ls | cat` → la salida de ls te la redirecciona a la siguiente instrucción (cat) (| = pipe = tubería)
- Tools + CLI
 - ros2bag
 - [Recording and playing back data — ROS 2 Documentation: Humble documentation](#)
 - `ros2 bag record /topic /other_topic /third_topic`
 - Guarda todos los datos que se publican en el topic /scan
 - `-o nombre` para escoger un nombre determinado
 - `ros2 bag info .../...db3`
 - Muestra información sobre la grabación (principio, fin, nº mensajes, etc.)
 - `ros2 bag play .../...db3`
 - Publica todos los mensajes en el topic como si se estuviera ejecutando el programa que hemos grabado
 - Teleop

- `ros2 run turtlesim turtle_teleop_key`
 - `ros2 launch turtlebot3_teleop turtlebot3_teleop_key.launch`
- **Gráfico** que muestra los nodos (círculos) y los topics (rectángulos) con flechas indicando las relaciones que hay entre ellos.
 - `rqt / rqt_graph`
- RViz
- **Remapping**
 - A topic: `--remap turtle1/cmd_vel:=turtle2/cmd_vel`
- **Tools for debugging** + Basics of Nodes, Topics, Services, Actions
 - **NODES CLI TOOLS**
 - `ros2 node list`
 - Lista de todos los *nodos* activos.
 - `ros2 node info <node_name>`
 - Muestra los subscribers, publishers, service servers, service clients, action servers y action clients de un nodo.ç
 - **TOPICS CLI TOOLS**
 - `ros2 topic list`
 - Lista de todos los topics *activos*.
 - `ros2 topic list -t`
 - will return the same list of topics, this time with the topic *type* appended in brackets
 - `/turtle1/cmd_vel [geometry_msgs/msg/Twist]`
 - `ros2 topic echo <topic_name>`
 - Muestra los *mensajes* que se están publicando en un topic en *tiempo real*
 - `ros2 topic info <topic_name>`
 - Muestra información sobre un topic (nombre, tipo de mensaje, publishers y subscribers).
 - `ros2 topic pub --once <topic_name> <msg_type> '<args>'` :
 - *Publica un mensaje* con unos argumentos en un topic.
 - “--once” para que solo se publique una vez
 - Ejemplos según tipos
 - `ros2 topic pub /number std_msgs/msg/Int32 "{data: '123'}"`
 - `ros2 topic pub /topic std_msgs/msg/String "{data: 'Hola'}"`
 - *RATE in Hz*
 - `ros2 topic pub --rate 1 /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"`
 - `ros2 topic hz <topic_name>`
 - Muestra cada cuánto se publican mensajes en un topic.
 - `ros2 interface show <msg type>`

- Muestra información sobre el *tipo de mensaje* de un publisher/subscriber/action/etc.
 - `ros2 interface show geometry_msgs/msg/Twist`
 - SERVICES CLI TOOLS
 - `ros2 service list`
 - Muestra todos los *servicios* activos.
 - `ros2 service call <service_name> <msg_type> <args>`
 - Llama a un servicio y le envía una request.
- ACTIONS
 - `ros2 action list -t`
 - Muestra todas las *acciones* del grafo. Añadir “-t” para ver el tipo de acción.
 - `ros2 action info <action_name>`
 - Muestra el nombre de la acción, sus clientes y sus servidores.
 - `ros2 run tf2_tools view_frames.py`
 - Muestra un diagrama de los marcos de referencia (frames) presentes en un sistema.
- ROS2 - Conceptos Adicionales > PUBLISHERS
- ROS2 - Conceptos Adicionales > SUBSCRIBERS
- ROS2 - Conceptos Adicionales > MESSAGES C
- Simuladores
 - RViz
 - `cd /home/diego/Documents/10.UJI/SimulacionRobots`
 - `rviz2 -d tb3.rviz`
 -
- Turtlesim
 - `source /opt/ros/foxy/setup.bash`
 - `export ROS_DOMAIN_ID=$N`

- `ros2 run turtlesim turtlesim_node`
- `ros2 param set /turtlesim background_r 255` : Pone el valor del parámetro `background_r` (red) del nodo `/turtlesim` a 255. También existen `background_g` y `background_b`.
- Gazebo
 - `source /opt/ros/foxy/setup.bash`
 - `export ROS_DOMAIN_ID=$N`
 - `export TURTLEBOT3_MODEL=burger` : Define el modelo de turtlebot3 como burger (también existe el waffle y el waffle_pi).
 - `ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py`
- Webots
 - `source /opt/ros/foxy/setup.bash`
 - `export ROS_DOMAIN_ID=$N`
 - `export WEBOTS_HOME=/home/usuario/webots-R2022b` : Define la versión de webots como la R2022b
 - `ros2 launch webots_ros2_turtlebot robot_launch.py`
- Turtlebot3 -> **CONECTAR AL ROBOT FÍSICO (ROS1)**
 - Parar robot:
 - `ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"`
 - Conectar:
 - ...
 - Desconectar:
 - `sudo halt -p`
 - Apagar placa
 - Quitar batería

Recursos

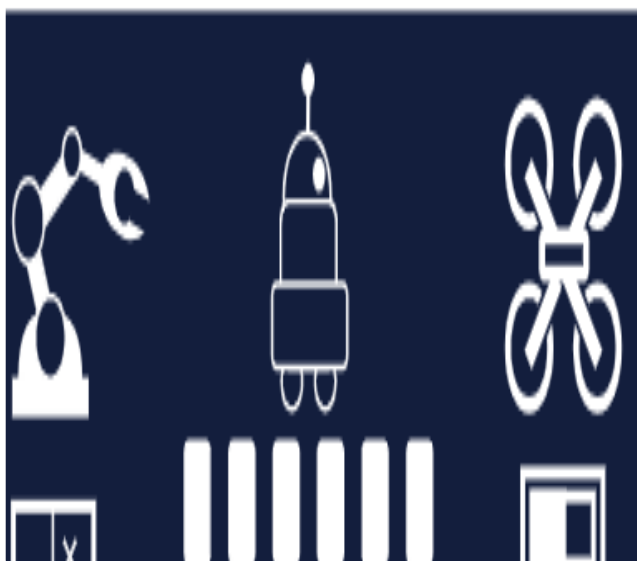
- RQT

- rclcpp (ROS Client Library for C++)
 - Guía muy completa sobre todo lo importable de ROS: [rclcpp: rclcpp: ROS Client Library for C++](#)

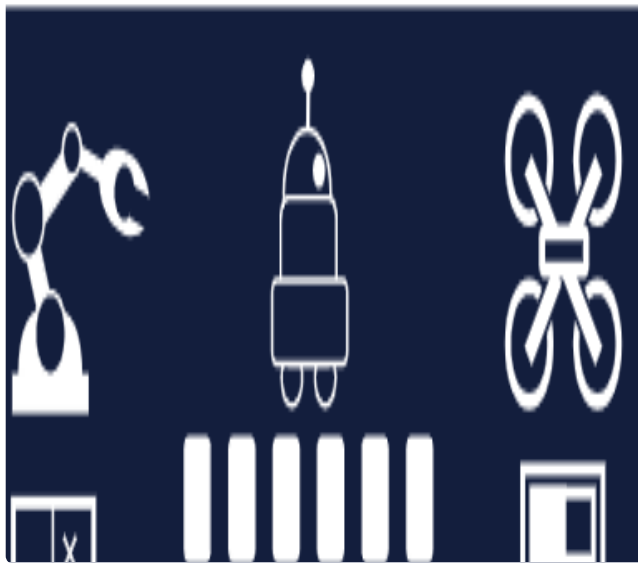
rclcpp provides the canonical C++ API for interacting with ROS. It consists of these main components:

- Node
 - `rclcpp::Node`
 - `rclcpp/node.hpp`
- Publisher
 - `rclcpp::Node::create_publisher()`
 - `rclcpp::Publisher`
 - `rclcpp::Publisher::publish()`
 - `rclcpp/publisher.hpp`
- Subscription
 - `rclcpp::Node::create_subscription()`
 - `rclcpp::Subscription`
 - `rclcpp/subscription.hpp`
- Service Client
 - `rclcpp::Node::create_client()`
 - `rclcpp::Client`
 - `rclcpp/client.hpp`
- Service Server
 - `rclcpp::Node::create_service()`
 - `rclcpp::Service`
 - `rclcpp/service.hpp`

Libros / PDF



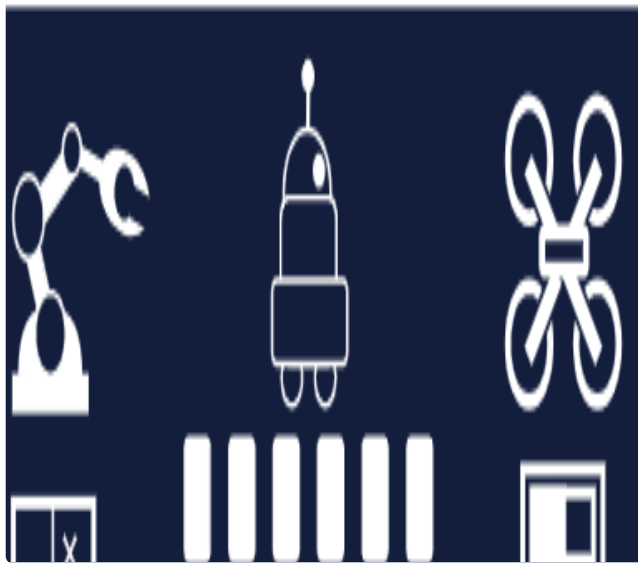
A Concise Introduction
to Robot Programming
in ROS2



A Concise Introduction to Robot Programming in ROS2



A Concise Introduction to Robot Programming in ROS2



A Concise Introduction to Robot Programming in ROS2
