`comando`
`usuario@equipo`
`~ruta actual`

`source /opt/ros/foxy/setup.bash` : Para tener acceso a los comandos de ROS (en este caso foxy). Solo hace falta ejecutarlo una vez en cada terminal.

`source install/setup.bash` : Después de compilar con colcon build se crea una carpeta “install” con un script (setup.bash) que contiene todas las variables de entorno necesarias para ejecutar los nodos del entorno de trabajo compilado. Ejecutando esta orden tenemos acceso a todas esas variables de entorno en el terminal que estemos utilizando.

`rosdep install -i --from-path src --rosdistro foxy -y` : Instala las dependencias que hayamos indicado que se necesitan en el directorio en el que estemos.

- -i : Indica que se instalarán todas las dependencias que se hayan indicado, incluso si ya se encuentran en el sistema.
- --from-path src : Indica la ruta donde se encuentran los paquetes de ROS2 que se desean analizar para determinar sus dependencias y realizar la instalación correspondiente. En nuestro caso, la carpeta src.
- --rosdistro foxy : Indica la versión de ROS2 que se está utilizando. En nuestro caso, foxy.
- -y : Indica que el proceso de instalación de dependencias se llevará a cabo sin solicitar confirmación al usuario.

`export ROS_DOMAIN_ID=$N` : Para no tener problemas entre ordenadores en espacios de trabajo compartidos (como clase). Si no se ejecuta se asigna automáticamente $N=0. Si tienen un ID diferente, no se pueden conectar entre sí los ordenadores.

`ros2 pkg create --build-type ament_cmake paquete --dependencies rclcpp` : Crea un nuevo paquete en ROS2 de nombre utilizando el sistema de construcción Ament con CMake. También añade las dependencias que se han indicado (rclcpp) a package.xml y CMakeLists.txt.

Debugging

`edgar@ubuntu:~$`
`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.

`ros2 topic list` : Lista de todos los topics activos.

`ros2 topic echo <topic_name>` : Muestra los mensajes que se están publicando en un topic.

`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. Poner "--once" para que solo se publique una vez.

`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.

`rqt / rqt_graph` : Gráfico que muestra los nodos (círculos) y los topics (rectángulos) con flechas indicando las relaciones que hay entre ellos.

`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.

`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.

Compilar

```
edgar@ubuntu:~/IR2117/directorio$
source /opt/ros/foxy/setup.bash
```

`colcon build` : Si queremos compilar el directorio entero.

`colcon build --packages-select paquete` : Si queremos compilar un paquete en específico dentro del directorio.

`source install/setup.bash`

Turtlesim

```
sudo apt update

sudo apt install ros-foxy-turtlesim

turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node

ros2 run turtlesim turtlesim_node

edgar@ubuntu:~$
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.

Webots

```
edgar@ubuntu:~$
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
```

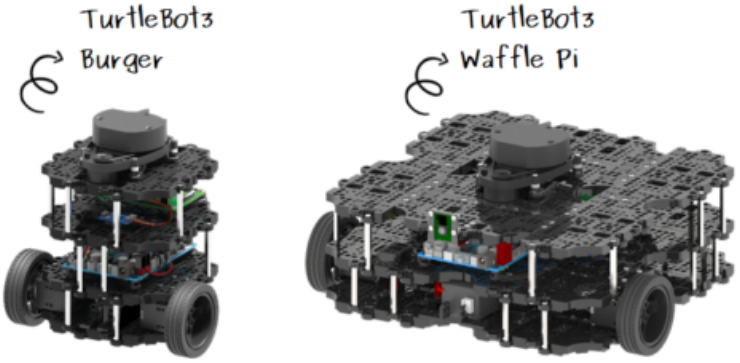
Gazebo

```
edgar@ubuntu:~$
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`



Turtlebot3

`edgar@ubuntu:~/IR2117/directorio$`
`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}}"` : Parar el robot

Conectarlo

1. Conectar la batería.
2. Encender la placa y esperar a que se encienda.
3. Si estás en un ordenador de clase se conectará automáticamente al internet por cable de clase. Si no, hace falta un usb wifi para conectarlo.
4. `ssh ubuntu@192.168.0.xxx` (password : turtlebot) : Se conecta a la placa del robot de forma remota.
5. `cd ~/ROS2 && ./configure.bash` : Configura el entorno de trabajo para poder usar ROS2.
6. Ejecutar los siguientes comandos:
   ```
   source /opt/ros/foxy/setup.bash
   export ROS_DOMAIN_ID=$N
   export TURTLEBOT3_MODEL=burger
   ros2 launch turtlebot3_bringup robot.launch.py
   ```

Desconectarlo

1. `sudo halt -p` : Se paran todos los procesos del turtlebot y se desconecta.
2. Apagar la placa cuando el LED amarillo pare de parpadear
3. Quitar la batería

Actions

`edgar@ubuntu:~$`

Las actions son como services que permiten ejecutar tareas que tardan en completarse, dan feedback continuo y son cancelables. En robots son muy útiles para navegación.

ejemplo.action
int32 goal
---
int32 result
---
int32[] feedback

`ros2 action send_goal <action_name> <action_type> <values> --feedback` : Enviar una acción desde la terminal. Añadir "--feedback" para ver también el feedback que va devolviendo.

ros2bag

`edgar@ubuntu:~$`
`ros2 bag record /scan` : Guarda todos los datos que se publican en el topic /scan

`ros2 bag info rosbag2_año_mes_dia-hora_minutos_segundos/rosbag2_año_mes_dia-hora_minutos_segundos_version.db3` : Muestra información sobre la grabación (principio, fin, nº mensajes, etc.)

`ros2 bag play rosbag2_año_mes_dia-hora_minutos_segundos/rosbag2_año_mes_dia-hora_minutos_segundos_version.db3` : Publica todos los mensajes en el topic como si se estuviera ejecutando el programa que hemos grabado

PREGUNTA PARCIAL

Ejecutar el programa square del directorio robot_trajectory en turtlesim con 0'5 m/s de velocidad lineal, 1'2 m/s de velocidad angular y 1'5 m de longitud del cuadrado.

```
ros2 run robot_trajectory square \
--ros-args \
--remap /cmd_vel:=/turtle1/cmd_vel \
--param linear_speed:=0.5 \
--param angular_speed:=1.2 \
--param square_length:=1.5
```

CMAKELISTS

```
cmake_minimum_required(VERSION 3.5)
project(olympic)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(turtlesim REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(rclcpp_components REQUIRED)
find_package(olympic_interfaces REQUIRED)

add_executable(rings src/rings.cpp)
target_include_directories(rings PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
ament_target_dependencies(
  rings
  rclcpp
  std_msgs
  geometry_msgs
  turtlesim
)

add_executable(action_server src/action_server.cpp)
ament_target_dependencies(action_server
  rclcpp
  rclcpp_action
```

```
  geometry_msgs
  turtlesim
  olympic_interfaces)

add_executable(action_client src/action_client.cpp)
ament_target_dependencies(action_client
  rclcpp
  rclcpp_action
  olympic_interfaces)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # uncomment the line when a copyright and license is not present in all
source files
  #set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # uncomment the line when this package is not in a git repo
  #set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

install(TARGETS
  rings
  action_server
  action_client
  DESTINATION lib/${PROJECT_NAME})

install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME})

ament_package()
```

package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>olympic</name>
  <version>1.0.0</version>
  <description>Olympic rings</description>
```

```
  <maintainer email="al416657@uji.es">edgar</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>geometry_msgs</depend>
  <depend>turtlesim</depend>
  <depend>rclcpp_action</depend>
  <depend>rclcpp_components</depend>
  <depend>olympic_interfaces</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

LAUNCH FILE

```
from launch import LaunchDescription
from launch_ros.actions import Node
import launch.actions

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='nombre_del_paquete',
            executable='nombre_del_nodo',
            namespace='front',
            remappings=[
                ('/cmd_vel', 'turtle1/cmd_vel'),
            ],
            parameters=[
                  {"parametro1": valor1},
                  {"parametro2": valor2},
                  # Añadir más parámetros si es necesario
            ]
        ),
        launch.actions.ExecuteProcess(
```

```
            cmd=['ros2', 'param', 'set', '/nombre_del_nodo', 'parametro'],
            output='screen'
        ),
        # Añadir más launch.actions.ExecuteProcess si es necesario
    ]
```

MAKEFILE

```
all: median mode

median: median.cpp
    g++ -I/usr/include/eigen3 -o median median.cpp

mode: mode.cpp
    g++ -I/usr/include/eigen3 -o mode mode.cpp
```

Eigen

`Vector2d`

`#include <Eigen/Dense>` : Importa la librería Eigen (solo son ficheros de cabecera).

`Vector`

`using Eigen::Vector2d;` : Para vectores fijos (más eficientes) de tamaño 2

…

`using Eigen::Vector6d;` : Hasta vectores fijos de tamaño 6

`using Eigen::VectorXd;` : Para vectores variables

`VectorXd ejemplo(3);` // Inicializa un vector de tamaño 3.

`Vector2d ejemplo(4, 3);` // Inicializa un vector de tamaño fijo con los valores 4 y 3.

`ejemplo(i) = 1;` // Guarda un “1” en el índice i del vector.

`double maximo = ejemplo.maxCoeff();` // Elemento máximo

`double minimo = ejemplo.minCoeff();` // Elemento mínimo

Para vectores variables:

`ejemplo.conservativeResize(6);` // Cambia el tamaño del vector a 6 conservando sus elementos.

`ejemplo.resize(10);` // Cambia el tamaño del vector a 10 sin conservar sus elementos.

`Matrix`

`using Eigen::MatrixXd;` // Para matrices variables

`using Eigen::Matrix2d;` // Para matrices 2x2 de tamaño fijo (más eficientes)

…

`using Eigen::Matrix4d;` // Para matrices 4x4 de tamaño fijo (usar tamaño fijo hasta matrices 4x4, si son más grandes usar MatrixXd).

`MatrixXd matriz(3,2);` // Crea una matriz de 3x2.
`MatrixXd matriz = MatrixXd::Random(3,3);` // Crea una matriz 3x3 con valores aleatorios.
`Matrix3d matriz = Matrix3d::Constant(3.2);` // Crea una matriz 3x3 con todos sus elementos iguales a 3’2.

`matriz(1, 0) = 2;` // Asigna un 2 al elemento de la fila 1, columna 0.
`double valor = matriz(0,0);` // Guarda el elemento que hay en el (0, 0) en la variable valor.
`Eigen::MatrixXd suma = matriz1 + matriz2;` // Suma de matrices.
`Eigen::MatrixXd resta = matriz1 - matriz2;` // Resta de matrices
`Eigen::MatrixXd producto = matriz1 * matriz2;` // producto de matrices (esto requiere que las dimensiones sean apropiadas).
`double maximo = matriz.maxCoeff();` // Elemento máximo
`double minimo = matriz.minCoeff();` // Elemento mínimo
`double suma_elementos = matriz1.sum();` // Suma de elementos

Para matrices variables:
`matriz.conservativeResize(5, 3);` // Cambia el tamaño de la matriz a una 5x3 conservando los elementos.
`matriz.resize(4, 2);` // Cambia el tamaño de la matriz a una 4x2 sin conservar los elementos.

TF2

`ros2 run tf2_ros tf2_echo turtle2 turtle1` : Muestra la transformación (traslación y rotación) de “turtle2” respecto a “turtle1”.

`ros2 run rviz2 rviz2 -d /opt/ros/foxy/share/turtle_tf2_py/rviz/turtle_rviz.rviz` : Especifica “turtle_rviz.rviz” como archivo de configuración de rviz y visualiza los marcos tf2 en 3D.

RViz2

`rviz -d ‘~/ro2_ws/src/unit3_config.rviz’` : Ejecuta RViz con el archivo indicado en la ruta que sigue a la flag “-d”.