Enlace a GitHub con todos los programas: https://github.com/al415491/IR2117

CADA VEZ QUE MODIFIQUEMOS UN PROGRAMA -> HACER COLCON BUILD

Para usar ros en una carpeta:

Terminal 1 - Subscriber

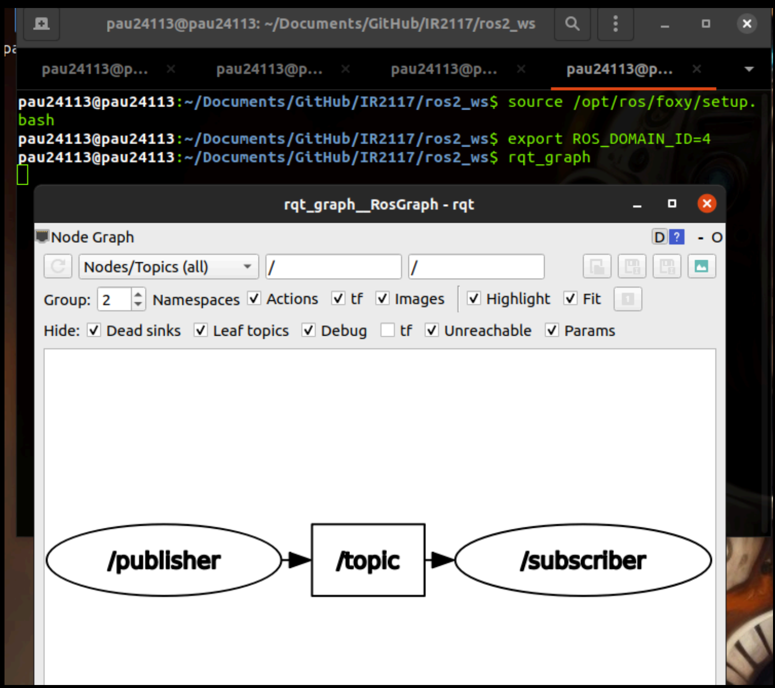
```
$ source /opt/ros/foxy/setup.bash  (Hace ajustes necesarios)
$ colcon build ( builder/compilador de ROS2)
$ source install/setup.bash → para utilizar programa
$ export ROS_DOMAIN_ID=4 → para que no interferir con los compañeros
$ ros2 run examples_topics subscriber  -> ejecutar el programa
            pckg        programa
```

Terminal 2 - Publisher

```
$ source /opt/ros/foxy/setup.bash  (Hace ajustes necesarios)
$ source install/setup.bash → para utilizar programa
$ export ROS_DOMAIN_ID=4 → para que no interferir con los compañeros
$ ros2 run examples_topics publisher  -> ejecutar el programa
            pckg        programa
```

Terminal 3 - Nodos

```
$ source /opt/ros/foxy/setup.bash  (Hace ajustes necesarios)
$ colcon build ( builder/compilador de ROS2)
$ source install/setup.bash → para utilizar programa
$ export ROS_DOMAIN_ID=4
$ rqt_graph (ver nodos)
```



Package.xml ----------

Fichero con info sobre pckg: autor, licencia, etc. Sirve para conocer al desarrollador por si se publica. También tiene dependencias que nuestro programa necesita de otros módulos.

```
<package format="3">
  <name>robot_trajectory</name>
  <version>0.0.0</version>
  <description>Robot trajectories</description>
  <maintainer email="al415491@uji.es">Pau Montagut</maintainer>
  <license>Apache License 2.0</license>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <build_depend>rclcpp</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>nav_msgs</build_depend>
  <exec_depend>rclcpp</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>nav_msgs</exec_depend>
  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

CMakeLists.txt----------

Fichero de instrucciones para compilar nuestro paquete.

- Find_package(busca las dependencias)
- Genera un ejecutable a partir del publisher.cpp y subscriber.cpp
- Instala cosas de ros2

Normalmente se generan de manera automática y si procede hay que cambiar alguna dependencia, ejecutable, etc.

⬇⬇ EJEMPLO CMAKE ⬇⬇

```
cmake_minimum_required(VERSION 3.5)
project(robot_trajectory)

# 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(geometry_msgs REQUIRED)
find_package(nav_msgs REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

add_executable(square src/square.cpp)
add_executable(square_odom src/square_odom.cpp)
ament_target_dependencies(square rclcpp geometry_msgs)
ament_target_dependencies(square_odom rclcpp nav_msgs)

install (TARGETS
 square
 square_odom
 DESTINATION lib/${PROJECT_NAME}
)

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()

ament_package()
```

COMANDOS ROS2 ----------

· ros2 node list (te dice nodos activos)
· ros2 node info <node_name> (info más concreta sobre un nodo → topic)
· ros2 topic list (topics del sistema, unos de uso interno, otros creados por nosotros)
· ros2 topic echo <topic_name> (muestra msgs → como ejecutar el subscriber básicamente)
· ros2 topic info <topic_name> (info del topic)
· ros2 interface show <msg type> (msg type lo sacamos del info)
· ros2 topic pub <topic_name> <msg_type> '<args>' (publica el mensaje, hacerlo en la terminal de publisher)
· ros2 topic pub /topic std_msgs/String "{data: Hello ROS Developers}"
· ros2 topic hz <topic_name> (frecuencia con la que publica)

APUNTES ROS2, C++, BASH ----------

· workspace es el directorio superior, con paquetes(subdirectorios) con diferentes ficheros
· int argc: nº de parámetros
· 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)

Terminal
$ myprog arg1 arg2 arg3

argc = 4 → argv[0] = myprog, argv[1] = arg1

Eigen

- **() starts index with 1; [] starts index with 0**
- **Sort:** `std::sort(elements.data(), elements.data() + elements.size());`
- **Vectors use fixed size.** `VectorXd elements(1);`
- **Resize:** `elements.conservativeResize(elements.size +1);`

Median:

```cpp
#include <iostream>
#include <Eigen/Dense>

using Eigen::VectorXd;

int main() {
   double m = 0;
   int n = 0, input;
   VectorXd elements(1);
   std::cout << "Give me a number (0 to finish): " << std::endl;
   std::cin >> input;

   while (input != 0) {
       elements.conservativeResize(n+1);
       elements(n) = input;
       n++;
       std::cout << "Give me a number (0 to finish): " << std::endl;
       std::cin >> input;
   }

   std::sort(elements.data(), elements.data() + elements.size());
   int middle = elements.size()/2;
   if (n % 2 == 0) {  // par
       std::cout << "Even " << std::endl;
       m = double(elements(middle) + elements(middle-1))/2;
   } else {  // impar
       m = elements(middle);
   }

   std::cout << "Median: " << m << std::endl;
   return 0;
}
```

- **Compile:**

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

(Cada uno puede tener un path diferente)

Mode:

```cpp
#include <iostream>
#include <Eigen/Dense>

using Eigen::VectorXd;

int main() {
   double m = 0;
   int n = 0, input;
   VectorXd elements(1);
   std::cout << "Give me a number (0 to finish): " << std::endl;
   std::cin >> input;

   while (input != 0) {
       elements.conservativeResize(n+1);
       elements(n) = input;
       n++;
       std::cout << "Give me a number (0 to finish): " << std::endl;
       std::cin >> input;
   }

   int max_count = 0;
   for (int i = 0; i < n; i++) {
       int count = 0;
       for (int j = 0; j < n; j++) {
           if (elements[j] == elements[i]) {
               count++;
           }
       }
       if (count > max_count) {
           max_count = count;
           m = elements[i];
       }
   }

   std::cout << "Mode: " << m << std::endl;
   return 0;
}
```

- Compile:

```
$ g++ -I/usr/include/eigen3 -o median mode_eigen.cpp
```

(Cada uno puede tener un path diferente)

Publicar msg ----------

Terminal 1 → PUBLICAR MENSAJES

```
$ source /opt/ros/foxy/setup.bash  (Hace ajustes necesarios)
$ source install/setup.bash → para utilizar programa
$ export ROS_DOMAIN_ID=4 → para que no interferir con los compañeros
$ros2 topic pub /topic std_msgs/msg/String "{data: 'klk manin'}"
```

Teleop ----------

Terminal 1 - Abrir GAZEBO

```
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

Terminal 2 - Ejecutar Programa

```
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ ros2 run turtlebot3_teleop teleop_keyboard
```

Parar robot en movimiento ----------

Terminal 1 - Parar publisher

ctrl+c, publisher de movimiento

Terminal 2 - Publicar velocidad 0

```
ros2 topic pub --once /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}}"
```

Square con iteraciones ----------

Terminal 1 - Abrir GAZEBO

```
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

Terminal 2 - Ejecutar Programa

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ ros2 run robot_trajectory square_iterations
```

Square Odometry

Terminal 1 - Abrir GAZEBO

```
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

Terminal 2 - Ejecutar Programa

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ ros2 run robot_trajectory square_odom
$ ros2 run robot_trajectory square_odom --ros-args --param linear_speed:=0.4
--param angular_speed:=0.4 --param square_length:=2.0
```

Turtlesim

Terminal 1 - Abrir Turtlesim

```
$ ros2 run turtlesim turtlesim_node
```

Terminal 2

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ 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
```

Terminal → Teleop

```
$ ros2 run turtlesim turtle_teleop_key
```

Test in Turtlebot 3 ----------

1º Connect your PC to the WiFi network with an USB dongle.
2º Switching the robot on:
    Plug in the battery
    Switch on the board
    Wait a minute approximately for the system to boot

The robot should automatically connect to the WiFi network

3º Connect to the robot with `$ ssh ubuntu@192.168.0.xxx`
(password “turtlebot”)
4º Upload the ROS2 firmware
`$ cd ~/ROS2 && ./configure.bash`
(a few seconds after uploading, the board should beep several times)

Run the following commands in the robot:

Terminal 1

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=...
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_bringup robot.launch.py
```

Terminal 2

Run the square_odom application on the PC.

Switching the robot off ----------

1º Connect to the robot with ssh ubuntu@192.168.0.xxx (password “turtlebot”)
2º Stop the system with the command: `$ sudo halt -p`
3º When the yellow LED stops blinking, switch off the board
4º Remove the battery
5º Put the battery in a charger

RViz ----------

· Similar en ROS y ROS2
· Muy versátil (Por ende difícil)

Terminal 1

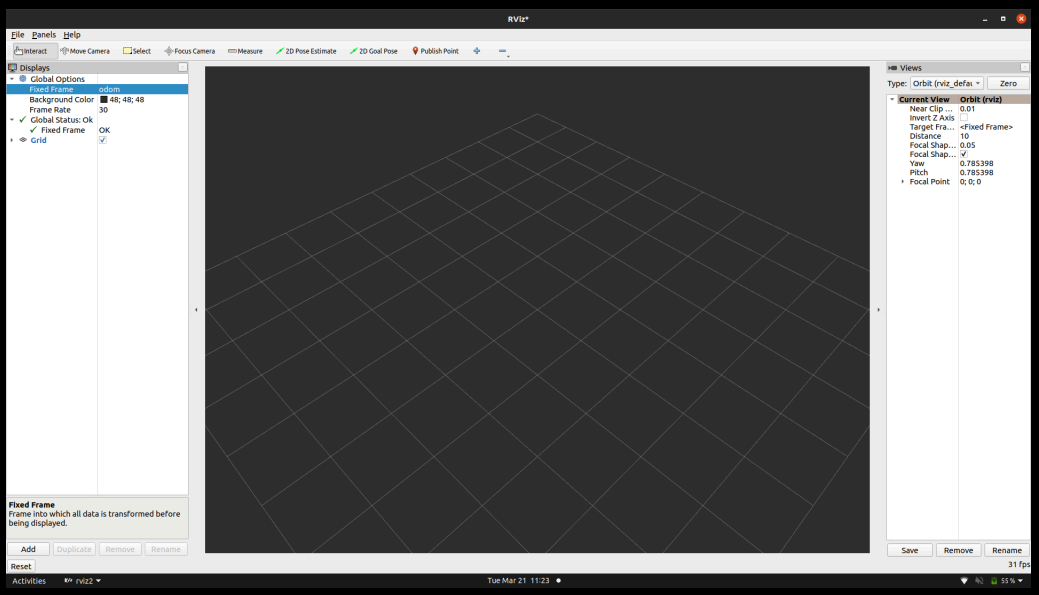
```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ export TURTLEBOT3_MODEL=waffle_pi
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Terminal 2 - Teleop

```
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export TURTLEBOT3_MODEL=waffle_pi
$ ros2 run turtlebot3_teleop teleop_keyboard
```

- **Terminal 3 - RViz**

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ rviz2
(Ajustes creados por nosotros usamos el siguiente comando -d indica directorio)
$ rviz2 -d Documents/Github/IR2117/tb3.rviz
```



- Cambiar Fixed Frame: map por odom
(Odometría → cálculo pos y orientación robot respecto a la inicial)

- Poner Type en TopDown y Scale 61

- Incorporar elementos en add: robot model

- Poner en description topic: robot_description

- add odometry y topic /odom y desmarcar covariance

- add laser scan y topic /scan, size = 0.03, color transformation = Flat Color, green

- add camera y topic /camera/image_raw

- Podemos organizar por Topic

RQt ----------

· Se usa para no usar tantas terminales

Terminal 3 - RViz
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ rqt

El menú plugins tiene todo lo que podemos visualizar:

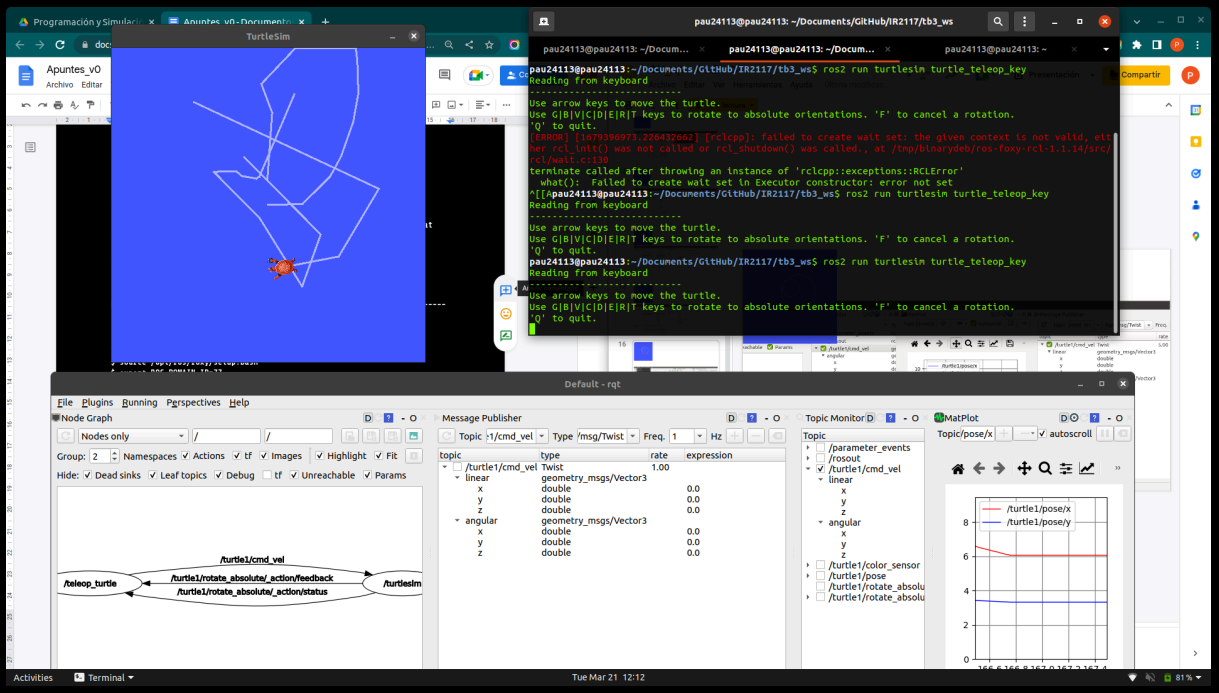
- rqt graph: Plugins/Introspection/graph

- Topics/Topic Monitor → lista de topics

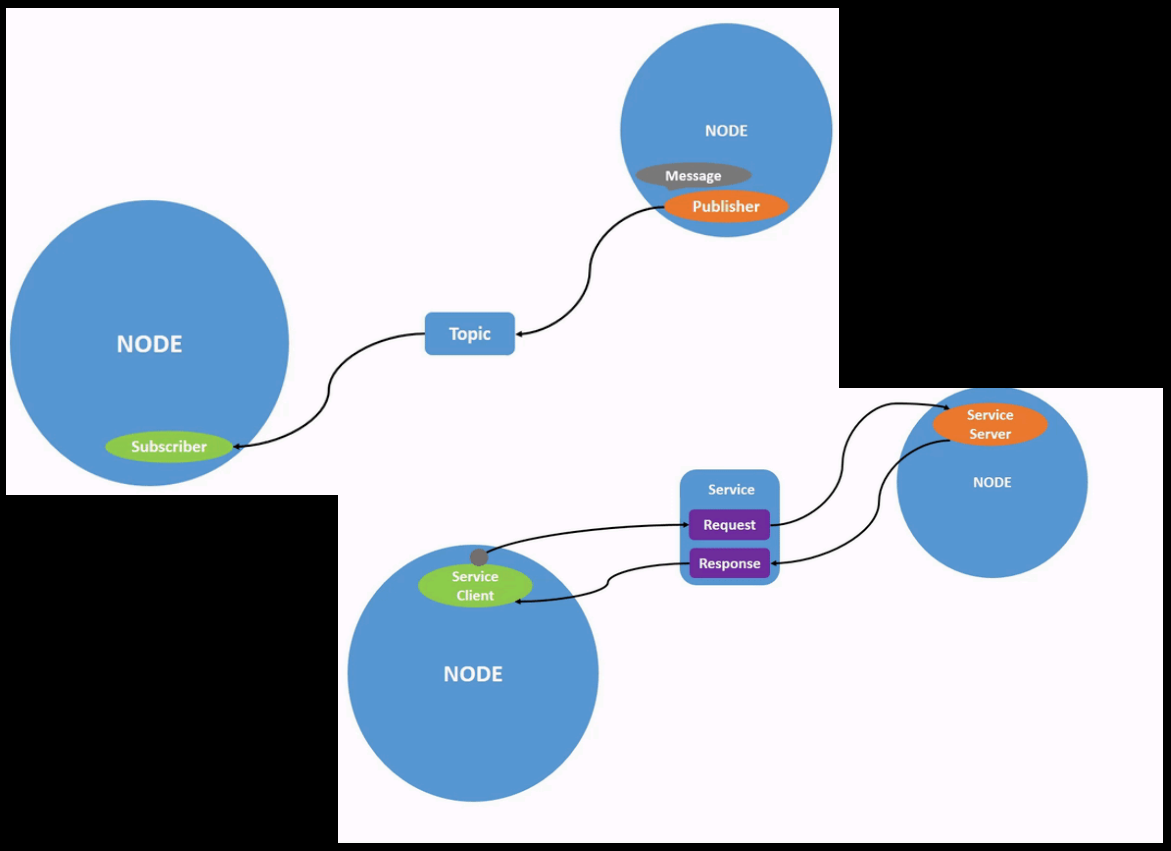
- Topics/message publisher → publicar velocidad

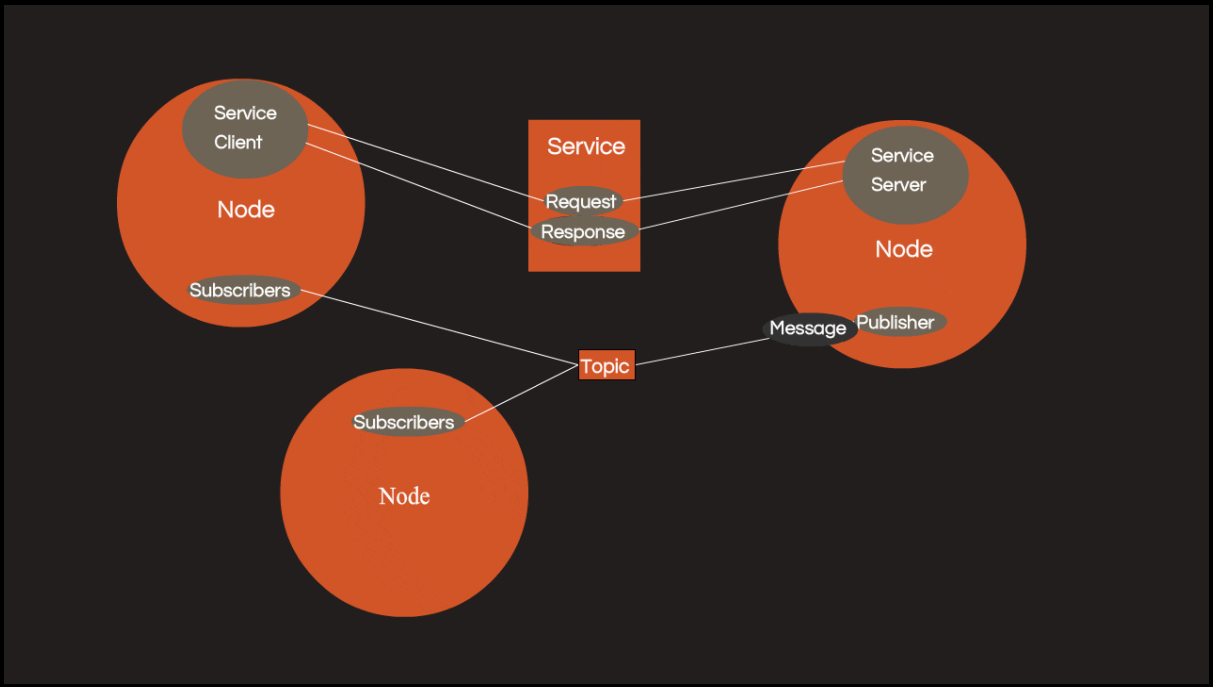
- Visualization/Plot → gráfico posición, añadimos turtle1/pose/x y turtle1/pose/y

- Configuration/Dynamic Reconfigure → cambiar fondos



ROS services ----------

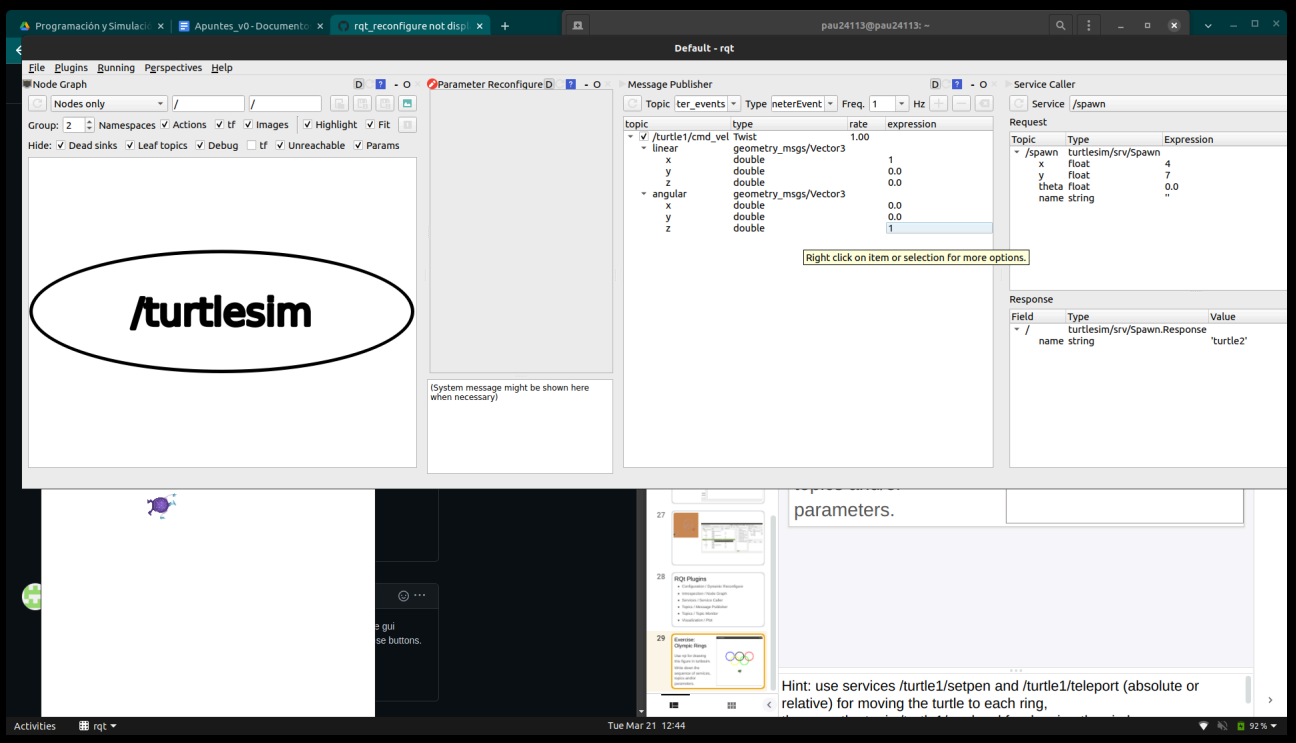




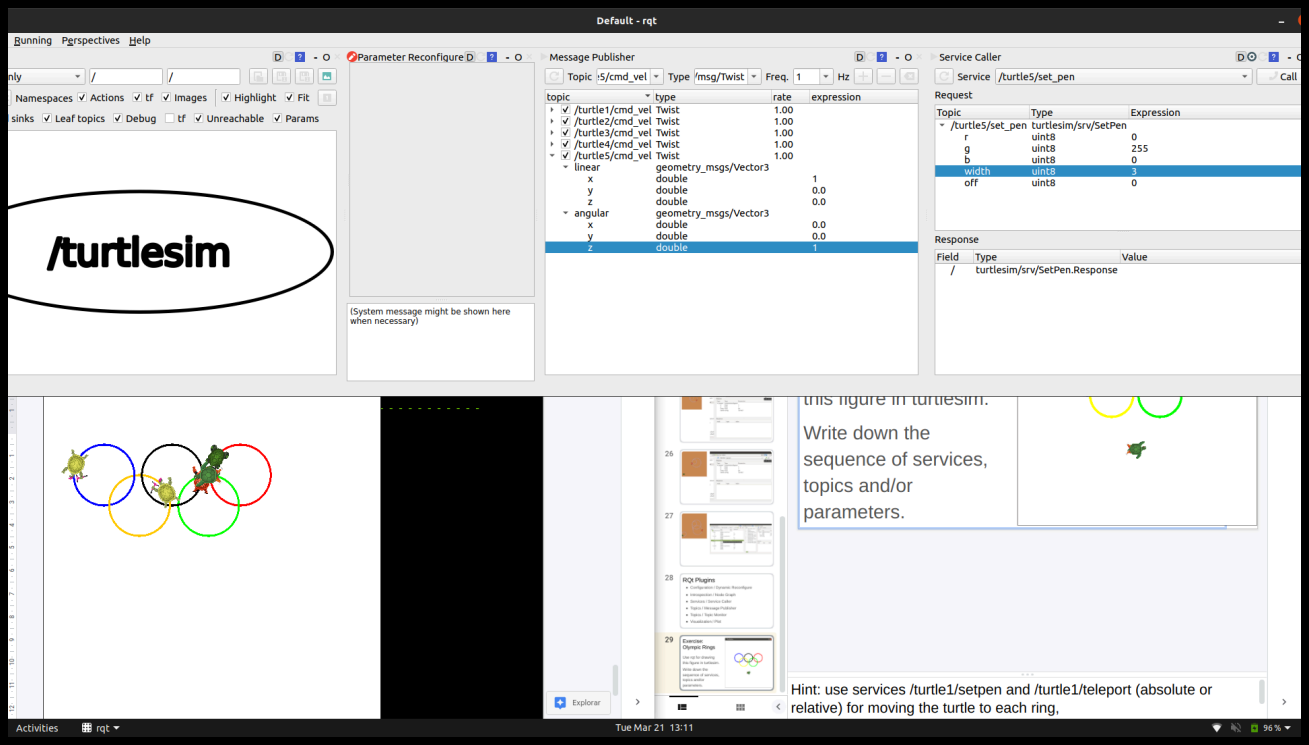
- Plugins/Services/Service Caller salen los servicios del turtlesim

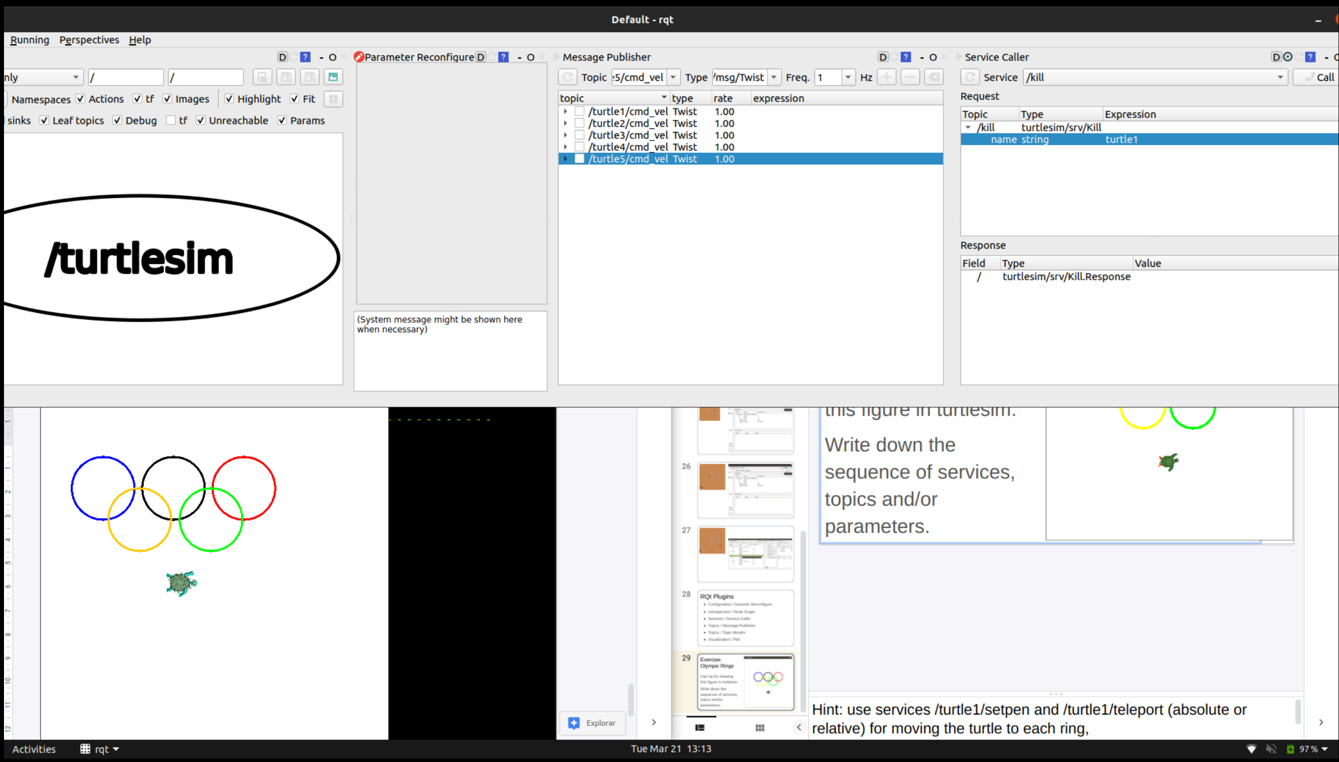
- /spawn → sirve para spawnear varias tortugas. Ponemos nombre y call
- /set_pen → rastro de la tortuga
- /teleport → absolute, relative

Exercise:
Olympic Rings
Use rqt for drawing this figure in turtlesim. Write down the sequence of services, topics and/or parameters.



1º Cambiar background en dynamic reconfigure
2º Spawnear 5 tortugas en diferentes posiciones y cambiar con set pen el rastro
3º Hacer círculos con linear.x = 1 y angular.z = 1
4º Borrar tortugas





Olympic rings services--

Terminal 1 - Abrir Turtlesim

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 run turtlesim turtlesim_node
```

Terminal 2 - Ejecutar nodo rings

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 run olympic rings --ros-args --param radius:=1.0
```

Wandering --

Terminal 1

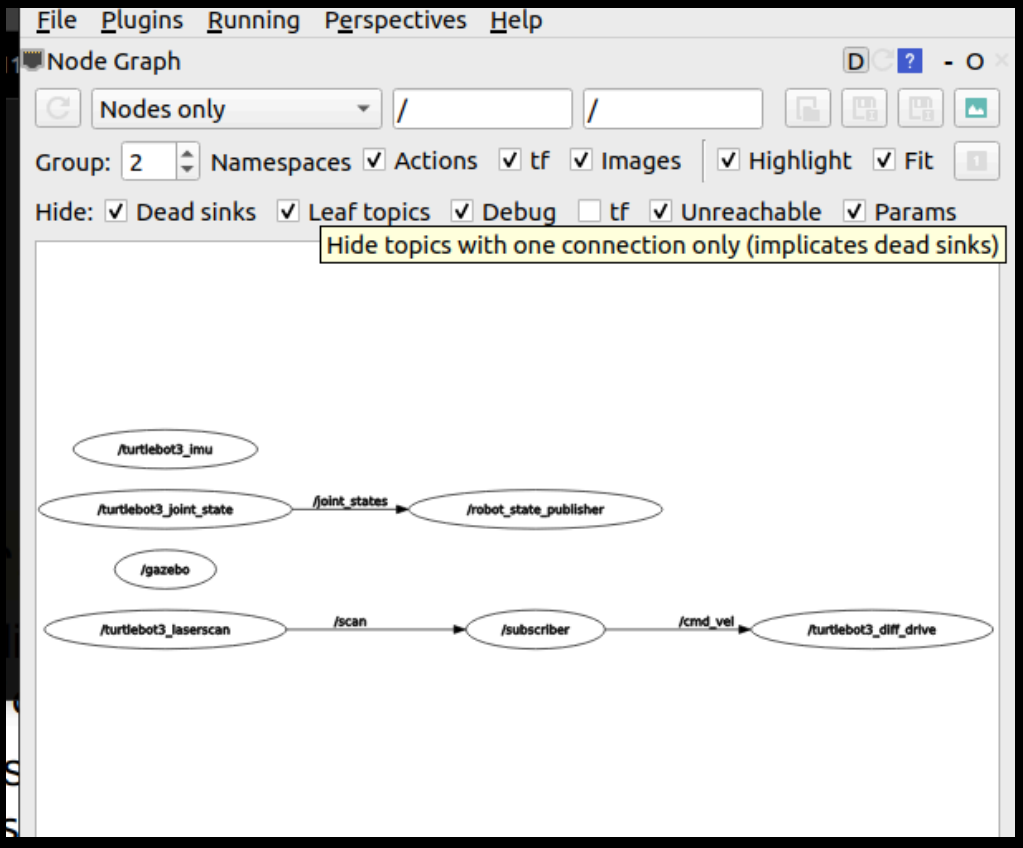
```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Terminal 2 - /scan

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 topic echo /scan --no-arr
```

Terminal 2 - Ejecutar Programa

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 run robot_trajectory wandering
```



v2

Weebots ----------

Terminal 1

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
```

```
$ export WEBOTS_HOME=/home/pau24113/webots-R2022b
$ ros2 launch webots_ros2_turtlebot robot_launch.py
```

Terminal 2 - /scan

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 topic echo /scan --no-arr
```

Terminal 2 - Ejecutar Programa

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 run robot_trajectory wandering
```

wandering++ ----------

Terminal 1

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Terminal 2 - Ejecutar Programa para detectar

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 launch obstacles triple_detector.launch.py
```

Terminal 3 - Ejecutar Programa para avoidance

```
$ cd Documents/GitHub/IR2117/tb3_ws/
$ source /opt/ros/foxy/setup.bash
$ source install/setup.bash
$ export ROS_DOMAIN_ID=77
$ ros2 run obstacles avoidance
```

Terminal 1 - Webots

```
$ source /opt/ros/foxy/setup.bash
$ export ROS_DOMAIN_ID=77
$ export WEBOTS_HOME=/home/usuario/webots-R2022b
$ ros2 launch webots_ros2_turtlebot robot_launch.py
```

25/04/23

.mgs -> simple ROS message
.srv -> service (client-server) with request and response
.action -> service with feedback