

Componentes en ROS2

- 1 Componentes en ROS2
- 2 Lanzamiento
- 3 Modelado de robots
- 4 RViz2
- 5 Bibliografía

Temas

- 1 Componentes en ROS2
 - Paquetes
 - Compilar e instalar un paquete

Crear un paquete C++

- Crea un paquete en tu espacio de trabajo.

```
cd ~/ROS/taller_ws/src
ros2 pkg create --build-type ament_cmake --node-name viz_node viz_package_cpp
  ↳--license Apache-2.0
```

Temas

- 1 Componentes en ROS2
 - Paquetes
 - Compilar e instalar un paquete

Compilar e instalar un paquete

Terminal 1: Compilar e instalar

- Puedes compilar e instalar localmente todos los paquetes en tu espacio de trabajo con:

```
colcon build
```

- Para compilar sólo el paquete que te interesa:

```
colcon build --packages-select viz_package_cpp
```

Terminal 2: Usar

- Para utilizar el paquete debes abrir **una nueva terminal** y hacer source:

```
source install/local_setup.bash
```

- Para correr un nodo:

```
ros2 run viz_package_cpp viz_node
```

Lanzamiento

- 1 Componentes en ROS2
- 2 Lanzamiento**
- 3 Modelado de robots
- 4 RViz2
- 5 Bibliografía

Archivos de lanzamiento

- Un *launch file* permite realizar varias configuraciones para el ambiente y ejecutar multiples nodos.
- Para utilizar esta herramienta dentro de tu paquete, en el archivo `package.xml` que se creó con el paquete, en cualquier lugar entre las etiquetas `<package>` agrega la línea:

```
<exec_depend>roslaunch</exec_depend>
```

- Dentro del paquete al que se agregarán archivos de lanzamiento crear un directorio llamado `launch`:

```
cd ~/ROS/taller_ws/src/viz_package/  
mkdir launch
```


Archivo

- Crea ahí un archivo con el nombre y contenido siguiente:

Código: viz_launch.py

```
1 from launch import LaunchDescription
2 from launch_ros.actions import Node
3
4 def generate_launch_description():
5     return LaunchDescription([
6         Node(
7             package='viz_package_cpp',
8             namespace='paquito1',
9             executable='viz_node',
10            name='viz',
11            output='screen',
12        )
13    ])
```

Lanzando

- Ahora puedes ejecutar los elementos configurados en el lanzador, estando dentro del directorio `launch`:

```
ros2 launch viz_launch.py
```

Configurando el archivo

- Cada vez que se agrega un elemento al paquete, es necesario indicarlo en su archivo `CMake.txt`, para que sea tomado en cuenta al instalarlo.
- Agregar a dicho archivo el código para indicar el directorio al cual deberá enviar los archivos *launch* cuando el paquete sea instalado:

Código: Modificaciones a CMakeLists.txt

```
1 # Install
2 install(DIRECTORY
3   launch      # Launch files
4   DESTINATION share/${PROJECT_NAME}
5 )
```

Lanzando con el paquete

- En la terminal 1, compila/instala el paquete para incluir el archivo lanzador:

```
colcon build
```

- Ahora es posible invocar el lanzador desde la terminal 2 desde cualquier punto:

```
ros2 launch viz_package_cpp viz_launch.py
```

Modelado de robots

- 1 Componentes en ROS2
- 2 Lanzamiento
- 3 Modelado de robots
- 4 RViz2
- 5 Bibliografía

Temas

- 3 Modelado de robots
 - Forma
 - Movimiento

URDF

- En la carpeta del paquete, agrega el directorio urdf.
- Dentro crea el archivo siguiente:

Código: robot.urdf

```
1 <?xml version="1.0"?>
2 <robot name="paquito">
3   <link name="base_link">
4     <visual>
5       <geometry>
6         <cylinder length="0.6" radius="0.2"/>
7       </geometry>
8     </visual>
9   </link>
10 </robot>
```

Más detalles sobre la geometría: [ROS2 URDF Tutorial](#)

[XML Robot Description Format \(URDF\)](#)

Agregar como recurso del paquete

- Agregar directiva install:

Código: Modificaciones a CMakeLists.txt

```
1 # Install
2 install(DIRECTORY
3     launch      # Launch files
4     urdf        # robot models
5     DESTINATION share/${PROJECT_NAME}
6 )
```


Temas

- 3 Modelado de robots
 - Forma
 - Movimiento

Publicaciones del robot

- Para que ROS2 esté al tanto de lo que hace el robot, deberemos publicar las modificaciones a su estado conforme ejecutemos comandos o sus sensores reciban información.
- Usaremos dos **nodos** de ROS2 para esto, por lo que hay que agregarlos entre los requisitos del paquete:

Código: Modificaciones a package.xml

```
1 <exec_depend>joint_state_publisher</exec_depend>
2 <exec_depend>robot_state_publisher</exec_depend>
```

- Para asegurarse de que estén instalados:
 - Desde la raíz del espacio de trabajo usar:

```
rosdep install -i --from-path src --rosdistro jazzy -y
```

- O ejecutar:

```
sudo apt install ros-jazzy-urdf-tutorial  
echo 'export LC_NUMERIC="es_MX.UTF-8"' >> ~/.bashrc
```

Lanzando al nodo que publica

- Hay que pedir al archivo de lanzamiento que ejecute al nodo que publicará la descripción del robot.
- Recibirá como parámetro el contenido del archivo con dicha descripción. Usamos las funciones de python para leer este contenido y pasarlo al nodo.

Código: Modificaciones a launch.py

```
1 import os
2 from ament_index_python.packages import get_package_share_directory
3 # ...
4 def generate_launch_description():
5     package_dir = get_package_share_directory('viz_package_cpp')
6     path_to_urdf = os.path.join(package_dir, 'urdf', 'robot.urdf')
7     with open(path_to_urdf, 'r') as f:
8         robot_desc = f.read()
```

Código: Modificaciones a launch.py

```
1 def generate_launch_description():
2     # ...
3     return LaunchDescription([
4         # ...
5         Node(
6             package='robot_state_publisher',
7             name='robot_state_publisher',
8             executable='robot_state_publisher',
9             output='screen',
10            parameters=[{
11                'robot_description': robot_desc,
12                'publish_frequency': 30.0,
13            }]
14        ),
```

- Compila y lanza en las terminales respectivas.
- Ahora revisa los temas sobre los que pueden platicar los nodos:

```
ros2 topic list
```

El que más nos importa ahora es `/robot_description`.

RViz2

- 1 Componentes en ROS2
- 2 Lanzamiento
- 3 Modelado de robots
- 4 RViz2**
- 5 Bibliografía

Temas

4 RViz2

- Ejecución
- Adición de marcadores
- Publicación desde un nodo C++
- Mover al robot

Ejecución y tópicos

- Antes de ejecutar `rviz2` veamos qué escucha ROS2.

```
ros2 topic list
```

- Ahora ejecuta `rviz2`:

```
rviz2
```

- Vuelve a listar los tópicos.
- Crea el directorio `rviz` al lado de `urdf`. en `rviz2` elige *File -> Save Config As* y guardar en ese directorio `panel.rviz`.
- Agregar este directorio archivo a `install` en `CMakeLists.txt`.

Lanzamiento de rviz2

- En el archivo `viz_launch.py` agrega otro nodo a la lista que se pasa al constructor de `LaunchDescription`:

Código: `viz_launch.py`

```
1 #...
2     Node(
3         package='rviz2',
4         executable='rviz2',
5         name='rviz2',
6         arguments=['-d', os.path.join(package_dir, 'rviz', 'panel.
➡rviz')],
7         output='screen'
8     )
```

Lanzamiento

- En la terminal 1 ejecuta:

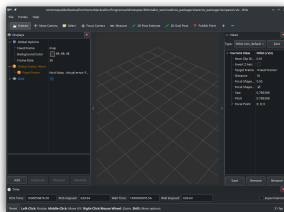
```
colcon build
```

- En la terminal 2 ahora puedes ejecutar con:

```
ros2 launch viz_package_cpp viz_launch.py
```

Ver al robot

- En la interfaz de RViz2, en el panel Displays a la izquierda, selecciona Fixed Frame y reemplaza map por base_link.
- Abajo a la izquierda presiona el botón Add.



- Elige RobotModel, ahora aparecerá la opción a la izquierda en Displays.

- Despliega RobotModel y en Description topic elige /robot_description.
¡Ahora puedes ver el robot!
- Guarda la configuración en el directorio dentro de src que hiciste para este propósito.
- Vuelve a instalar. La próxima vez que hagas el lanzamiento ya aparecerá el robot desde un principio.

Temas

4 RViz2

- Ejecución
- Adición de marcadores
- Publicación desde un nodo C++
- Mover al robot

Requerimientos

- Es posible agregar marcadores para ser visualizados en ROS 2 mientras el programa está en ejecución.
- Utiliza el comando siguiente para ver qué paquetes están instalados con ROS2:

```
ros2 pkg list
```

Necesitaremos:

```
visualization_msgs
```

agrega el `exec_depend` al `package.xml`.

Publicación manual

- Agrega un cubo manualmente (puede ser desde la terminal 1):

```
ros2 topic pub --once /paquito1/marker_topic visualization_msgs/msg/  
  Marker "{header:{frame_id:'base_link'},type:1,pose:{position:  
  {z:3.0}},scale:{x:0.5,y:1.0,z:1.0},color:{b:1.0,a:0.5},  
  lifetime:{sec:2}}"
```

- Hay más marcadores en <http://wiki.ros.org/rviz/DisplayTypes/Marker>

- Puedes usar

```
ros2 topic echo /paquito1/marker_topic
```

para escuchar lo que está siendo publicado.

- Para ver de qué tipo de mensaje se habla en cada tópico:

```
ros2 topic list -t
```

- Para ver qué información se incluye en un mensaje:

```
ros2 interface show visualization_msgs/msg/Marker
```

- En RViz2, así como agregaste el robot, agrega ahora un Marker y asigna el tópico correspondiente.

Temas

4 RViz2

- Ejecución
- Adición de marcadores
- Publicación desde un nodo C++
- Mover al robot

Requisitos

- Agregar dos dependencias a `package.xml`

```
1 <depend>rclcpp</depend>
2 <depend>visualization_msgs</depend>
```

- Agregar a `CMakeLists.txt`

```
1 find_package(rclcpp REQUIRED)
2 find_package(visualization_msgs REQUIRED)
3
4 # Después de add_executable
5 ament_target_dependencies(viz_node rclcpp visualization_msgs)
```

Publicación desde un nodo I

Código: viz_node.cpp

```
1 #include <chrono>
2 #include <memory>
3 #include <cstdio>
4
5 #include "rclcpp/rclcpp.hpp"
6 #include "visualization_msgs/msg/marker.hpp"
7
8 using namespace std::chrono_literals;
9
10 class MarkerPublisher : public rclcpp::Node
11 {
12 public:
13     MarkerPublisher() : Node("marker_publisher"), _count(0)
14     {
15         _publisher = this->create_publisher<visualization_msgs::msg::Marker>("marker_topic", 10);
16         //while(!_publisher->get_subscription_count()) {
17         //    sleep(1);
18         //    RCLCPP_INFO(this->get_logger(), "Waiting for subscribers...");
19         //}
20         auto timer_callback =
21             [this]() -> void {
22                 auto marker = visualization_msgs::msg::Marker();
```

Publicación desde un nodo II

```
23 marker.header.frame_id = "/base_link";
24 marker.header.stamp = this->get_clock()->now();
25
26 // set shape, Arrow: 0; Cube: 1 ; Sphere: 2 ; Cylinder: 3
27 marker.type = 1;
28 marker.id = 0;
29
30 // Set the scale of the marker
31 marker.scale.x = 1.0;
32 marker.scale.y = 1.0;
33 marker.scale.z = 1.0;
34
35 // Set the color
36 marker.color.r = 0.0;
37 marker.color.g = 1.0;
38 marker.color.b = 0.0;
39 marker.color.a = 0.5;
40
41 marker.lifetime.sec = 3;
42
43 // Set the pose of the marker
44 marker.pose.position.x = 5.0;
45 marker.pose.position.y = 0.0;
46 marker.pose.position.z = 0.0;
47 marker.pose.orientation.x = 0.0;
```

Publicación desde un nodo III

```
48         marker.pose.orientation.y = 0.0;
49         marker.pose.orientation.z = 0.0;
50         marker.pose.orientation.w = 1.0;
51
52         this->_publisher->publish(marker);
53     };
54     _timer = this->create_wall_timer(1000ms, timer_callback);
55 }
56 private:
57     rclcpp::TimerBase::SharedPtr _timer;
58     rclcpp::Publisher<visualization_msgs::msg::Marker>::SharedPtr _publisher;
59     size_t _count;
60 };
61
62 int main(int argc, char ** argv)
63 {
64     rclcpp::init(argc, argv);
65     rclcpp::spin(std::make_shared<MarkerPublisher>());
66     rclcpp::shutdown();
67     return 0;
68 }
```

Temas

4 RViz2





- Ejecución
- Adición de marcadores
- Publicación desde un nodo C++
- Mover al robot

<https://docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Using-URDF-with-Robot-State-Publisher.html>

Bibliografía

- 1 Componentes en ROS2
- 2 Lanzamiento
- 3 Modelado de robots
- 4 RViz2
- 5 Bibliografía**

Bibliografía I

-  Documentación oficial de ROS2
<https://docs.ros.org/en/iron/index.html>
-  Turtlebot 4 RViz2
<https://turtlebot.github.io/turtlebot4-user-manual/software/rviz.html>
-  Marker publishing example
<https://answers.ros.org/question/373802/minimal-working-example-for-rviz-marker-publishing/>
-  Webots y RViz
<https://www.youtube.com/watch?v=L9ID4QQJ8Cw&t=1065s>