URDF in ROS 2 — Explained for Embedded/Robotics Engineers

What is URDF

URDF (Unified Robot Description Format) is an XML format used in ROS to describe:

- Robot's structure
- Links (bodies) and joints (connections)
- Inertial, visual, and collision properties

It **does not** describe behavior, only **physical structure**.

Why URDF Matters

- Needed for simulation (Gazebo, RViz)
- Required for visualization
- Foundation for robot state publishers
- Input to tools like `robot_state_publisher`, `joint_state_publisher`, and controllers

Basic URDF Terms

Tag	Description
`<robot>`	Root tag

<link>	Describes a rigid body
<joint>	Connects two links
<visual>	Mesh/shape for RViz
<collision>	Shape used for collision detection
<inertial>	Mass, inertia, origin
<origin>	Position and rotation (xyz, rpy)

Example: Simple 2-Link Robot (Arm)

```
<!-- simple_arm.urdf -->

<robot name="two_link_arm">


  <link name="base_link"/>


  <joint name="joint1" type="revolute">

    <parent link="base_link"/>

    <child link="link1"/>

    <origin xyz="0 0 1" rpy="0 0 0"/>
```

```
    <axis xyz="0 0 1"/>
  </joint>

  <link name="link1">
    <visual>
      <geometry>
        <box size="0.5 0.1 0.1"/>
      </geometry>
    </visual>
  </link>

</robot>
```

Step-by-Step Practice with C++

1. Create Package

ros2 pkg create --build-type ament_cmake urdf_cpp_demo --dependencies rclcpp

---

2. Create URDF File

Create a folder:

mkdir -p urdf_cpp_demo/urdf

Add the `simple_arm.urdf` file inside.

---

3. C++ Code to Load and Display URDF

Create `src/urdf_loader.cpp`:

```
#include "rclcpp/rclcpp.hpp"

#include <fstream>

#include <streambuf>

#include <string>


int main(int argc, char **argv) {

    rclcpp::init(argc, argv);

    auto node = rclcpp::Node::make_shared("urdf_loader_node");


    std::ifstream urdf_file("src/urdf_cpp_demo/urdf/simple_arm.urdf");

    std::string urdf_str((std::istreambuf_iterator<char>(urdf_file)),
std::istreambuf_iterator<char>());


    if (urdf_str.empty()) {

        RCLCPP_ERROR(node->get_logger(), "Failed to load URDF");

        return 1;

    }


    RCLCPP_INFO(node->get_logger(), "URDF Loaded:\n%s", urdf_str.c_str());


    rclcpp::shutdown();

    return 0;
```

}

---

4. Modify `CMakeLists.txt`

Add:

add_executable(urdf_loader src/urdf_loader.cpp)

ament_target_dependencies(urdf_loader rclcpp)

install(TARGETS urdf_loader DESTINATION lib/${PROJECT_NAME})

---

5. Build and Run

colcon build

. install/setup.bash

ros2 run urdf_cpp_demo urdf_loader

Suggested Practice Tasks

- Modify URDF to add a second joint and link
- Add visual properties (colors, meshes)
- Load URDF into RViz using `robot_state_publisher`
- Create a robot with a gripper (fixed joint)
- Convert URDF to `xacro` format and parametrize link length

Advanced Ideas

- Build a 4-DOF robotic arm
- Use URDF + joint_state_publisher + RViz for real-time simulation
- Integrate with controllers (ros2_control)
- Link with real hardware using `ros2_control` and `hardware_interface`

Let's upgrade the URDF project step by step by:

1. Loading the URDF into **RViz**
2. Publishing TF using **robot_state_publisher**
3. Simulating joint movement using **joint_state_publisher_gui**

---

1. Create a Launch File

Create folder:

```
mkdir -p urdf_cpp_demo/launch
```

Create file: `launch/visualize.launch.py`

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

def generate_launch_description():
    urdf_path = 'src/urdf_cpp_demo/urdf/simple_arm.urdf'

    return LaunchDescription([
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='state_publisher',
```

```
        output='screen',
        parameters=[{'robot_description': open(urdf_path).read()}],
    ),
    Node(
        package='joint_state_publisher_gui',
        executable='joint_state_publisher_gui',
        name='joint_state_publisher',
        output='screen',
    ),
    Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        output='screen',
    )
])
```

---

2. Add to `CMakeLists.txt`

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

---

3. Build Again

```
colcon build
. install/setup.bash
```

---

4. Run the Robot in RViz

```
ros2 launch urdf_cpp_demo visualize.launch.py
```

You will see:

- A GUI to move joints

- Your URDF in RViz
- TF tree published
- Interactive control if `joint_state_publisher_gui` is running

---

5. Suggested Enhancements

Task	Purpose
Add `inertial` tag	Simulate physics later
Add more joints	Practice multi-DOF kinematics
Convert to `xacro`	Make URDF reusable
Add controllers	Enable velocity or effort control

Convert URDF to XACRO Format (Next Step)

XACRO = XML Macro
Gives you:

- Parameters
- Loops
- Cleaner URDF

---

1. Install Xacro

sudo apt install ros-humble-xacro

---

2. Create a New XACRO File

```
mkdir -p urdf_cpp_demo/urdf
```

Save this as `urdf_cpp_demo/urdf/simple_arm.xacro`:

```
<?xml version="1.0"?>
<robot name="two_link_arm" xmlns:xacro="http://ros.org/wiki/xacro">

  <xacro:property name="link_length" value="0.5"/>
  <xacro:property name="link_width" value="0.1"/>
  <xacro:property name="link_height" value="0.1"/>

  <link name="base_link"/>

  <joint name="joint1" type="revolute">
    <parent link="base_link"/>
    <child link="link1"/>
    <origin xyz="0 0 1" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>

  <link name="link1">
    <visual>
      <geometry>
        <box size="${link_length} ${link_width} ${link_height}"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 1 1"/>
      </material>
    </visual>
  </link>

</robot>
```

---

3. Update Launch File to Use `.xacro`

Modify this part in `launch/visualize.launch.py`:

```
import xacro
```

def generate_launch_description():
    urdf_path = 'src/urdf_cpp_demo/urdf/simple_arm.xacro'
    robot_description = xacro.process_file(urdf_path).toxml()

Then update the launch node:

parameters=[{'robot_description': robot_description}]

---

4. Test It

ros2 launch urdf_cpp_demo visualize.launch.py

If it works:

- You'll see the robot in RViz
- You can tune length/width without rewriting full URDF

---

5. Practice Tasks

- Create a `for` loop to generate 3 identical links
- Create a parameter to enable/disable colors
- Add arguments from launch file to control link size
- Add multiple joints with varying axis (`x`, `y`, `z`)

Now: Add `ros2_control` Support to XACRO URDF

Use `ros2_control` to simulate or control hardware (joints, actuators).

---

1. Update Your `simple_arm.xacro` File

Add this at the top **after** robot tag:

```
<ros2_control name="SimpleArmHardware" type="system">
  <hardware>
    <plugin>fake_components/GenericSystem</plugin>
  </hardware>
  <joint name="joint1">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
  </joint>
</ros2_control>
```

Then in the `<joint>` tag, add transmission:

```
<transmission name="joint1_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

---

2. Install Required Packages

```
sudo apt install ros-humble-ros2-control ros-humble-ros2-controllers
ros-humble-gazebo-ros2-control
```

---

3. Create a Controller Config File

Create: `config/position_controllers.yaml`

```
controller_manager:
```

```yaml
  ros__parameters:
    update_rate: 50

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

    joint1_position_controller:
      type: position_controllers/JointGroupPositionController

joint1_position_controller:
  ros__parameters:
    joints:
      - joint1
```

---

4. Create a New Launch File

`launch/control.launch.py`:

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

def generate_launch_description():
    urdf_file = "src/urdf_cpp_demo/urdf/simple_arm.xacro"
    robot_description = xacro.process_file(urdf_file).toxml()

    return LaunchDescription([
        Node(
            package='controller_manager',
            executable='ros2_control_node',
            parameters=[
                {'robot_description': robot_description},
                'src/urdf_cpp_demo/config/position_controllers.yaml'
            ],
            output='screen'
        ),
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            parameters=[{'robot_description': robot_description}],
            output='screen'
        ),
```

```
    Node(
        package='rviz2',
        executable='rviz2',
        output='screen'
    )
])
```

---

5. Build & Run

colcon build
. install/setup.bash
ros2 launch urdf_cpp_demo control.launch.py

---

6. Activate the Controller

In a new terminal:

ros2 control list_controllers
ros2 control load_controller --set-state start joint_state_broadcaster
ros2 control load_controller --set-state start joint1_position_controller

---

7. Send Joint Command

ros2 topic pub /joint1_position_controller/commands std_msgs/msg/Float64MultiArray "data: [1.0]"

The arm should rotate in RViz.

Next Steps

- Add real hardware plugin for STM32 or ESP32 control
- Move to `Gazebo Classic` or `Gazebo Harmonic`

- Add 2nd and 3rd joints and control them
- Connect to custom `C++ Node` for command generation

ROS 2 + Gazebo Classic Simulation for URDF

You'll simulate the same `simple_arm` in **Gazebo Classic** using `gazebo_ros2_control`.

---

1. Install Gazebo Classic for ROS 2 Humble

sudo apt install ros-humble-gazebo-ros-pkgs

---

2. Add Gazebo Plugins to `simple_arm.xacro`

At the **bottom of your `<robot>` tag**, insert:

<gazebo>

<plugin name="gazebo_ros_control" filename="libgazebo_ros2_control.so"/>

</gazebo>

Then, inside **each link** that moves (like `link1`), add:

<gazebo reference="link1">

<material>Gazebo/Blue</material>

</gazebo>

Optional: Add gravity or damping as needed.

---

3. Add Gazebo Launch File

Create: `launch/gazebo.launch.py`

```
from launch import LaunchDescription

from launch.actions import IncludeLaunchDescription

from launch.launch_description_sources import PythonLaunchDescriptionSource

from launch_ros.actions import Node

import os

import xacro


def generate_launch_description():

    urdf_file = 'src/urdf_cpp_demo/urdf/simple_arm.xacro'

    world_file = 'src/urdf_cpp_demo/worlds/empty.world'

    robot_description = xacro.process_file(urdf_file).toxml()


    gazebo_launch = IncludeLaunchDescription(

        PythonLaunchDescriptionSource([

            os.path.join(

                get_package_share_directory('gazebo_ros'),

                'launch',

                'gazebo.launch.py'

            )

        ]),

        launch_arguments={'world': world_file}.items()

    )
```

```
return LaunchDescription([
    gazebo_launch,
    Node(
        package='controller_manager',
        executable='ros2_control_node',
        parameters=[
            {'robot_description': robot_description},
            'src/urdf_cpp_demo/config/position_controllers.yaml'
        ],
        output='screen'
    ),
    Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        parameters=[{'robot_description': robot_description}],
        output='screen'
    ),
])
```

---

4. Create World File

Create folder and file:

mkdir -p urdf_cpp_demo/worlds

File: `worlds/empty.world`

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
  </world>
</sdf>
```

---

5. Build and Launch

```
colcon build
. install/setup.bash
ros2 launch urdf_cpp_demo gazebo.launch.py
```

You’ll see:

- Gazebo opens with ground
- Your arm is spawned

- Ready for commands

---

6. Control the Robot in Gazebo

Use:

ros2 control load_controller --set-state start joint_state_broadcaster

ros2 control load_controller --set-state start joint1_position_controller

ros2 topic pub /joint1_position_controller/commands std_msgs/msg/Float64MultiArray "data: [1.5]"

The robot in Gazebo will move.

Next Practice

- Add a second joint and control both
- Add collision boxes and mass to links
- Add damping and limits to joints
- Build your own world with obstacles
- Add sensors (IMU, Camera, Lidar)