Migrate to ROS2

August 21, 2024

1 Why Migrating to ROS2

Ubuntu 20, the last version that supports ROS1, is going to retire in Mid 2025. Even though many robots support both ROS1 and ROS2 on 2024, many companies, such as Neuromeka, said that they will put most of their energy on maintaining their ROS2 version robot project instead of the old one. Most likely, the ROS1 version robots will not receive frequent updates from comercial companies and if there are any issues with ROS1 version repository, they are less likely to support (and in the worst case not support since 2025). Thus, if robots were bought from comercial companies, it

Migrating a package from ROS1 to ROS2 involves updating the build system and dependencies, which can be complex due to differences in package naming, libraries, and conventions. Here's a detailed guide to help you with this migration process, including documentation and package mappings:

2 Documentation and Resources

1. **ROS1 to ROS2 Migration Guide**
   - **ROS2 Humble**: `https://docs.ros.org/en/humble/How-To-Guides/Migrating-from-ROS1.html`
2. **ROS2 Build System Documentation**
   - **ament_cmake**: `https://docs.ros.org/en/humble/How-To-Guides/Ament-CMake-Documentation.html`
   - **colcon**: `https://colcon.readthedocs.io/en/released/`

Common Package Mappings (May have some errors, not garentee)

ROS1 Package	ROS2 Package	Notes
roscpp	rclcpp	ROS2 uses `rclcpp` for C++ client libraries.
rospy	rclpy	ROS2 uses `rclpy` for Python client libraries.
std_msgs	std_msgs	The package name remains the same, but the build and usage might differ.
geometry_msgs	geometry_msgs	The package name remains the same.
tf	tf2	ROS2 uses `tf2` and `tf2_ros` for transform operations.
tf_conversions	tf2_geometry_msgs	Provides conversions for geometry messages.
std_srvs	std_srvs	The package name remains the same.
actionlib	action_msgs	For actions, ROS2 uses `action_msgs` and `rclcpp_action`.
eigen_conversions	eigen3	Use `eigen3` for Eigen support in ROS2.
kdl_parser	kdl_parser	The package name remains the same.
sensor_msgs	sensor_msgs	The package name remains the same.
cmake_modules	Custom	ROS2 doesn't have a direct equivalent; use ROS2-specific CMake modules if needed.
realtime_tools	realtime_tools	The package name remains the same.
rosbag	rosbag2	ROS2 uses `rosbag2` for bag file operations.

Example Conversion Steps

1. Change `find_package` Directives:
    - Replace `find_package(catkin REQUIRED COMPONENTS ...)` with `find_package(ament_cmake REQUIRED)` and `find_package()` for each specific dependency.
2. Update `catkin_package` to `ament_package`:
    - Replace `catkin_package()` with `ament_package()` and adjust its arguments.
3. Replace `add_library` and `add_executable`:
    - Update `add_library` and `add_executable` to use `ament_target_dependencies()` for linking libraries and dependencies.
4. Adjust Install Rules:
    - Modify the `install` commands to use ROS2 conventions.
5. Handle Messages and Services:
    - Use `rosidl_generate_interfaces()` for message and service files in ROS2.
6. Handle Dynamic Reconfigure:
    - Implement parameter handling as `dynamic_reconfigure` is not available in ROS2.

Example Conversion

ROS2 CMakeLists.txt after conversion

```
#1.Make sure you use at least VERSION 3.5 of cmake for ROS2
cmake_minimum_required(VERSION 3.5)
project(nac)

# Note that ROS2 use newer version of C++
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

#2.Change the package name. Line by line please :O.
find_package(ament_cmake REQUIRED) #catkin -> ament_cmake
find_package(rclcpp REQUIRED)      #roscpp -> rclcpp
find_package(tf2 REQUIRED)         #tf -> tf2
find_package(tf2_ros REQUIRED)

#Those are the same from ROS1 to ROS2.
find_package(std_msgs REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(sensor_msgs REQUIRED)
find_package(Eigen3 REQUIRED)
find_package(realtime_tools REQUIRED)

#3.Include directories. USE CAPITAL LETTERS!!! USE CAPITAL
    LETTERS!!! USE CAPITAL LETTERS!!!
include_directories(
  include
  ${EIGEN3_INCLUDE_DIRS}  # Eigen3 headers are included
      similarly in ROS1 and ROS2. USE CAPITAL LETTERS!!!
)

#4.Declare a C++ library, linking it against the ROS2
    dependencies.
add_library(${PROJECT_NAME} src/nac_nn_two.cpp)
ament_target_dependencies(${PROJECT_NAME}
  rclcpp                #Changed
  std_msgs
  geometry_msgs
  tf2                   #Changed
  tf2_ros               #Changed
  sensor_msgs
  Eigen3
  realtime_tools
)
```

```
#5. Optionally declare a C++ executable, showing how
    executables are managed in ROS2.
# add_executable(${PROJECT_NAME}_node src/nac_node.cpp)
# ament_target_dependencies(${PROJECT_NAME}_node
#   rclcpp              #Changed
#   std_msgs
#   geometry_msgs
#   tf2                 #Changed
#   tf2_ros             #Changed
#   sensor_msgs
#   Eigen3
#   realtime_tools
# )
# target_link_libraries(${PROJECT_NAME}_node ${PROJECT_NAME
    })

#5. Installation rules for libraries and executables,
    similar to ROS1 but using ROS2's ament_cmake.
install(TARGETS ${PROJECT_NAME}
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION bin
)
install(DIRECTORY include/${PROJECT_NAME}/
  DESTINATION include/${PROJECT_NAME}
  FILES_MATCHING PATTERN "*.h"
)

# Uncomment below lines for adding tests using ament in ROS2
    .
# ament_add_gtest(${PROJECT_NAME}-test test/test_nac.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${
    PROJECT_NAME})
# endif()

#6. Registers the package with ament, essential for ROS2
    packages.
ament_package()
```

Example XML Code

```
<?xml version="1.0"?>
<package format="3">
  <name>nac</name>
  <version>0.0.0</version>
  <description>The nac package</description>

  <!-- One maintainer tag required, multiple allowed, one
      person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</
      maintainer> -->
  <maintainer email="jordan@todo.todo">jordan</maintainer>

  <!-- One license tag required, multiple allowed, one
      license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3,
      LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but multiple are allowed, one
      per tag -->
  <!-- Optional attribute type can be: website, bugtracker,
      or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/nac</url> -->

  <!-- Author tags are optional, multiple are allowed, one
      per tag -->
  <!-- Authors do not have to be maintainers, but could be
      -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author
      > -->

  <!-- The depend tags are used to specify dependencies -->
  <!-- Dependencies can be ament packages or system
      dependencies -->
  <!-- Use depend for both build and exec dependencies -->
  <!-- Use build_depend for compile-time dependencies -->
  <!-- Use exec_depend for runtime dependencies -->
  <!-- Use test_depend for testing dependencies -->
  <!-- Use doc_depend for documentation dependencies -->

  <!-- Build tool dependencies -->
  <buildtool_depend>ament_cmake</buildtool_depend>
```

```
  <!-- Build dependencies -->
  <build_depend>rclcpp</build_depend>
  <build_depend>rclpy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>realtime_tools</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>eigen3_cmake_module</build_depend>
  <build_depend>octocan</build_depend>

  <!-- Exported build dependencies -->
  <build_export_depend>rclcpp</build_export_depend>
  <build_export_depend>rclpy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>octocan</build_export_depend>

  <!-- Execution dependencies -->
  <exec_depend>rclcpp</exec_depend>
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>realtime_tools</exec_depend>
  <exec_depend>sensor_msgs</exec_depend>
  <exec_depend>eigen3_cmake_module</exec_depend>
  <exec_depend>octocan</exec_depend>

  <!-- ROS2 doesn't have an equivalent to ROS1's export tag
     for message generation -->
  <export>
    <!-- Any other export information can go here -->
  </export>
</package>
```

Example C++ Code

```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "re2_flexiforce/msg/nac_input.hpp"
#include "Eigen/Dense"

using std::placeholders::_1;

class NACNode : public rclcpp::Node {
public:
    NACNode() : Node("nac_node") {
        setupMatrices();

        // Publishers
        joint_command_publisher = this->create_publisher<
            sensor_msgs::msg::JointState>("/command/
            joint_state", 10);

        // Subscribers
        cart_subscriber = this->create_subscription<
            re2_flexiforce::msg::NACInput>(
            "/nac_input", 10, std::bind(&NACNode::
                cart_callback, this, _1));

        // Timer
        timer_ = this->create_wall_timer(
            std::chrono::milliseconds(3), std::bind(&NACNode
                ::update_control, this));
    }

private:
    void setupMatrices() {
        // Setup matrices and neural network parameters
    }

    void cart_callback(const re2_flexiforce::msg::NACInput::
        SharedPtr msg) {
        // Handle incoming cart data
    }

    void update_control() {
        // Update control and publish commands
    }

    rclcpp::Publisher<sensor_msgs::msg::JointState>::
        SharedPtr joint_command_publisher;
    rclcpp::Subscription<re2_flexiforce::msg::NACInput>::
        SharedPtr cart_subscriber;
```

```
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char** argv) {
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<NACNode>());
    rclcpp::shutdown();
    return 0;
}
```

3 What's changed? What's not changed?

Some libraries have changed. Many stuff are merged into rclcpp built-in libraries. And ROS2 no longer uses cmake but ament_cmake. C++ and cmake are using newer versions. The way of initiating nodes has changed. It's inherited from Node now.

Since catkin is abandoned, catkin_make now doesn't work. Instead we use colcon build.

4 Warning

Most online ROS1 and ROS2 conversion tools are used for Ubuntu 20, which both ROS1 and ROS2 can be used. However, for Ubuntu 22 and beyond, the conversion tools are not guaranteed.