ROS-Industrial Universal Robot Package: Composition, Usage, and UR3 Example Application

Bum Yong Park
Kumoh National Institute of Technology

July 2025

Abstract

Recent advancements in robotics have expanded the applications of industrial robots, with Universal Robots' collaborative robots gaining prominence in manufacturing, service, and research domains. The Robot Operating System (ROS) and its industrial extension, ROS-Industrial (ROS-I), provide robust frameworks for developing robotic systems. The `universal_robot` meta-package supports Universal Robots' manipulators, including the UR3, through drivers and tools for real-time control and simulation. This paper details the composition and usage of the `universal_robot` package, focusing on its application to the UR3. We present a practical example from the `ur_arm_test` package, specifically `test_ur_position_moveit.cpp`, which demonstrates Cartesian space motion planning using MoveIt!. The example showcases incremental z-axis movements within a safe workspace, providing a foundation for industrial applications. Other programs from the package are briefly compared to highlight different control approaches. This work aims to guide developers in leveraging ROS-I for UR3-based automation.

1 Introduction

The Robot Operating System (ROS) is an open-source framework that simplifies robotic system development through hardware abstraction, device drivers, and advanced algorithms [6]. ROS-Industrial (ROS-I) extends ROS to industrial settings, enabling integration with manipulators like Universal Robots' UR3 [3]. The `universal_robot` meta-package offers tools for communication, motion planning, and simulation [4]. In recent years, robotic technology development has been accelerating rapidly, with significant advancements particularly in industrial robotics, which has traditionally dominated the robot market. While industrial robots were primarily used in manufacturing environments, the miniaturization and increased safety features of robots have expanded their applications across various fields. Among these industrial robots, Universal Robots' collaborative robots are widely utilized not only in manufacturing and service sectors but also for research purposes. Universal Robots continuously provides ROS-compatible drivers to enable diverse applications of their robots. Consequently, professionals from various fields seeking to utilize industrial robots are developing robotic systems using Universal Robots through both actual hardware and simulation environments. However, comprehensive guides for such development remain scarce. This paper explores the package's composition, installation, and usage, focusing on the UR3, and includes an example from the user-provided `ur_arm_test`[8] package to demonstrate Cartesian motion planning with MoveIt!. Through detailed analysis and practical examples, this work aims to provide a systematic guide for researchers and engineers looking to leverage ROS-Industrial tools for Universal Robot integration.

2 Composition of the Universal Robot Package

The `universal_robot` meta-package is a collection of ROS packages designed for Universal Robots' manipulators, including the UR3 [4]. Key components include:

- **ur_description**: Provides URDF and xacro files for the UR's kinematic and visual models, used in RViz and Gazebo.
- **ur*_moveit_config**: Supplies MoveIt! configurations for the UR (e.g., `ur3_moveit_config`) for motion planning.
- **ur_gazebo**: Enables Gazebo simulation for UR testing.
- **ur_kinematics**: Offers kinematic solvers for UR forward and inverse kinematics.

These components enable developers to interface with the UR3, plan motions, and simulate behaviors, supporting both research and industrial applications [2].

3 Installation and Usage

The `universal_robot` package supports ROS 1 (e.g., Noetic) and is transitioning to ROS 2 (e.g., Humble) [4]. Installation can be done via binary packages or source compilation [1].

3.1 Installation via Binary Packages

For ROS Noetic on Ubuntu 20.04, install the package using:

```
sudo apt-get update
sudo apt-get install ros-noetic-universal-robots
```

3.2 Installation from Source

For the latest features, clone the repository into a Catkin workspace:

```
cd ~/catkin_ws/src
git clone -b noetic-devel https://github.com/ros-industrial/
    universal_robot.git
cd ~/catkin_ws
rosdep install --from-paths src --ignore-src --rosdistro noetic
catkin_make
source devel/setup.bash
```

3.3 Usage with UR3 Hardware

To control a real UR3, configure the network and launch the driver and MoveIt!:

```
roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=IP_OF_THE_ROBOT
roslaunch ur3_moveit_config moveit_planning_execution.launch
```

Replace `IP_OF_THE_ROBOT` with the UR3 controller's IP address.

3.4 Usage with Gazebo Simulation

For simulation, launch the Gazebo environment:

```
roslaunch ur_gazebo ur3_bringup.launch
roslaunch ur3_moveit_config moveit_planning_execution.launch sim:=true
```

4 Example Program: UR3 Cartesian Motion Planning with MoveIt!

We present a C++-based ROS node from the `ur_arm_test` package, `test_ur_position_moveit.cpp`, which commands a UR3 robot to perform incremental z-axis movements in Cartesian space using MoveIt!. The program ensures movements stay within a safe workspace.

```cpp
#include <ros/ros.h>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit/planning_scene_interface/planning_scene_interface.h>
#include <moveit_msgs/DisplayTrajectory.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "ur_moveit_custom_position");
    ros::NodeHandle nh;
    ros::AsyncSpinner spinner(1);
    spinner.start();
    ros::Duration(2.0).sleep();

    moveit::planning_interface::MoveGroupInterface group("manipulator")
        ;
    moveit::planning_interface::PlanningSceneInterface
        planning_scene_interface;
    ros::Publisher display_publisher = nh.advertise<moveit_msgs::
        DisplayTrajectory>("/move_group/display_planned_path", 1, true);
    moveit_msgs::DisplayTrajectory display_trajectory;
    ROS_INFO("Reference frame: %s", group.getEndEffector().c_str());

    geometry_msgs::Pose target_pose;

    ros::Rate rate(10);
    int loop_num = 10;
    const double z_step = 0.01;
    const double z_upper = 0.5;
    const double z_lower = 0.1;

    while (ros::ok())
    {
        geometry_msgs::Pose current_pose = group.getCurrentPose().pose;
        target_pose = current_pose;
        ROS_INFO("Current z: %f", current_pose.position.z);

        target_pose.position.z += z_step;
        ROS_INFO("Target z: %f", target_pose.position.z);

        if (target_pose.position.z > z_upper)
        {
            ROS_WARN("Target z is above safe workspace. Stopping.");
            break;
        }
        if (target_pose.position.z < z_lower)
        {
            ROS_WARN("Target z is below safe workspace. Stopping.");
            break;
        }

        group.setPoseTarget(target_pose);
        moveit::planning_interface::MoveGroupInterface::Plan my_plan;
        if (group.plan(my_plan) == moveit::core::MoveItErrorCode::
            SUCCESS)
        {
            auto move_result = group.move();
            if (!move_result)
            {
                ROS_ERROR("Move execution failed!");
            }
        }
        else
        {
            ROS_WARN("Planning failed. Skipping this step.");
        }

        rate.sleep();
        loop_num -= 1;
        if (loop_num == 0)
        {
            ROS_INFO("Loop is finished!!");
            break;
        }
    }

    ros::shutdown();
}
```

4.1 Program Explanation

The program, `test_ur_position_moveit.cpp`, implements a ROS node for controlling a UR3 robot in Cartesian space using MoveIt!. It performs the following steps:

1. **Node Initialization**: Initializes a ROS node named `ur_moveit_custom_position` and sets up an asynchronous spinner for handling callbacks.
2. **MoveIt! Setup**: Configures the `manipulator` group for the UR3 and a publisher for displaying planned trajectories in RViz.
3. **Cartesian Motion Planning**: Retrieves the current end-effector pose, increments the z-coordinate by 0.01 meters per step, and plans a trajectory to the new pose. The program runs for 10 iterations or until the z-coordinate exceeds the safe workspace (0.1 to 0.5 meters).
4. **Safety Checks**: Ensures the target z-coordinate stays within the UR3's safe workspace, stopping if it exceeds defined limits.
5. **Execution and Error Handling**: Executes the planned trajectory if planning succeeds, logging errors if planning or execution fails.

To run this program, ensure the UR3 (real or simulated) and MoveIt! are launched as described in Section 3.3 or 3.4. Compile the `ur_arm_test` package in a Catkin workspace, then run:

```
rosrun ur_arm_test test_ur_position_moveit
```

5 Discussion

The universal_robot package simplifies UR3 integration into ROS-based systems, supporting rapid prototyping and deployment for diverse applications [5]. The test_ur_position_moveit.cpp program demonstrates Cartesian space control using MoveIt!, ideal for tasks requiring precise end-effector positioning, such as pick-and-place or assembly tasks. Other programs in the ur_arm_test package offer complementary approaches:

- test_ur_joint_traj_action.cpp: Uses actionlib for synchronous joint trajectory control, suitable for precise joint-space movements.
- test_ur_joint_trajectory_publisher_node.cpp: Publishes joint trajectories directly, offering simplicity but less robustness for complex tasks.
- test_ur_joint_traj_realtime.cpp: Implements real-time joint trajectory updates, useful for dynamic applications requiring continuous adjustments.
- test_ur_joint_moveit.cpp: Similar to test_ur_position_moveit.cpp, it uses MoveIt! for Cartesian control, indicating some redundancy in the package.

While MoveIt! provides high-level motion planning, challenges include the complexity of configuration and the need for precise workspace calibration to avoid singularities or collisions. The ur_arm_test package highlights the versatility of UR3 control, but future enhancements could include multi-point trajectories, integration with vision systems, or ROS 2 support for improved real-time performance [7].

6 Conclusion

This paper provided a systematic overview of the ROS-Industrial universal_robot metapackage, emphasizing its role in facilitating UR3-based robotic systems for manufacturing, service, and research applications. The example program, test_ur_position_moveit.cpp, demonstrated Cartesian motion planning with MoveIt!, offering a practical approach for precise UR3 control. By leveraging ROS-I and MoveIt!, developers can efficiently develop and test robotic applications, bridging the gap between research and industry. Future work could explore advanced features, such as adaptive control or integration with ROS 2, to further enhance the capabilities of UR3-based systems.

References

[1] ROS-Industrial Universal Robots support, https://wiki.ros.org/universal_robot, 2021.

[2] Getting Started with a Universal Robot and ROS-Industrial, https://wiki.ros.org/universal_robot/Tutorials/Getting%20Started%20with%20a%20Universal%20Robot%20and%20ROS-Industrial, 2020.

[3] ROS-Industrial, https://rosindustrial.org, 2022.

[4] universal_robots - ROS Wiki, https://wiki.ros.org/universal_robots, 2019.

[5] Universal Robots ROS Driver, https://github.com/UniversalRobots/Universal_Robots_ROS_Driver, 2022.

[6] ROS: Home, https://www.ros.org, 2024.

[7] The Robot Operating System (ROS1 & 2): Programming Paradigms and Deployment, `https://link.springer.com`, 2022.

[8] UR3 ROS Test, `https://github.com/IRaC-Lab/UR3_ROS_Test.git`, 2025.