# A ROS Industrial Driver for ABB Robots Using the IRC5 and IRC5Compact

## Introduction

Robot Operating System (ROS) is a popular framework for robotic development in the research community. ROS encapsulates algorithms as nodes, which pass information to each other through sockets as messages. The use of modular nodes makes it easy to add functionality to the robot without adding complexity. Standardization of messages within ROS makes it easy to swap nodes for other nodes that perform similar functions. For example, ROS provides several navigation nodes, each implementing a different algorithm. Because the message interface is standardized across these nodes, they are “drop-in” replacements, which makes it easy to experiment with different algorithms without changing any code in the rest of the robot’s software. ROS also has a vast library of existing nodes and algorithms, allowing researchers to leverage prior work without having to reimplement the algorithms.

Although ROS is widely used in research robotics, it is virtually unheard-of in industry. Most industrial robots are programmed in proprietary languages with rather limited interfaces. If an industrial robot user wants to port an algorithm or task to a new robot, they must rewrite the entire program in the new language, which often has only a partial subset of the features of a modern programming language. The Southwest Research Institute (SWRI) has begun development on a series of ROS stacks called ROS industrial. The purpose of ROS Industrial is to provide a common interface to industrial robots by many manufacturers and allow Industrial users to leverage the large existing codebase of ROS.

SWRI has successfully demonstrated their ROS Industrial platform with a Motoman dx100 industrial robot controller, but support for robots from other manufacturers and robots has been slow to arrive. As part of a masters project at Case Western Reserve University, a ROS Industrial driver for an ABB IRB-120 being controlled by an IRC5 Compact was developed. In addition, a new ROS Industrial message and message handler were created to handle state and diagnostic information from the robot controller and publish them in ROS.

## Architecture

This driver’s architecture was primarily dictated by the structures of ROS and ROS Industrial and by the limitations of the RAPID software system on the IRC5 Compact.

ROS’s arm navigation stack consists of a series of processes encapsulated in nodes. The arm path planning node publishes ROS JointTrajectory messages. These messages contain a sequence of points in a trajectory, represented by the angles of the robot joints to achieve each of the points. These trajectory messages must be fed into a simulator or into robot hardware. ROS arm navigation also expects feedback on the robot’s current state in the form of ROS JointState messages. These messages contain the current angular position, velocity, and acceleration of the arm’s joints.

ROS Industrial sits “on top of” the hardware drivers for the industrial robots it supports, creating several standard messages and processes for communication with industrial robots. A ROS industrial driver consists of three ROS nodes. The first node publishes a model of the robot loaded from a Universal Robot Description File (URDF). The second node gets physical state and software status information from the robot controller over a network connection and publishes it in ROS using the JointState message and other appropriate ROS message types. The third node gets JointTrajectory messages from ROS and sends them to the robot controller over a network connection.

The first part of the ROS driver, the URDF node, already exists in ROS, but it was necessary to create the URDF description of the robot. Using 3D CAD files of the IRB-120, and URDF file was created using MathWorks’s SimMechanics plugin for AutoDesk Inventor and David Lu’ SimMechanics to URDF script. This outputs a simple URDF of the robot arm, defining all of the links and joints of the arm.

The second part of the ROS driver, the state feedback node, was based heavily on existing ROS Industrial feedback nodes. ROS Industrial communicates with robot controllers over TCP or UDP sockets using messages defined by SWRI. The messages are defined as serializable C++ classes and consist of a simple header and a data payload. The header contains the message type, communication type, and a reply code. The data payload contains the data of the message, serialized into a byte array. SWRI had already defined a message type for joint state feedback. A new message type was defined for robot controller status, based on SWRI’s existing development roadmap. This ROS node is a TCP client that connects to a TCP server, written in RAPID, running on the IRC5 Compact. The RAPID server periodically sends serialized ROS Industrial joint messages and serialized ROS Industrial robot status packages to the ROS node. The ROS node has two message handlers, one for each message. The message handler for the joint state deserializes the ROS Industrial joint state message and uses it to populate a ROS JointState message, which it them publishes on the ROS topic /joint\_states. The message handler for the robot status deserializes the ROS Industrial robot status message and uses it to update the robot status in the ROS diagnostic system. This handler was written completely from scratch for this driver; no existing ROS industrial robot drivers provide this functionality. The new message and handler for robot status should be reusable in other ROS industrial robot drivers.

The third part of the ROS driver, the trajectory sender node, was based heavily on existing ROS Industrial trajectory sender nodes. SWRI had already defined a message type for joint trajectory points. This ROS node is a TCP client that connects to a TCP server, written in ROS, running on the IRC5 Compact. The node consists of a callback function that waits for a ROS JointTrajectory message to be published on the ROS topic /command. When a JointTrajectory message is received, the ROS node breaks it up into ROS Industrial joint trajectory point messages, serializes them, and sends them to the IRC5 Compact. The RAPID server on the IRC5Compact.

In addition to the RAPID server modules and a global configuration module, there is a motion control module, containing a motion control process. This requires three processes (two servers and a motion control process) to run in parallel using the ABB’s Multitasking option. The servers are static processes, and only the motion control process issues motion commands. When the RAPID trajectory server finishes receiving a trajectory from the ROS client, it copies the trajectory into a global trajectory variable, using a rudimentary semaphore system implemented with a global variable to prevent race conditions with the motion process. It then sets a flag in the semaphore. The motion module has the current trajectory stored in a local variable. The main motion process consists of an infinite loop. In every iteration of the loop, it runs a RAPID MoveJ command to move to the next point in the trajectory, then checks if the semaphore flag is set indicating that there is a new trajectory available in the global variable. If the flag is set, it copies the global trajectory into the local trajectory variable and resets the local trajectory pointer to the beginning of the trajectory. Again, the rudimentary semaphore system is used to prevent race conditions with the server module. The semaphore flag is then cleared.

## Using this Driver

Use of this driver requires an ABB IRC5 or IRC5 compact with the Multitasking option. There are existing URDF files in ROS Industrial for the IRB-5400 and IRB-120; to use a different ABB robot, the user must define their own URDF file, using the same joint name and link name conventions used in these files.

Install the RAPID modules on the IRC5

Run the launch file

## Future Work

Publishing robot status information to topics in addition to diagnostics

Use real RAPID interrupts