ROS Industrial and the ABB IRB-120 Interface

Wyatt Newman
March, 2015

Background: ROS Industrial (http://rosindustrial.org) was founded as a consortium of robot vendors and users to help spur growth of a wider variety of robot applications. The approach is to link open-source ROS code with industrial robots.

The original repository (hosted on github at: https://github.com/ros-industrial) was created in 2012 by Shaun Edwards--a CWRU alum and research engineer at SouthWest Research Institute. The first ROS-I interface to an ABB IRB-120 robot was written by Ed Venator during his M.S. research at CWRU ("A Low-Cost Mobile Manipulator for Industrial and Research Applications", August, 2013). Ed subsequently joined SWRI, where he continued his work with ROS-Industrial.

Industrial robot controllers are typically closed systems, and these are programmed with proprietary robot programming languages. These languages lack the generality of ROS, making it difficult to integrate complex systems, including perceptual processing and graphical user interfaces. However, if an industrial robot controller can be bridged to ROS effectively, then it can exploit the available libraries within ROS in general.

At present, there are still few existing bridges from ROS to industrial robots, and these bridges are fairly limited. The baseline interface to an industrial robot consists of transmitting a joint-space trajectory to the robot, then having the robot execute the downloaded trajectory. This requires that the native robot controller run an interface program that is able to receive, interpret and execute downloaded trajectories. This interface program is written in the native programming language of the industrial robot and, preferably, interacts with ROS applications via Ethernet sockets.

ABB IRB-120 ROS Interface: For our ABB IRB-120 robot, three programs (created by Ed Venator) written in ABB's language "RAPID" are run on the robot controller. (See: http://wiki.ros.org/abb/Tutorials. The programs may be found here: https://github.com/ros-industrial/abb/blob/hydro-devel/abb_common/rapid/ROS_motionServer.mod). These RAPID programs listen for incoming trajectories via a TCP/IP socket, invoke motion of the transmitted trajectories, and send out robot state data via the TCP/IP socket. The RAPID robot interface programs are set up on our system, ABBY, to launch automatically when the ABB IRC5 controller is powered up in "Auto" mode. (Installation instructions may be found here: http://wiki.ros.org/abb/Tutorials).

Trajectories sent to the IRB120 robot controller consist of a sequence of joint "poses", where each pose consists of 6 joint-angle values, corresponding to the 6 axes of the IRB-120. The maximum number of poses in a trajectory is limited in the RAPID code (which appears to default 100 points, although the IRC5 ABB controller can handle longer trajectories). Poses are encoded with a parameter to mark the first and last poses of a trajectory. When the last pose of a trajectory is received by the ABB controller, the entire trajectory is executed.

The user is responsible for making sure that transmitted trajectories make sense—including obeying joint limits, joint velocity limits, maximum number of poses in the trajectory, and—most important and most difficult--that the resulting motion will not cause damage to the robot or its environment. User-written ROS code must perform the joint-space planning that produces safe and feasible joint trajectories.

ROS-side Trajectory Interface: On the ROS host computer(s), there are three bridge nodes that relay commands to the robot and sensed states back to the ROS environment. Quoting from Ed Venator's thesis:

> "The robotic arm driver, like the mobile base driver, consists of two ROS nodes that communicate with a server running on the IRC5 robot controller. ROS trajectory messages describe the trajectory of a robotic arm as a series of points, with each point describing the position and velocity of all of the robot's joints. One of the ROS nodes subscribes to ROS trajectory messages, breaks them up into packets, and sends them to the IRC5 controller over TCP using a standard packet structure defined by SWRI. The other ROS node connects to the IRC5 controller over TCP and listens for state information from the controller, which is sent using another packet structure defined by SWRI. It publishes this state information, consisting of all of the robot's joint angles, as ROS joint state messages and ROS joint trajectory feedback messages."

Specifically, the ROS bridge nodes are launched using the ROS-Industrial launch file:
industrial_robot_client/launch/robot_interface_download.launch
which is launched with the option:
<arg name="robot_ip" value="$(arg irc5_ip)"/>
where "irc5_ip" is (for ABBY) specified in our launch file as:
<arg name="irc5_ip" default="192.168.0.50" />

As of this writing, it appears that the ROS-Industrial wiki needs to be updated. The wiki currently states: (http://wiki.ros.org/industrial_robot_client/design)
"Expect two main nodes: robot_state node and joint_trajectory node."

However, the current industrial-robot client launch file at:
industrial_robot_client/launch/robot_interface_download.launch
starts up three nodes: robot_state, motion_download_interface, and joint_trajectory_action.

The first node, "robot_state", receives arm angles from the ABB controller and republishes these on the topic "joint_states."

The other two nodes provide two optional ways to relay trajectory commands from ROS to the robot. The node "motion_download_interface" subscribes to the topic ""joint_path_command", which carries messages of type "trajectory_msgs::JointTrajectory" (a standard ROS type). (See: https://github.com/ros-industrial/industrial_core/blob/indigo-devel/industrial_robot_client/src/joint_trajectory_interface.cpp). This node receives ROS joint-trajectory commands, translates them into a format suitable for transmission to the robot controller's motion service, and sends these out via TCP/IP. The tedious details of how this translation and communication are performed are encapsulated to hide them from the user.

The third node, "joint_trajectory_action", provides an alternative ROS "actionServer" interface (see wiki.ros.org/actionlib). The name of the action server provided by this node is "joint_trajectory_action." User ROS code can instantiate action clients that connect to this server then communicate desired trajectories as "goal" requests. The action server will provide a "response" message back to the action client when done. (see source code at: https://github.com/ros-industrial/industrial_core/blob/indigo-devel/industrial_robot_client/src/joint_trajectory_action.cpp). The action server re-uses the same functionality of the motion_download_interface node to translate ROS trajectory commands and

send these commands to the robot controller for execution. By abstracting this layer of communications, new industrial-robot interfaces can be added, largely re-using the existing ROS code.

Note that with the addition of the action-server interface, the ROS-Industrial package name “industrial_robot_client” may seem confusing. The joint_trajectory_action node is both a service and a client (and thus better described as a “broker”). This “bridge” node offers a service to ROS nodes (via a ROS interface), and at the same time, it is a client of the industrial-robot's motion service, via a custom TCP/IP socket interface.

Simulating the Robot Interface: The example package in our class repository “example_robot_interface” contains source code for a node “simu_motion_download_interface”, which subscribes to the topic "joint_path_command", which carries messages of type “trajectory_msgs::JointTrajectory.” This node thus takes the place of the ROS-Industrial motion_download_interface node. It is capable of receiving trajectory commands of the same style as the ROS-I interface, but it does not implement actual communications with a real robot. Instead, it merely republishes the joint angles to a Gazebo simulation. (A Gazebo simulation of the IRB120 is in progress; more on this later).