Introduction to action servers and clients: the Joint Trajectory Behavior

Wyatt Newman
September, 2014

So far, we have seen how to control Atlas joints at a very low level, as illustrated by the package "simple_joint_interface," and explained in the document "ROS and DRC." However, using this interface revealed undesirable behavior to step inputs, e.g. with fast motions inducing falling. Further, improper values sent to the low level could easily cause instability and/or damage to the physical robot.

For protection, an intermediate layer—the "low-level joint controller" (LLJC) was inserted, as described in the document "Writing Open-Loop Motion Scripts for Atlas." This layer accepts (or rejects) requests from "action servers" to be granted ownership/control of specified joints. However, the LLJC remains a broker between such action servers and Atlas, monitoring commands for acceptability (e.g., in range and incrementally close to current joint angles).

One of the existing "behaviors" (action servers) that interacts with the LLJC is the "Joint Trajectory Behavior" (in the package: joint_traj_behavior). This server waits for clients to connect to it and to send over "goal" instructions. A goal instruction can be as simple as a command to move a single joint to a single destination angle, or it may combine multiple, joint-coordinated moves through via points, interleaved with control of the right and left-hand fingers. Compliant-motion behaviors can also be commanded.

Previously, the "play_file" action client was introduced. This node is capable of parsing a YAML file that describes one or more desired moves, repackaging this information into the "goal" message format consistent with the trajectory behavior, and communication these goals to the joint trajectory behavior action server. This allowed one to write simple motion programs. A limitation, though, is that these programs were expressed as joint-space moves (which is not natural for a programmer), and the specified moves were hard-coded (not responsive to sensors or user inputs).

A more general interface to the existing foundation (LLJC and joint-trajectory behavior) is illustrated by the node "example_traj_client.cpp" in the package "example_traj_client." To run this code, the LLJC and joint-trajectory behavior must both be running, and "USER" mode must be set. This can be accomplished by executing steps 1) through 3) of section "launching nodes for executing play-file scripts" in the document "Writing open-loop motion scripts for Atlas." However, instead of step 4 (i.e., as an alternative to play_file), in another window, run:

```
rosrun example_traj_client example_traj_client
```

This program will prompt the user to enter a joint number (restricted to 22 through 27—the right-arm joints) or code 6 (close right hand) or code 7 (open right hand). The example program will use this input to populate a consistent "goal" message, then transmit this message to the trajectory behavior. The interface will seem similar to the earlier "simple_joint_interface", except the resulting motions will be smoothly interpolated with profiled velocity to complete each move command gently over 3 seconds.

More importantly, this program illustrates how one can generate motion commands programmatically. For example, motion commands might respond to user input, such as through a mouse, joystick, or rviz interactive marker. Or motion commands might be generated automatically based on sensory

interpretation.

When running example_traj_client, it can be revealing to also run (in another terminal):

```
rostopic echo /joint_traj_behavior/goal
```

This will display the entire goal message, as transmitted by the example client to the trajectory server.