Sensor Visualization and Interactive Marker Goals
Wyatt Newman Nov 12, 2013

Setting goals for Atlas can be done with respect to sensors. To do so interactively, one can move a marker within an "rviz" display and publish the corresponding pose as a goal location, e.g. for grasping. These notes describe recent software to enable interactive goal setting. Much of the tedium of roslaunching packages described below can be automated in launch files—but the present state of development uses some user input from the monitor, which is not consistent with launching groups of packages from one window.

Launch drcsim:
Bring up the simulator with the "cheats" enabled (so the torso can be pinned, optionally). There are a variety of interesting worlds that can be used. A world with valves is:
VRC_CHEATS_ENABLED=1 roslaunch roslaunch drcsim_gazebo drc_practice_task_7.launch

To reach the valves, move Atlas in front of the valve wall. To do so,

- first wait for the initialization (Atlas drops, then enters "dynamic standing" after about 10 sec).
- Select the up/down/left/right icon in the menu bar of the Gazebo display
- click on Atlas in the Gazebo display
- drag him to a location in front of the valve wall such that a valve is reachable
- click on the arrow icon on the menu bar of the Gazebo display, so you don't accidentally move Atlas again

If desired, pin the torso. From a terminal, enter:
rostopic pub --once /atlas/mode std_msgs/String '{data: "pinned_with_gravity"}'

Visualizing Sensors:
Bring up "rviz": rosrun rviz rviz. If your configuration is not saved, you will need to add multiple items to the display. To do so, press the "add" button on the left panel and do the following:

- choose RobotModel
- in the "displays" panel, item "fixed frame", from the drop-down, choose "utorso" as the fixed frame. The Atlas robot should now be visible in rviz
- add a "pointcloud2"
- under this new topic (in "displays" panel) select from the "topic" drop-down menu "multisense_sl/camera/points2".

This will bring up a display of 3-D points based on stereo vision from the robot's sensor head. If Atlas's head is pointed up, you can control the neck angle to angle 0.0, by bringing up the following.

Optionally, start the LIDAR spinning with the command:
rostopic pub --once /multisense_sl/set_spindle_speed std_msgs/Float64 '{data: 3.0}'
Then in rviz, "add", choose "laserScan" and this item will appear in the "displays" panel. Expend this item, and under "topic" choose "multisense_sl/laser/scan". Increase the "decay time" value to get persistence from the LIDAR pings.

Visualizing Interactive Markers:
One can interact with the visualization of sensor data via interactive markers. A useful example is run with:

- ***roslaunch grasping grasping.launch***

After starting this node, the corresponding marker can be displayed in rviz. To do so, press the “add” button and select InteractiveMarker” from the list. This new item will appear in the “displays” panel (on the left). Expand this item and next to “update topic”, choose: /grasp_goal/update. A red arrow will now appear in the sensor view (initially buried inside the robot's torso). You can move this arrow around interactively via the GUI provided (including 3-D translation and rotation). When the marker is moved, it publishes its “pose” (3-D origin and quaternion orientation) to the topic “grasp_goal”. If you invoke: ***rostopic echo grasp_goal,*** you will see the pose values change when you move the marker. These values are expressed in the fixed frame of the display (which was previously chosen to be “utorso”).

In addition to enabling the interactive marker, this launch file brings up a simple GUI with a “move” button. When this button is pressed, it publishes a trigger value to the topic “grasp_move_trigger”. This trigger signal is used to initiate a Jacobian-behavior based move.

Starting Atlas Low-Level Control:
A low-level joint controller currently under development utilizes some additional nodes for gravity-load estimates, inertia estimates and Jacobian computations (for whole-body damping). To run this low-level controller, do:

- ***rosrun lowLevelJointServoV2 lowLevelJointServoV2***
- start the gravity-load publisher with: ***rosrun mass_calcs stand_torque_calc***
- start the inertia calculator/publisher: ***rosrun mass_calcs H_calc***
- start the Jacobian publisher: ***rosrun jacobian_publisher jacobian_publisher***

Starting The Behavior Server and Playfile Client:
At this point, one can start up “action servers” that talk to the low-level joint controller to invoke specific behaviors. The “behavior server” is useful for running “playfile” scripts. To do so:

- start the behavior server: ***rosrun behavior_server behavior_server***
- as desired, execute a “playfile” script. E.g.,
  - cd to: elec7078/rosbuild/elec7078_newman/traj_files
  - ***rosrun play_file play_file prep_valve_grab.traj***

This will invoke a smooth joint-space move to a target pose, as prescribed in the named trajectory file. The example file will pre-position the right arm to anticipate turning a valve, as well as tilt the head down for a better view of the valve wall.

Starting the Jacobian Move Behavior and Client:
Another useful action server is the Jacobian Move behavior. This action server will connect with clients to perform Jacobian-based hand moves to prescribed poses. In the example code, this behavior is *NOT* complete. The Jacobian computations must be inserted into the code, as noted in the comments embedded in the code. To start this action server, do:

- ***rosrun jacobian_move_behavior rhand_jacobian_move_behavior***

A corresponding action client connects to this server and sends goals. Start the client with:

- ***rosrun jacobian_behavior_client_example rhand_jacobian_client_example***

This client will listen on the topic “grasp_goal” for marker poses. However, it will not ask

for a Jacobian move to a goal until the goal is approved. To approve a goal, press the "move" button from the GUI that was launched earlier via "grasping.launch".

Alternatively, you could enter the command line:
rostopic pub --once grasp_move_trigger std_msgs/UInt64 '{data: 1}'

to invoke a trigger, telling the client to use the most recent marker pose as a goal for the Jacobian move behavior.

To Do: Complete the Jacobian Move Behavior
At present, the Jacobian move behavior ("rhand_jacobian_move_behavior.cpp" in package "jacobian_move_behavior") is missing the most important part. This code does communicate with an action client to receive goal poses. However, look for the comments:
// REPLACE THIS WITH ACTUAL SET-UP: new pose, both position and orientation
and
// THIS IS WHERE YOU SHOULD PUT INCREMENTAL COMMANDS
and
// THIS IS WHERE TO TEST FOR CONDITION TO TERMINATE MOVE
The code, at present, increments joint angles for the first 3 joints of the right arm—which is not at all the intent. Instead, a Jacobian should be used to increment and populate values in lowLevelJointControllerCmd.jntAngles[], which is a message sent out to the low-level joint controller for execution.

The Jacobian used could be the Jacboian published by the jacobian_publisher. However, the hand frame used for this Jacobian is not what is desired. An alternative, more useful hand frame is defined by running:
rosrun grasp_tf grasp_tf_node

By running this node, two new frames are published with respect to the robot: l_grasp and r_grasp. These can be visualized in rviz via the "add" button and selecting "axes". Once the new item is added, in the "displays" panel, select the reference frame to be "l_grasp" or "r_grasp". A coordinate frame will be displayed showing this frame.

Although a Jacobian with respect to this frame is not being published, one can be computed. To see an example of how to do this, look for the object "vlad_fancy_solver_" in low_level_joint_servo_class.cpp of the package lowLevelJointServoV2. Instead of the line:
vlads_fancy_solver_->Init(std::tuple<std::string,std::string>("r_foot","pelvis"));
a solver can be instantiated that refers to "r_grasp" and "utorso". The Jacobian returned by this means will have joint ordering from the "start" frame to the "end" frame for all involved joints.