Vision and Grasp with Baxter

Wyatt Newman

Nov, 2015

A Kinect sensor and a Yale gripper (http://www.eng.yale.edu/grablab/openhand/) have been added to the Baxter robot at CWRU. Coordinating Baxter with these devices is described here.

Kinect sensor: A Microsoft Kinect sensor is mounted to Baxter's "head" link. At the time of this writing, the mechanical mount is imprecise and sensitive to disturbances. We anticipate upgrading to a robust mechanical mount and incorporating the higher-precision, more precise "Kinect-2" sensor. For the immediate future, however, we will be using the Kinect-1.

To start up the Kinect, run the following. Within a terminal, declare that the ROS master is running on the Baxter robot by entering the alias:

```
baxter_master
```

This must be done in each terminal that runs a ROS node.

In this window, launch the following:

```
roslaunch cwru_baxter_launch kinect.launch
```

This will start the Kinect driver, as well as a transform publisher that publishes the pose of the Kinect sensor with respect to the robot. It also remaps the Kinect publications to the topic /kinect/depth/points (making it compatible with code written for the Baxter gazebo simulation).

The launch file "kinect.launch" is relatively brief, containing:

```
<remap from="/camera/depth_registered/points" to="/kinect/depth/points" />
<include file="$(find freenect_launch)/launch/freenect.launch" />

<include file="$(find cwru_baxter_launch)/kinect_transform.launch" />
```

This file brings in the details of two other launch files: freenect.launch and kinect_transform.launch. The freenect.launch file is an open-source ROS driver for the Kinect camera. Data published by this node is on the topic "/camera/depth_registered/points". This topic is different from the topic used by our gazebo simulator, which is "/kinect/depth/points." With the launch-file line:

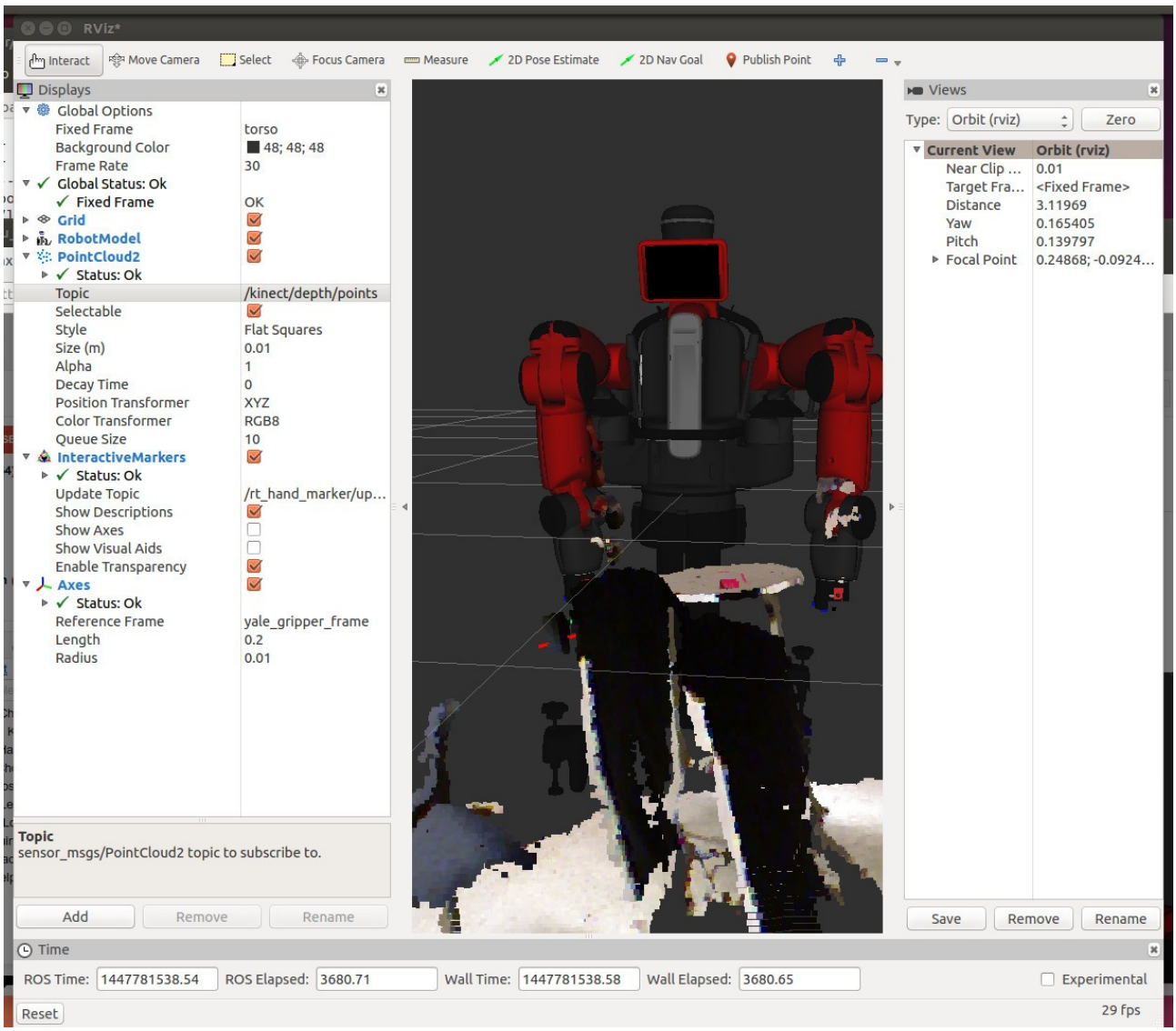
```
<remap from="/camera/depth_registered/points" to="/kinect/depth/points" />
```

the freenect topic is redirected to our preferred topic name, "/kinect/depth/points". Thus our code in cwru_pcl_utils is compatible with the freenect Kinect driver.

The launch file "kinect_transform.launch" in package "cwru_baxter_launch" contains the line:

```
<node pkg="tf" type="static_transform_publisher" name="camera_link_frame_bdcst" args="0.19
-0.00 0.08  -2.95 -4.1 -2.92   kinect_link camera_link 50"/></launch>
```

This invokes a transform publisher relating the camera_link frame to the kinect_link_frame, which is necessary to relate camera data in the sensor frame to a convenient robot frame (e.g. the "torso" frame). The data: 0.19 -0.00 0.08 -2.95 -4.1 -2.92 corresponds to x, y, z, roll, pitch, yaw values. These values are only approximate, and they should be further refined. With the present values, an rviz view of Baxter with Kinect pointcloud display is shown below:



The topic for the pointcloud2 item is “kinect/depth/points”, which is the desired topic per the “remap” operation in the launch file. The “color transformer” option is set to “RBG8”, so the pointcloud appears together with its associated color.

Note that some of the pointcloud points lie on parts of Baxter's arms. This effect can be exaggerated by moving Baxter's arms in from of the Kinect. If the Kinect transform is accurate, then the pointcloud data will be consistent with the robot model. The robot pose displayed in rviz is based on the sensed robot joint angles and computed forward kinematics. The Kinect points follow from a separate transformation. If the two agree (as evidenced by pointcloud points aligning with robot arm model surfaces), then the Kinect transform is well tuned.

The values of the Kinect transform (0.19 -0.00 0.08 -2.95 -4.1 -2.92) should be tuned such that pointcloud points are displayed consistent with surfaces on the robot model. When this alignment is adequate, then poses computed with respect to Kinect pointclouds can be used as goal poses for the robot's arms.

Yale Gripper Control: The Yale gripper has 4 fingers that are actuated by a single motor. The cable/pulley system allows the fingers to reconfigure to accommodate different shapes. (see http://www.eng.yale.edu/grablab/openhand/).

The Yale gripper contains a Dynamixel motor, model MX-64

(http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm).

Code to drive this motor resides in the package “baxter_gripper.” To motor driver can be started from a terminal as follows. First, set the remote roscore master with:

```
baxter_master
```

Then start the ROS motor driver with:

```
rosrun baxter_gripper dynamixel_motor_node
```

This code is still under development. You can ignore the “communication error” warnings (and, hopefully, this will get fixed). The gripper driver is now listening on topic “dynamixel_motor1_cmd” for position commands (in the range 0 to 4096 over one revolution), which will be passed along to the Dynamixel motor.

To command motor angles, a convenient test/diagnostic is:

```
rosrun baxter_gripper baxter_gripper_test
```

If all is well (including USB device plugged in, motor plugged into power distribution board, and power supply plugged in), then the gripper fingers will open and close sinusoidally at 0.1 Hz.

Alternatively, one can manually command the gripper open/close angle with a command-line publication. To command the value 3600, enter:

```
rostopic pub dynamixel_motor1_cmd std_msgs/Int16 3600
```

Under current conditions, the useful range of angle commands is approximately 3000 to 4000. Individual values can be tested to see what commands are appropriate for various objects. Subsequently, a ROS node can publish a std_msgs/Int16 message type on topic dynamixel_motor1_cmd to operate the gripper under program control.

At present, the Dynamixel control uses position commands to a position servo algorithm. If the gripper is commanded to close the to an angle (i.e. gripper opening) that is significantly smaller than an object between the fingers, the motor will hit a torque limit and shut down. More desirably, the motor should be commanded to exert a target torque, which will be reached when the object is grasped with a corresponding force. The address mappings in (http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm) may be interpreted and used in the code in the baxter_gripper package to achieve force-based gripping instead of position-based gripping, which would be a significant improvement.

The Yale hand is mounted to Baxter's tool flange at a radial offset. In performing inverse kinematics, one cares about a gripper frame, corresponding to an origin between the fingertips, a z-axis pointing out from the tool flange, and an x-axis pointing along a line between oppositve fingertips. This frame is the “yale_gripper_frame”. It is related to the “right_hand” frame (actually, the right tool-flange frame) via a transform publisher. To make this connection, run the following:

```
roslaunch cwru_baxter_launch yale_gripper_xform.launch
```

This publishes the values: <node pkg="tf" type="static_transform_publisher" name="tool_xform" args="-0.030 0 0.120 -0.24 0 0 right_hand yale_gripper_frame 50"/>

The intent of this transform is to relate the yale gripper frame (aligned with the fingertips) to Baxter's right-arm tool-flange frame (right_hand frame). The current numbers are only approximate, and they should be refined.

After launching the yale_gripper_xform launch file, a tf_listener can find transforms between the gripper frame and the torso frame, or between the gripper frame and the Kinect sensor frame.

The gripper frame can be visualized in rviz by adding an “axes” item and selecting “yale_gripper_frame” to display.

The tool transform can be found by any node using a transform listener. The action server:

```
rosrun baxter_cartesian_moves baxter_cart_move_as
```

services goals expressed as desired poses of the right gripper frame. In this package, the function:

```
bool ArmMotionInterface::rt_arm_plan_path_current_to_goal_pose()
```

converts a desired gripper frame to a corresponding right-arm tool-flange frame using:

```
goal_flange_affine_right_ = goal_gripper_affine_right_ * A_tool_wrt_flange_.inverse();
```

The tool transform can be obtained from a member function of the kinematics object, e.g. as:

```
A_tool_wrt_flange_ = baxter_fwd_solver_.get_affine_tool_wrt_flange();
```

The resulting tool-flange pose is used in Cartesian-motion planning within the node baxter_cartesian_moves/baxter_cart_move_as, e.g. within the member function:

```
path_is_valid_ = cartTrajPlanner_.cartesian_path_planner(q_start, goal_flange_affine_right_, optimal_path_);
```

Thus, as long as the yale_gripper_frame is being published (e.g. by yale_gripper_xform.launch), forward and inverse kinematics can be computed with respect to the gripper frame. Specifically, the Cartesian-motion action server can be sent a goal gripper pose, along with an action code to select the server case:

```
case cwru_action::cwru_baxter_cart_moveGoal::RT_ARM_PLAN_PATH_CURRENT_TO_GOAL_POSE:

    rt_arm_plan_path_current_to_goal_pose();

    break;
```

This function will account for the gripper tool transform in its path planning.