Baxter RR Bridge

Contents

1 Reading This Document

This document is intended for a reader that has a medium understanding of using Robot Raconteur `http://robotraconteur.com`, ROS `http://ros.org`, and the Baxter robot. Documentation here is to help facilitate usage of this bridge between ROS and Robot Raconteur for communication / control over Baxter. This document assumes the robot has already been set up, and installed. It is up to date for RSDK 1.0.0 and Robot Raconteur version 0.4-testing. This has been tested using Ubuntu 12.04 and ROS Groovy.

2 Baxter

Baxter is a humanoid robot with two 7-degree-of-freedom arms and a full suite of sensors. Each joint has an encoder and a torque sensor, at the end of each arm is a gripper, camera, and range finder, there are 'Navigator' buttons on the body and arms, a ring of sonar sensors around the head, a panning head camera, and an LCD screen for its face. For full information, check out `http://www.rethinkrobotics.com/products/baxter/`

2.1 Startup

To start up the baxter communication, you need to run the following in a terminal:

```
$ cd ros_ws
$ ./baxter.sh
```

If run correctly, the terminal should now look something like:

```
[baxter - http://01xxxxP000x.local:11311] user@computername:~/ros_ws$
```

2.2 Baxter Bridge installation

In order to install the `baxter_RR_bridge` package, simply copy the `baxter_RR_bridge` folder into `ros_ws/src`. Then, after startup, run the following command while in the ros_ws directory: `$ catkin_make`

2.3 Available services

The Baxter Bridge services wrap Baxter's ROS control system with the Robot Raconteur interface, and allow for a simplified communication between client controllers and Baxter. The services are broken up as:

- **jointcontroller_host.py** - Joint and end effector information for both arms as well as joint control.

  `tcp://localhost:`*`[p]`*`/BaxterJointServer/Baxter`

- **camera_host.py** - Image stream from the hand and head cameras, as well as camera control.

  `tcp://localhost:`*`[p]`*`/BaxterCameraServer/left_hand_camera`
  `tcp://localhost:`*`[p]`*`/BaxterCameraServer/right_hand_camera`
  `tcp://localhost:`*`[p]`*`/BaxterCameraServer/head_camera`

- **peripherals_host.py** - Access to the grippers, head servos, and full suite of sensors.

  `tcp://localhost:`*`[p]`*`/BaxterPeripheralServer/BaxterPeripherals`

For each service, the *`[p]`* is the tcp port to access it. If this port is not provided at startup in the command line (using the `--port` flag), then it is automatically generated and displayed in the console.

3 jointcontroller_host.py

This service contains the main joint control over the robot. There is one main thread that handles the main RobotRaconteur interactions, and three backgrounds to constantly talk to Baxter to request data and transmit joint commands. To run this service:

```
$ rosrun baxter_RR_bridge jointcontroller_host.py [--port p]
```

3.1 Joint Measurements

The following properties provide information about the 14 joints on the Baxter robot. Data is sent in a stacked vector with indexes $1-7$ relating to the left arm and indexes $8-14$ relating to the right arm. Each arm's vector is always in the order:

$$q = \begin{bmatrix} s_0 & s_1 & e_0 & e_1 & w_0 & w_1 & w_2 \end{bmatrix}^T$$

translating to the two shoulder joints (s_0, s_1), two elbow joints (e_0, e_1) and the three wrist joints (w_0, w_1, w_2).

- `property double[] joint_positions`

  The joint angles as a 14×1 stacked vector $[q_L, q_R]^T$.

- `property double[] joint_velocities`

  The joint velocities as a 14×1 stacked vector $[\dot{q}_L, \dot{q}_R]^T$.

- `property double[] joint_torques`

  The joint torques as a 14×1 stacked vector $[\tau_L, \tau_R]^T$.

3.2 End Effector Measurements

The following properties provide information about the 2 end effectors at the end of each arm on the Baxter robot. Data is sent in a stacked vector with the first indexes relating to the left arm, and the last set of indexes relating to the right arm.

- `property double[] endeffector_positions`

  The end effector positions as a 6×1 vector of stacked 3D positions relative to the robot's base frame. $[p_L, p_R]^T$.

- `property double[] endeffector_orientations`

  The end effector orientations as an 8×1 vector of stacked quaternions that represent the end effector's coordinate frame in the robot base's coordinate frame $[\mathtt{q}_L, \mathtt{q}_R]^T$.

- `property double[] endeffector_twists`

  The end effector twists as a 12×1 stacked vector of twists in the robot base's coordinate system $[\omega_L, v_L, \omega_R, v_R]^T$.

- `property double[] endeffector_wrenches`

  The end effector wrenches as a 12×1 stacked vector of wrenches in the robot base's coordinate system $[\tau_L, f_L, \tau_R, f_R]^T$.

3.3 Joint Control

Functions to control the Baxter robot. A joint command must always be in the order:

$$\begin{bmatrix} s_0 & s_1 & e_0 & e_1 & w_0 & w_1 & w_2 \end{bmatrix}^T$$

- `function void setControlMode(uint8 mode)`

  Sets the control mode of the robot. 0 - Position, 1 - Velocity, 2 - Torque.

  Important Note: In torque control mode, the limbs may drift. This is a known issue within the RSDK. It is considered a negligible amount relative to any position / velocity control you may be implementing. A command of all zeros will likely still have motion, so using state feedback is necessary for full control.

- `function void setJointCommand(string limb, double[] command)`

  The 7×1 command to issue to one of the arms for whichever mode of operation the robot is currently in. Choose which arm with limb = {"left", "l"} or {"right", "r"}. The background thread will continuously issue the most recently received command at a rate of 100 Hz.

- `function void setPositionModeSpeed(double speed)`

  There is an internal trajectory generator in position mode, and this command sets the velocity at which it sends the arms to the set-point position. The input speed must be in $[0, 1]$. Default is 0.3.

4 camera_host.py

This is a generalized script to run any of the cameras on Baxter. It can only run a single camera at a time, so multiple cameras require multiple instances of this script to be run in separate terminals. **IMPORTANT: ONLY 2 CAMERA SERVICES CAN STREAM IMAGES AT A TIME**. Within this service, there is access to the latest image captured by the camera, as well as controls over the camera functionality.

```
$ rosrun baxter_RR_bridge camera_host.py camera_name [--mode {0,1,2,3,4,5}] [--half_res
{0,1}] [--port p]
```

Streaming at half-resolution ("half_res") will give the same field of view as its "parent resolution" but with 2x pixel binning (e.g. (320,200) at half resolution has the same field of view as (640, 400)), while "mode" crops the resolution for the camera stream based on the following table:

mode	resolution (width, height)	can display at half resolution
0	(1280, 960)	no
1	(960, 600)	no
2	(640, 400)	yes
3	(480, 300)	yes
4	(384, 240)	no
5	(320, 200)	yes

Note that the highest two resolutions cannot display at half resolution since there is no way to increase the field of view. This service will not engage the servos.

4.1 Open/Close

The service can be open without the data streaming. This allows for remote enabling / disabling of cameras, without shutting down the program. AT STARTUP THE CAMERA IS SET TO CLOSED.

- `property uint8 camera_open`

  A simple property that will show 1 if the camera stream is open, and 0 if the camera stream is closed. Will show 0 at startup.

- `function void openCamera()`

  A function which will open the camera and open a subscriber to its image topic.

- `function void closeCamera()`

  This function closes the camera and removes its subscription to the image topic.

4.2 Current Image

Information for the current image stream.

- `function BaxterImage getCurrentImage()`

  Returns the current image as a 'BaxterImage' struct. It consists of the fields

  - `field int32 width` - width of image
  - `field int32 height` - height of image
  - `field int32 step` - data step for single pixel in image
  - `field uint8[] data` - actual image data in BGRα raster order.

- `function ImageHeader getImageHeader()`

  Returns information on the size of the image. Useful for initialization before beginning the actual data stream.

  - `field int32 width` - width of image
  - `field int32 height` - height of image
  - `field int32 step` - data step for single pixel in image raster order.

- `function CameraIntrinsics getCameraIntrinsics()`

  Returns a struct containing the intrinsic parameters for the camera according to the pinhole model with distortion effects. The CameraIntrinsics struct contains the fields

  - `field double[] K` - vectorized form of the 3 x 3 intrinsic matrix K.
  - `field double[] D` - vector containing standard lens distortion coefficients.

- `pipe BaxterImage ImageStream`

  Provides an endpoint for streaming image data with an event thrown whenever new data is acquired. This is considered an advanced implementation and more information on pipes can be found at `http://robotraconteur.com`. The data returned is the same BaxterImage struct as in `getCurrentImage()`

4.3 Camera Control

Control functions for the various settings for the camera. Will reload the camera after completion, so it may drop some frames if sent during a data stream. It is recommended to use these settings before opening the camera.

- `function void setExposure(int16 exposure)`

  Sets the exposure for the camera. Must be in range [0, 100] or -1 for Auto Control (default).

- `function void setGain(int16 gain)`

  Sets the gain for the camera. Must be in range [0, 79] or -1 for Auto Control (default).

- `function void setWhiteBalance(int16 red, int16 green, int16 blue)`

  Sets the white balance for the camera. Must be in range [0, 4095] or -1 for Auto Control (default).

- `function void setFPS(double fps)`

  Sets the frames-per-second for the camera. Must be in range (0, 30]. On startup, it is set to 20.

5 peripherals_host.py

This service is intended to host all the various sensors, and give control over the various peripherals such as the grippers, head servos, and digital I/O. To start this script, run in the terminal:

```
$ rosrun baxter_RR_bridge periphals_host.py [--port p]
```

This service will not engage the servos. If you need to enable the servos, but aren't running the Baxter-JointServer, the best method is to run the script in the terminal:

```
$ rosrun baxter_tools enable_robot.py -e
```

5.1 Gripper Control

These are the functions which define the gripper operation. Each of the functions are parameterized with "string gripper", which can either be 'left', 'l', 'right', 'r', or any capitalized variation of those four. No action is performed if an unrecognized gripper is indexed. It is necessary to run `calibrateGripper(string gripper)` before future gripper usage is possible.

- `function void openGripper(string gripper)`

  Sends the gripper to its fully open pose (100).

- `function void closeGripper(string gripper)`

  Sends the gripper to its fully closed pose (0).

- `function void calibrateGripper(string gripper)`

  Runs gripper calibration routine. It is necessary to run this function before operation of the gripper is possible.

- `function void setGripperPosition(string gripper, double position)`

  Sends the gripper to the defined position. Position must be in $[0, 100]$.

- `function void setGripperVelocity(string gripper, double velocity)`

  Sets the velocity for the gripper to execute position commands. Velocity must be in $[0, 100]$. At startup, is set to 50.0.

- `function void setGripperHoldForce(string gripper, double force)`

  Sets the holding force of a successful gripper grasp. Force must be in $[0, 100]$. At startup, is set to 30.0.

- `function void setGripperMoveForce(string gripper, double force)`

  Sets the moving force threshold of the position move execution. Force must be in $[0, 100]$. At startup, is set to 40.0.

- `function void setGripperDeadband(string gripper, double deadband)`

  Sets the gripper deadband for successful position moves. Deadband must be in $[0, 100]$. At startup, is set to 5.0. **Note: If set to 0, will cause chattering.**

- `function single getGripperPosition(string gripper)`

  Returns the current gripper position in $[0, 100]$.

- `function single getGripperForce(string gripper)`

  Returns the current measured gripping force as a percentage of maximum force, $[0, 100]$.

5.2 Hand Rangers

The rangers at the end effector of each arm have a field of view of 5°, with a minimum range of 0.004 meters and a maximum range of 0.4 meters.

- `function single getRangerValue(string arm)`

  Returns the current range value in millimeters of the arm denoted by 'left', 'l', 'right' or 'r'. If there is nothing within the field of view, returns 65535.

5.3 Hand Accelerometers

There are accelerometers in the center of each wrist, with z pointing into the palm, x pointing towards the camera, and y pointing towards the cuff buttons according to standard right-hand-rule notation.

- `function single getAccelerometerValue(string arm)`

  Returns the current accelerometer vector for the arm denoted by 'left', 'l', 'right' or 'r'.

5.4 Head Control

The head has the ability to pan controllably, and also has a 'nodding' function. High speed servoing with this panning function is not recommended. **Note: These commands will not run unless the servos are enabled.**

- `function void panHead(double angle)`

  Sends the head to rotate to the commanded angle (in radians). Acceptable range is $\pm\pi/2$.

- `function single getHeadPanAngle()`

  Returns the current pan angle for the head in radians.

- `function void nodHead()`

  Runs the head nodding routine.

5.5 Head Sonar

The sonar ring at the top of Baxter consists of 12 sonar sensors, spaced equally in a full circle about its head. During sampling periods, the sensors may acquire multiple data points or zero data points, so the returned data is simply the sensors who received information during the last sampling period.

- `function void enableSonar()`

  Turns all of the sonar sensors on.

- `function void disableSonar()`

  Turns all of the sonar sensors off.

- `property SonarPointCloud sonar_pointcloud`

  Returns a SonarPointCloud struct with the fields

  - `field single[] sensors` - $N \times 1$ vector of sensor IDs which received data
  - `field single[] distances` - $N \times 1$ vector with the raw distances each sensor detected
  - `field single[] points` - $3N \times 1$ vector in XYZ order with the 3D points, in the robot base's coordinate system, that those distances translate to.

5.6 Background Service Suppression

There are a number of background services running on the Baxter robot, with access to suppress some of them available to the user. Each function is parameterized by limb = 'left', 'l', 'right', or 'r', and suppress = 0 or 1, although any nonzero value for suppress will be considered a 'true'. **IT IS IMPORTANT TO REMEMBER THAT SUPPRESSING ANY SAFETY SERVICE IS DANGEROUS TO THE USER AND ROBOT.**

- `function void suppressBodyAvoidance(string limb, uint8 suppress)`

  body_avoidance is a legacy service that will not change the actions of the robot. I have kept it here, but consider it deprecated.

- `function void suppressCollisionAvoidance(string limb, uint8 suppress)`

  collision_avoidance prevents the arm from colliding into other known bodies in its workspace (i.e. its own body or the other arm).

- `function void suppressContactSafety(string limb, uint8 suppress)`

  contact_safety stops motion for the arms during a significant opposing force (such as a collision).

- `function void suppressCuffInteraction(string limb, uint8 suppress)`

  cuff_interaction puts the arm in gravity compensation mode when the cuff touch button is engaged.

- `function void suppressGravityCompensation(string limb, uint8 suppress)`

  gravity_compensation applies torque to each servo to compensate for the gravitational load of the arm links.

5.7 Controller Information

There is some information about the controller that can be retrieved from the robot.

- `property double[] gravity_compensation_torques`

  Returns the gravity compensation torques for the current configuration as a 14×1 stacked vector with the first 7 indexes for the left arm, and remaining 7 indexes for the right.

5.8 Navigators

The 'Navigators' or ITBs are the digital I/O panels located on each arm and on either side of the robot's torso. They consist of 2 buttons, one with the rethink logo (called the "show" button), and the other with an arrow (called the "cancel" button). There is also a continuous scroll wheel, which also presses in to act as a 3rd button (called the "ok" button). For output, there is an LED corresponding to the inner circle of the Navigator, and another on the outer circle.

- `function NavigatorState getNavigatorState(string navigator)`

  This returns the current NavigatorState of the Navigator denoted by either 'left', 'right', 'torso_left', or 'torso_right'. NavigatorState is a struct containing the fields

  - `field uint8 ok_button` - 0 or 1 if pressed
  - `field uint8 cancel_button` - 0 or 1 if pressed
  - `field uint8 show_button` - 0 or 1 if pressed
  - `field uint8 scroll_wheel` - can be in range 0 - 255
  - `field uint8 inner_led` - 0 or 1 if on
  - `field uint8 outer_led` - 0 or 1 if on

- `function void setNavigatorLEDs(string navigator, uint8 inner_led, uint8 outer_led)`

  Sets the state of the Navigator LEDs, denoted by the string 'left', 'right', 'torso_left', or 'torso_right'. Send either a 0 for off or a 1 for on in the parameters inner_led, outer_led.