Project 2

Bookmark this page

Project 2 due Nov 29, 2020 18:30 EST *Completed*

ACADEMIC HONESTY

As usual, the standard honor code and academic honesty policy applies. We will be using automated **plagiarism detection** software to ensure that only original work is given credit. Submissions isomorphic to (1) those that exist anywhere online, (2) those submitted by your classmates, or (3) those submitted by students in prior semesters, will be detected and considered plagiarism.

Project 2

Description

This project will introduce you to '`tf`', the ROS framework for handling transforms. Please make sure you have read the entry on this package on the ROS wiki. In this project we consider a ROS ecosystem, which consists of a robot with a camera mounted on it as well as an object. To describe the poses of all these items, we define the following coordinate frames:

- A base coordinate frame called '`base`'
- A robot coordinate frame called '`robot`'
- A camera coordinate frame called '`camera`'
- An object coordinate frame '`object`'

The following relationships are true:

1. The transform from the '`base`' coordinate frame to the '`object`' coordinate frame consists of a rotation expressed as (roll, pitch, yaw) of (0.79, 0.0, 0.79) followed by a translation of 1.0m along the resulting y-axis and 1.0m along the resulting z-axis.
2. The transform from the '`base`' coordinate frame to the '`robot`' coordinate frame consists of a rotation around the z-axis by 1.5 radians followed by a translation along the resulting y-axis of -1.0m.
3. The transform from the '`robot`' coordinate frame to the '`camera`' coordinate frame must be defined as follows:
   - The translation component of this transform is (0.0, 0.1, 0.1)
   - The rotation component of this transform must be set such that the camera is pointing directly at the object. In other words, the x-axis of the '`camera`' coordinate frame must be pointing directly at the origin of the '`object`' coordinate frame.

In the provided `solution.py` write a ROS node that publishes the following transforms to TF:

- The transform from the '`base`' coordinate frame to the '`object`' coordinate frame
- The transform from the '`base`' coordinate frame to the '`robot`' coordinate frame
- The transform from the '`robot`' coordinate frame to the '`camera`' coordinate frame

Additional Information

You will probably want to make use of the `transformations.py` library. The documentation for using that is in the library itself; you can reference the version used with ROS online on Github (be careful - other versions of this file exist on the Internet, so if you just Google for it you might get the wrong one).

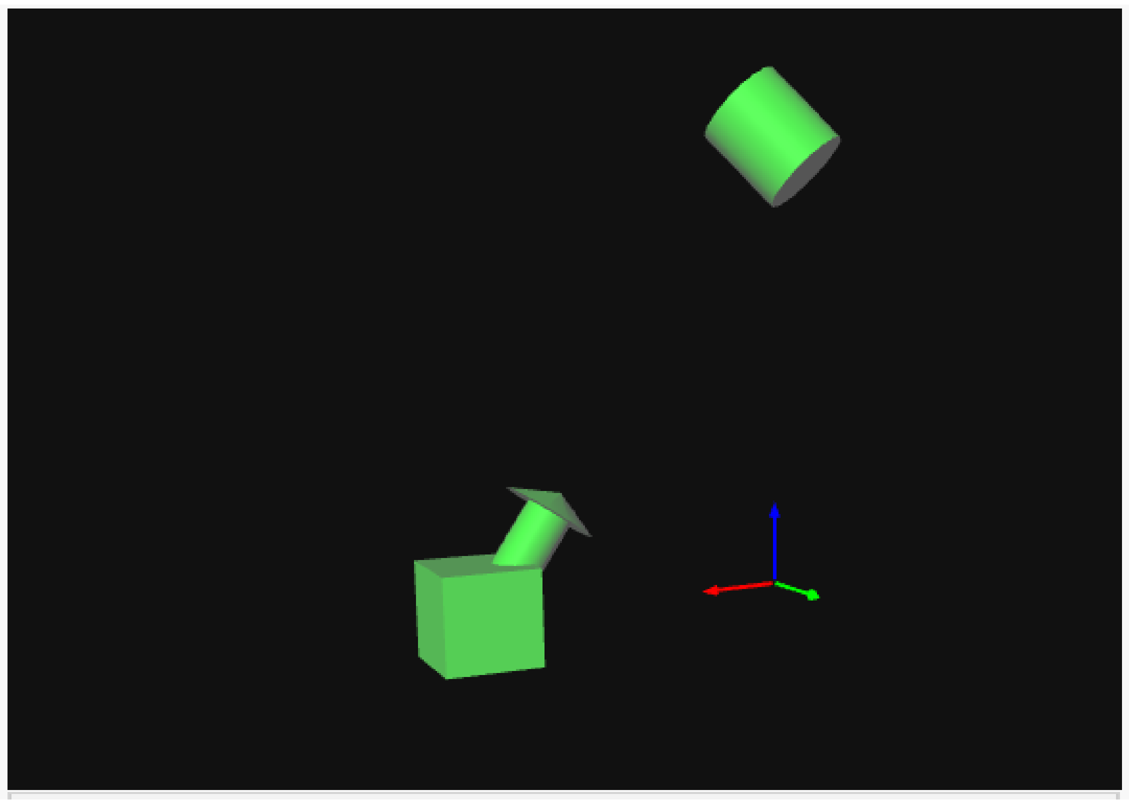
For a rotation expressed as roll-pitch-yaw, you can use the `quaternion_from_euler()` or `euler_matrix()` functions with the default axes convention - i.e. `quaternion_from_euler(roll_value, pitch_value, yaw_value)`. You can also use the code in `tf_examples.py` for guidance.

Be careful about the order of operations. If a transform specifies that the rotation must happen first, followed by the translation (e.g. at points 1. and 2. above), make sure to follow that.

The transforms must be published in a continuous loop at a rate of 10Hz or more. The skeleton code you are provided already does that, so all you need to do is edit the `publish_transforms()` function to fill in the transforms with the appropriate values.

This assignment also includes some visual feedback. Once you have sourced `setup_project2.sh` you can click the 'Connect' button. You will see an interactive visualization containing a cube, a cylinder and an arrow. Initially they are all placed at the origin (and the cube will occlude the cylinder).

Once you run your code, these bodies will position themselves in space according to the transforms your code is publishing. The cylinder denotes the object, the cube and arrow the robot and camera respectively. If your code works correctly, you should see the arrow point out of the cube directly at the cylinder. Here is an example of the correct output (note that the colored axes show you the location of the base coordinate frame with the usual convention: x-red, y-green, z-blue):



Setup

As always, make sure to `source setup_project2.sh` before trying to invoke any ROS commands (`catkin_make`, `roscd`, etc.). This will also start a `roscore` for your session. **Please do not start your own `roscore`.**

As mentioned above, after you have sourced `setup_project2.sh` simply run your node `rosrun project2_solution solution.py`. After that, you can click the 'Canvas' button on the right corner and then click the 'Connect' button. You will see an interactive visualization of the transforms in the assignment (if you're curious, this was created using ROS Markers).

Grading

Similarly to Project 1, your code will automatically be graded when you submit. After a short wait, there will be a 'Submission Report' available for you under the 'Details' tab. **If you submit your project before October 11th 2020, 23:30 UTC, you will earn an early submission bonus.** The maximum number of points you can earn in the class is still 100. **The project's final due date is November 29th 2020, 23:30 UTC**.

Each of the three transforms that you need to publish is worth 5 points. For each transform, you will get the points only if the transform you publish is correct in its entirety (within numerical precision) - no partial credit if only the rotation part is correct, or only the translation, etc.

Launch Project 2 (External resource) (10.999999999999948 / 15.0 points)

< Previous   Next >