Project 3

Bookmark this page

Project 3 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.

Description

In this project you will implement the forward kinematics for a robot arm defined in a URDF file and running in a ROS environment.

The setup contains a "simulated" robot that continuously publishes its own joint values. After you have run through the Setup instructions (see below), you can check that the robot is indeed publishing its joint values by using the '`rostopic echo /joint_states`' command. However, that is not enough for the robot to be correctly displayed: a forward kinematics module must use the joint values to compute the transforms from the world coordinate frame to each link of the robot. This is the code you must fill in.

Your job will be to complete the code '`solution.py`' in the '`forward_kinematics'` package provided to you. When you familiarize yourself with the starter code you will see that the '`ForwardsKinematics'` class subscribes to the topic '`joint_states`' and publishes transforms to '`tf`'. It also loads a URDF description of the robot from the ROS parameter server. You will only have to edit '`solution.py`' and fill in the `compute_transforms` function. If you want, you can also peruse the rest of the skeleton we provide to get an even better understanding of what is going on behind the scenes.

Every time the subscribed receives new joint values, we do some prep work for you. We unpack from the URDF all the data you will need, including the structure of the robot arm as lists of joint objects and link names. Then, we pass this data, along with the joint values, to the `compute_transforms` function which you must fill in.

The '`compute_transforms`' function

This is the function that performs the main forward kinematics computation. It accepts as parameters all the information needed about the joints and links of the robot, as well as the current values of all the joints, and must compute and return the transforms from the world frame to all the links, ready to be published through tf.

Parameters are as follows:

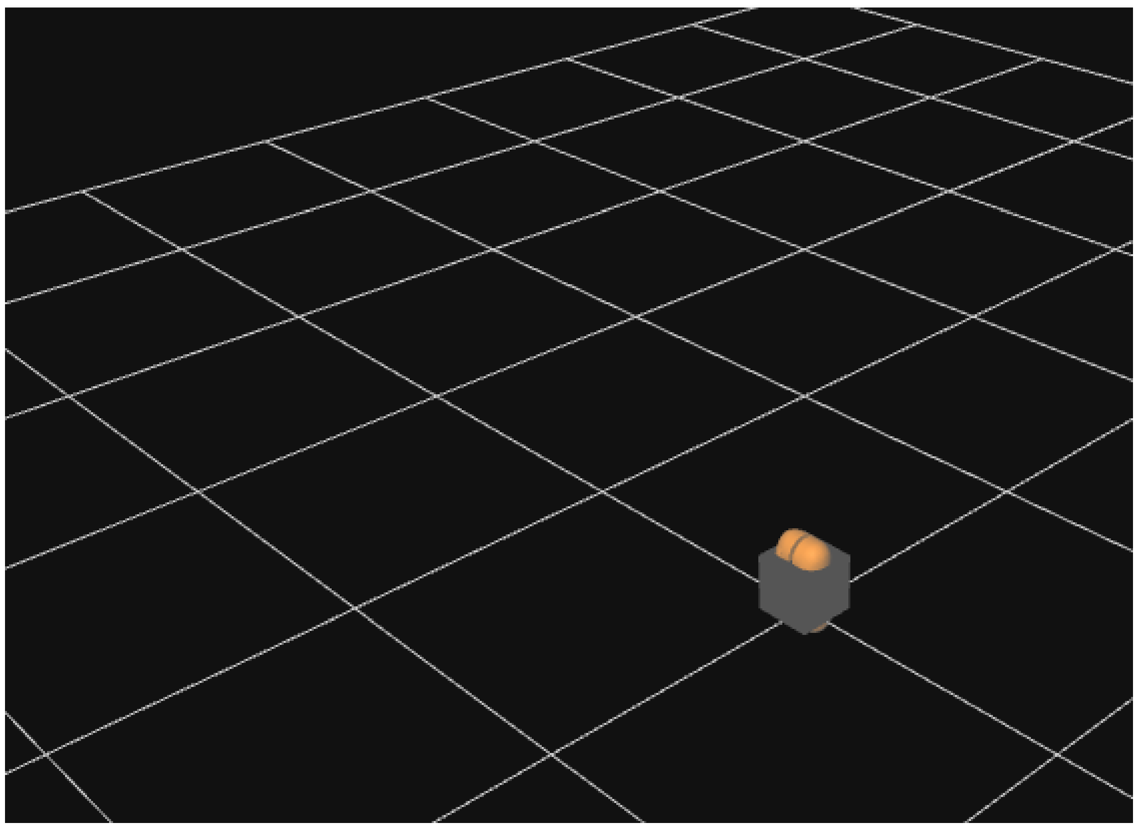
- `link_names`: a list with all the names of the robot's links, ordered from proximal to distal. These are also the names of the link's respective coordinate frame. In other words, the transform from the world to link i should be published with `world_link` as the parent frame and `link_names[i]` as the child frame.
- `joints`: a list of all the joints of the robot, in the same order as the links listed above. Each entry in this list is an object which contains the following fields:
  - `joint.origin.xyz`: the translation from the frame of the previous joint to this one
  - `joint.origin.rpy`: the rotation from the frame of the previous joint to this one, in ROLL-PITCH-YAW XYZ convention
  - `joint.type`: either '`fixed`' or '`revolute`'. A fixed joint does not move; it is meant to contain a static transform.
  - `joint.name`: the name of the current joint in the robot description
  - `joint.axis`: (only if type is '`revolute`') the axis of rotation of the joint
- `joint_values` contains information about the current joint values in the robot. It contains information about **all** the joints, and **the ordering can vary**, so we must find the relevant value for a particular joint you are considering. We can use the following fields:

- `joint_values.name`: a list of the names of all the joints in the robot;
- `joint_values.position`: a list of the current values of all the joints in the robot, in the same order as the names in the list above. To find the value of the joint we care about, we must find its name in the `name` list, then take the value found at the same index in the `position` list.

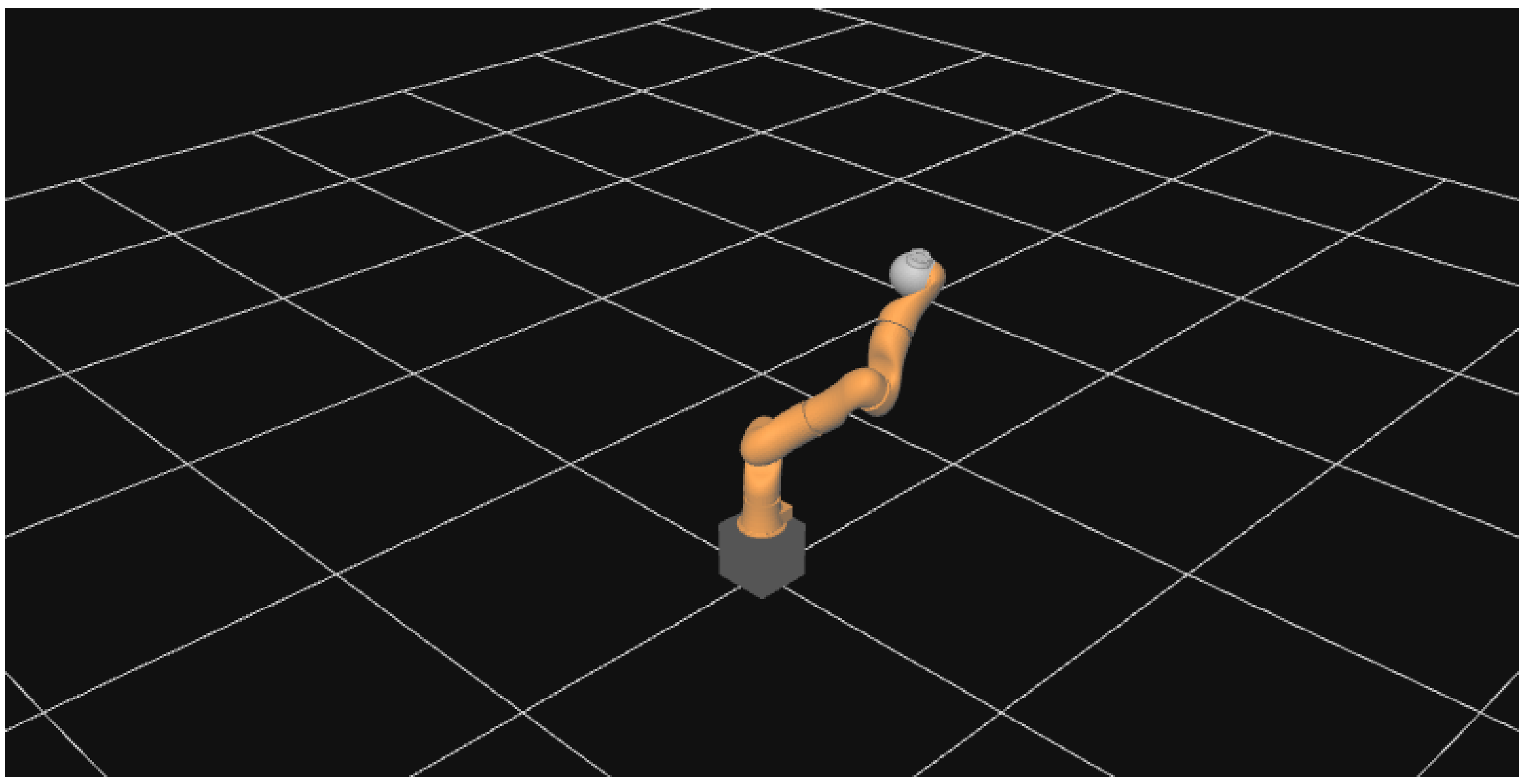
The function must return one tf message. The `transforms` field of this message must list **all** the transforms from the world coordinate frame to the links of the robot. In other words, when you are done, all_transforms.transforms must contain a list in which you must place all the transforms from the `world_link` coordinate frame to each of the coordinate frames listed in `link_names`. You can use the `convert_to_message` function (defined above) for a convenient way to create a tf message from a transformation matrix.

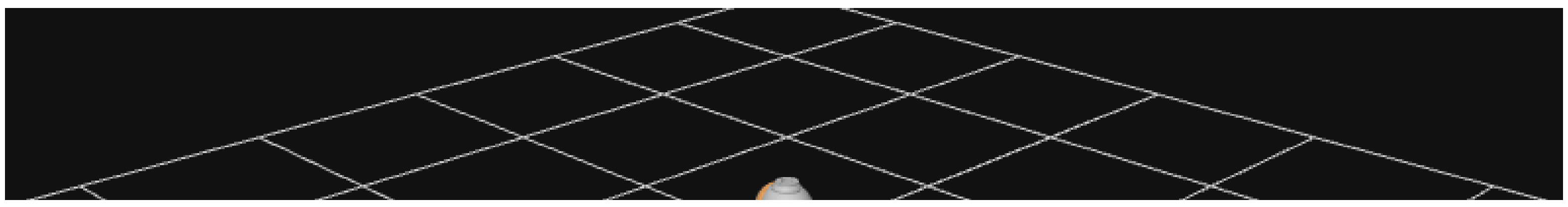
Setup

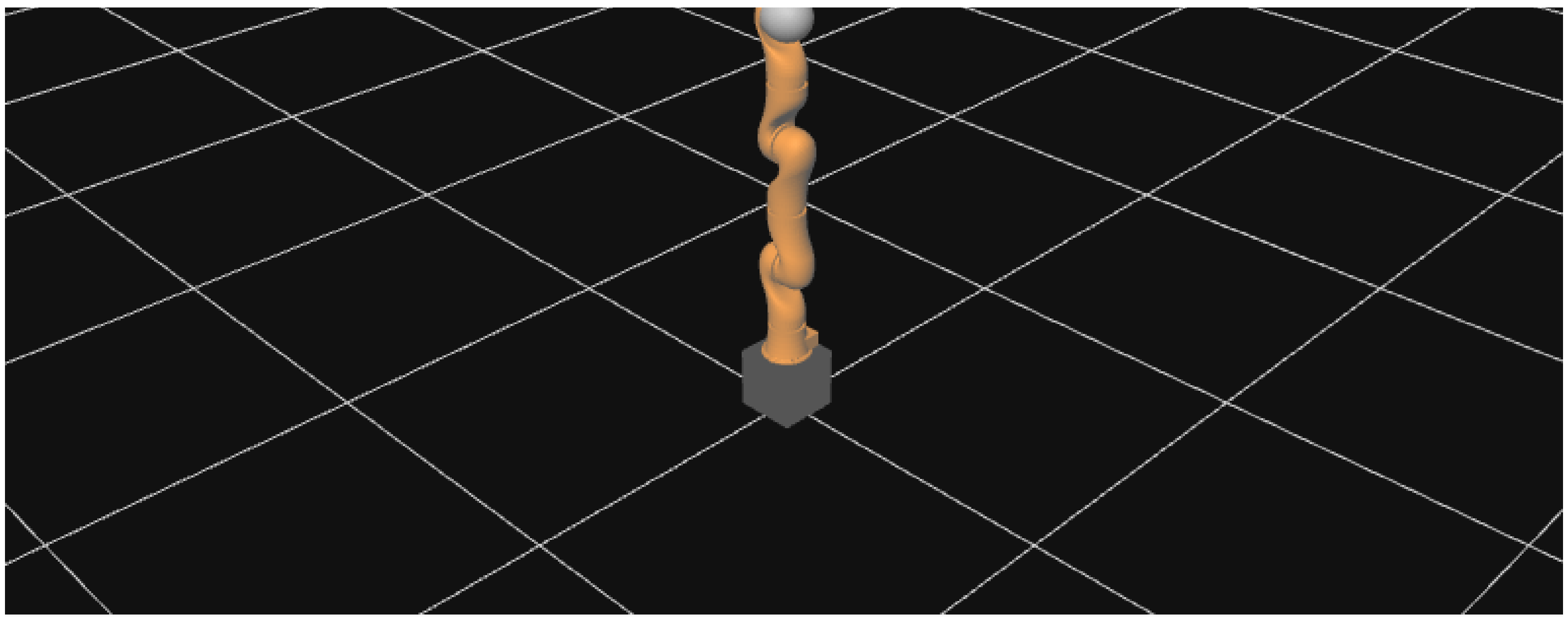
Similarly to the first two projects, please make sure you source the '`setup_`project3`.sh`' before you attempt to run your code. This starts a roscore and loads the robots URDF into the ROS parameter server. After you have done that, you can press the 'Connect' button and you should see the robot arm with all its links placed at the origin. This is because no transform tree is being published and ROS does not know where to place the links.



The setup script will also start a nodes that you can find in the '`robot_mover`' and '`robot_sim`' package. These node publish joint values on the '`joint_states`' topic, which your forward kinematics code subscribes to. All that is left for you to do is to run your completed code. If you have done everything correctly, you should see the robot arm move back and forth in a physically correct fashion.







When you run solution.py, you will get a Warning along the lines of "`Unknown tag: comScalar element defined multiple times...`". You can safely ignore this.

If you get a notification that the websocket connection has closed that means that the connection between ROS and the Canvas has broken down. You will have to reload the page and source the setup script again before ROS can use the Canvas again.

Resources and Hints

It will help to get familiar with the URDF documentation. In particular, the documentation for the URDF Joint element will be very helpful in understanding the nature of the `joint` object that is being passed to the `compute_transforms` function, and what you must do with the data in each `joint` object.

Remember that you must compute (and publish) the transform **from the world coordinate frame** (called `world_link`) to each link of the robot. However, the URDF tells you the transform **from one link to the next** one in the chain (through the joint between them). Thus, one way to complete the assignment is in iterative fashion: assuming you have compute the transform from the `world_link` coordinate frame to `link i`, you just need to update that with the transform from `link i` to `link i+1` and you now have the transform from the `world_link` frame to `link i+1`.

Grading

As always, your code will be graded as soon as you submit - you can submit as many times as you like, the final submit is the one that counts. You will receive 5 points each for correctly publishing the transforms of three different links along the kinematic chain for a total of 15 points. **If you submit your project before October 18th 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**.

Good luck!

Launch Project 3 (External resource) (15.0 / 15.0 points)

< Previous   Next >

About
Affiliates
edX for Business
Open edX
Careers
News

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap

Connect

Blog
Contact Us
Help Center
Media Kit
Donate

GET IT ON Google play
Download on the App Store