Example Tf Listener

Wyatt Newman
March, 2015

An earlier document, “Survival Guide for Coordinate Transforms,” introduced coordinate transforms, with an emphasis on localization. This document describes the use of ROS's “transform listener” for obtaining coordinate transformations.

One representation of a robot's pose (and twist) in space is based on odometry—typically by keeping track of the cumulative influence of incremental wheel rotations. Publications of “odom” are useful for steering feedback control, since these updates are fast and smooth—albeit subject to drift.

Another representation of a robot's pose is estimation of coordinates relative to a pre-defined map. (See the class document “Mapping in ROS” for details on one way to create a ROS-compatible map). A map has a reference frame (the “map” frame), defining the map's origin and x,y,z axes. The robot's pose map be specified in the map frame. In fact, desired poses should be expressed in the map frame, since this is the frame in which planning is performed, so as to avoid obstacles in the map.

With reference to a (2-D) pre-defined map, LIDAR data can be interpreted in terms of hypothetical poses of the robot in the map. The hypothetical pose that best explains the LIDAR data in terms of boundaries in the map is accepted as the current pose of the robot in the map frame. With more opportunity to acquire data, to observe revealing features in the environment, and to consider a larger number of pose hypotheses, the accepted robot pose becomes increasingly credible. One algorithm that performs this estimation is Adaptive Monte Carlo Localization (AMCL; see: http://wiki.ros.org/amcl). These estimates are subject to relatively slow and potentially jerky updates. A good use of map-frame pose estimates is to identify the “odom” frame relative to the “map” frame, as discussed in the “Survival” transform guide. With the transform from map origin to odom origin (and rotation), subgoals can be transformed from a plan in map coordinates into subgoals in odom coordinates. Provided the subgoals are transformed only as needed (using the most current estimate of map-to-odom transform), this approach can achieve both smooth feedback control and low drift.

Using a Transform Listener: Performing transforms from one frame to another is enabled by the “tf” package in ROS (see: http://wiki.ros.org/tf/Tutorials). Use of the tf listener is illustrated with example code in the package “example_tf_listener” of our class repository. The key line of this node is:

```
tfListener_->lookupTransform("map", "base_link", ros::Time(0), baseLink_wrt_map_);
```

The object tfListener_ is a (pointer to a) transform listener. A transform listener subscribes to the topic “tf”, and it constantly attempts to assemble the most current chain of transforms possible from all published parent/child spatial relationships. The transform listener, once it has been instantiated and given a brief time to start collecting published transform information, offers a variety of useful methods. In the above case, the method “lookupTransform” finds a transformation between the named frames (map and base_link). Equivalently, this transform tells us where is the robot in the map (both position and orientation).

The “lookupTransform” method fills in baseLink_wrt_map_, which is an object of type tf::StampedTransform. A stamped transform contains both an origin (a 3-D vector) and an orientation

(a 3x3 matrix). The example code prints compnents of this transform, using the accessor functions getOrigin() and getRotation(). These functions return objects defined in the “tf” package. They can be converted into alternative representations, e.g. geometry_msgs Pose objects.

The constructor of the example code (a class, DemoTfListener) includes a check for the existence of a logical chain from “odom” to “map.” For the constructor to pass this test, there must by some localization algorithm (e.g. AMCL) publishing estimated transforms from map frame to odom frame. This would also require: a robot publishing “odom” coordinates, a reference map, and LIDAR data being published (with a known transform from the LIDAR sensor frame to the robot's base frame).

Transforming frames using tf: The transform listener can be used to compute a transform object. It can also be used to convert coordinates from an initial frame to some desired frame. Two ways to perform such transforms are illustrated in the class code “example_des_state_generator.cpp.” The method “map_to_odom_pose()” converts a (stamped) pose expressed in map coordinates into a (stamped) pose in odom coordinates (as desired for steering feedback in odom coordinates).

One of the illustrated methods is:

```
tf_odom_goal = mapToOdom_*tf_map_goal;
```

where tf_odom_goal and tf_map_goal are both of type tf::Point. Although mapToOdom_ is an object, the operator “*” is defined (for suitable operand types).

A more convenient means of transformation is to use the tf method “transformPose()”. This method takes arguments of type geometry_msgs::PoseStamped. It operates on the first pose argument (in an arbitrary reference frame) and fills in the second argument (with respect to a named coordinate frame).

The line of code:

```
tfListener_->transformPose("odom", map_pose, odom_pose);
```

converts “map_pose”, which is expressed in “map” coordinates, into “odom_pose”, which is expressed in “odom” coordinates. In this case, only one frame—the target frame—was explicitly declared. This is possible because a PoseStamped carries its own label defining its reference frame.

For illustration, the print statement:

```
std::cout<<odom_pose.header.frame_id<<std::endl;
```

examines the frame_id of odom_pose and prints it to the screen.

The frame id of the original pose was established by the example_path_sender node, with the line:

```
vertex.header.frame_id = "map"; // specify this, so tf will know how to transform it
```

With this information in the PoseStamped header, the pose values are more meaningful, since we know from which frame they are measured.

Running simulated cwruBot with subgoals in map space: A relatively sophisticated example can be run, combining examples to date. Follow the steps below:

- **start up cwruBot**

1. start roscore
2. rosparam set use_sim_time true
3. rosrun gazebo_ros gazebo

4. roslaunch cwru_urdf cwruBot.launch
5. insert a Gazebo model, e.g. the starting pen
6. rosrun map_server map_server startingPMap.yaml (from the cwru_urdf directory: load a map of the starting pen).

Note that the above steps would differ for a physical robot. Gazebo would not be necessary, the environment would be real (no model insertion necessary for Starting Pen), and the map that is loaded should be relevant for the robot's actual environment.

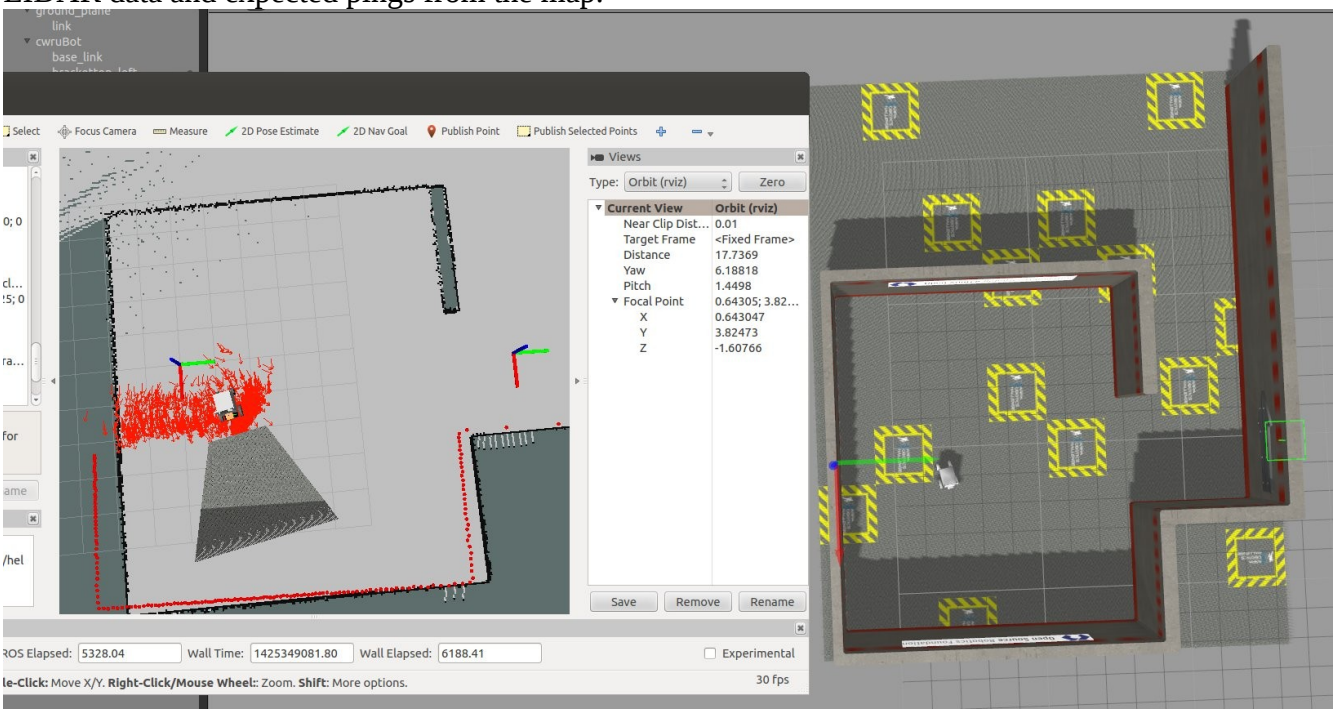
- **Start up navigation**

1. rosrun amcl amcl scan:=/laser/scan (start up AMCL localization node, based on LIDAR)
2. rosrun example_steering_algorithm example_steering_algorithm (this should be a version that performs real, tuned feedback)
3. rosrun example_des_state_generator des_state_generator (publishes streams of desired states, in odom coordinates; tuning parameters will vary between cwruBot and Jinx).
4. rosrun example_des_state_generator example_path_sender_starting_pen (this node publishes hard-coded subgoal poses in “map” coordinates, relevant to the Starting Pen map; use an alternative for Jinx's actual environment map)

- **Visualize the results**

1. rosrun rviz rviz
   include topics to display axes for map and odom frames, a “map” topic, and a PoseArray topic (to display AMCL pose hypotheses).
2. The “2D Pose Estimate” tool in Rviz may be used to help give AMCL a reasonable initial estimate of cwruBot's pose (or Jinx's pose)

The screenshot below shows a view of both Gazebo and Rviz while cwruBot is attempting to localize and navigate to subgoals. The misalignment between the LIDAR data and the map walls reveals the error in the transform between map and odom frames. The large cloud of red arrows shows that AMCL is considering a large variance of hypothetical poses, as motivated by the poor agreement between the LIDAR data and expected pings from the map.



As cwruBot moves, more data is gathered, and AMCL improves its estimate of the robot's pose in the map. As shown from the figure below, the LIDAR data aligns better with expectation (the map boundaries), and the cloud of pose hypotheses has shrunk to a relatively small region, centered under (and obscured by) cwruBot.