electric	fuerte	groovy	hydro	indigo	jade	kinetic	lunar	Documentation Status

***geometry (/geometry?distro=lunar)**: angles (/angles?distro=lunar) | eigen_conversions (/eigen_conversions?distro=lunar) | kdl_conversions (/kdl_conversions?distro=lunar) | tf | tf_conversions (/tf_conversions?distro=lunar)*

Package Links

- **Code API (http://docs.ros.org/lunar/api/tf/html)**
- **Msg/Srv API (http://docs.ros.org/lunar/api/tf/html/index-msg.html)**
- Tutorials (/tf/Tutorials)
- Troubleshooting (/tf/Troubleshooting)
- FAQ (http://answers.ros.org/questions/scope:all/sort:activity-desc/tags:tf/page:1/)
- Changelog (http://docs.ros.org/lunar/changelogs/tf/changelog.html)
- Change List (/geometry/ChangeList)
- Reviews (/tf/Reviews)

Dependencies (13)
Used by (77)
Jenkins jobs (13)

Package Summary

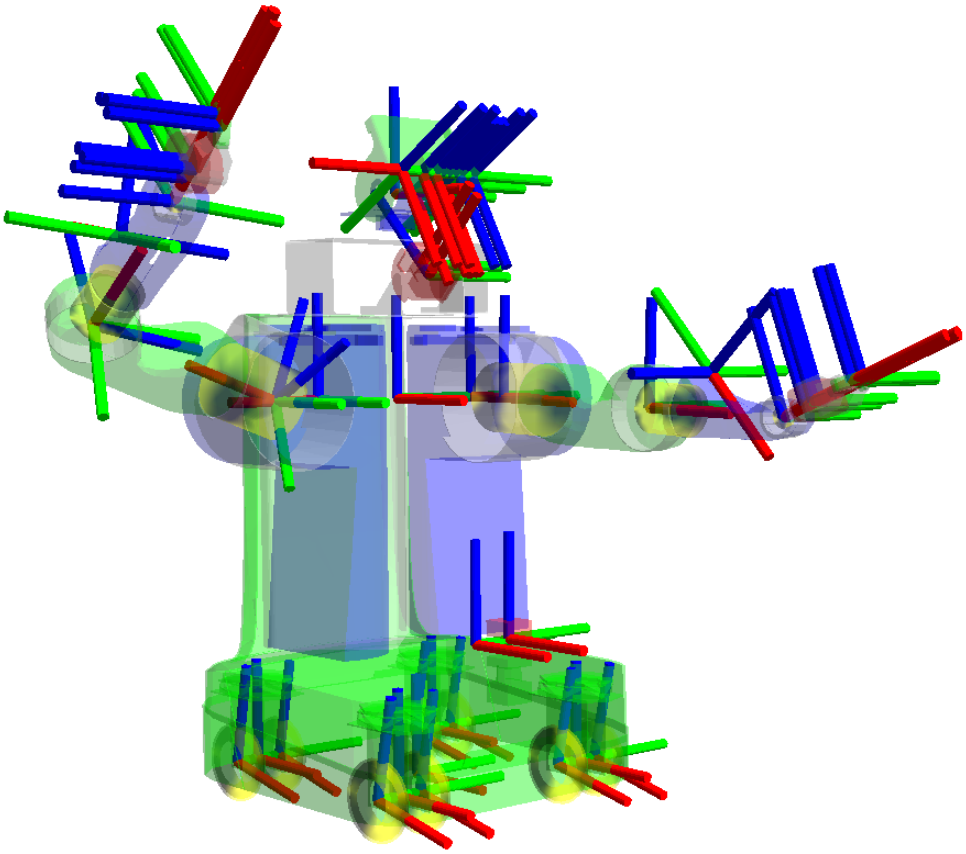
✔ **Released** ✔ **Continuous integration** ✔ **Documented**

tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

Migration: Since ROS Hydro, tf has been "deprecated" in favor of tf2 (http://wiki.ros.org/tf2). tf2 is an iteration on tf providing generally the same feature set more efficiently. As well as adding a few new features.
As tf2 is a major change the tf API has been maintained in its current form. Since tf2 has a superset of the tf features with a subset of the dependencies the tf implementation has been removed and replaced with calls to tf2 under the hood. This will mean that all users will be compatible with tf2. It is recommended for new work to use tf2 directly as it has a cleaner interface. However tf will continue to be supported for through at least J Turtle.

- Maintainer status: maintained
- Maintainer: Tully Foote <tfoote AT osrfoundation DOT org>
- Author: Tully Foote, Eitan Marder-Eppstein, Wim Meeussen
- License: BSD
- Source: git https://github.com/ros/geometry.git (https://github.com/ros/geometry) (branch: indigo-devel)

Contents



1. What does tf do? Why should I use tf?

You want to **see** what tf can do instead of just reading about it? Check out the tf introduction demo (/tf/Tutorials/Introduction%20to%20tf).

A robotic system typically has many 3D coordinate frames (/geometry/CoordinateFrameConventions) that change over **time**, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time, and allows you to ask questions like:

- Where was the head frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in my gripper relative to my base?

- What is the current pose of the base frame in the map frame?

tf can operate in a **distributed system**. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. There is **no central server** of transform information.

For more information on the design see /Design (/tf/Design)

2. Paper

There is a paper on tf presented at TePRA 2013 Papers/TePRA2013_Foote (/Papers/TePRA2013_Foote)

3. Tutorials

We created a set of tutorials (/tf/Tutorials) that walk you through using tf, step by step. You can get started on the introduction to tf (/tf/Tutorials/Introduction%20to%20tf) tutorial. For a complete list of all tf and tf-related tutorials check out the tutorials (/tf/Tutorials) page.

There are essentially two tasks that any user would use tf for, listening for transforms and broadcasting transforms.

Anyone using tf will need to listen for transforms:

- **Listening for transforms** - Receive and buffer all coordinate frames that are broadcasted in the system, and query for specific transforms between frames. Check out the writing a tf listener tutorial (Python) (/tf/Tutorials/Writing%20a%20tf%20listener%20%28Python%29) (C++) (/tf/Tutorials/Writing%20a%20tf%20listener%20%28C%2B%2B%29).

To extend the capabilities of a robot you will need to start broadcasting transforms.

- **Broadcasting transforms** - Send out the relative pose of coordinate frames to the rest of the system. A system can have many broadcasters that each provide information about a different part of the robot. Check out the writing a tf broadcaster tutorial (Python) (/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28Python%29) (C++) (/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28C%2B%2B%29).

Once you are finished with the basic tutorials, you can move on to learn about tf and time. The tf and time tutorial (Python) (/tf/Tutorials/tf%20and%20Time%20%28Python%29) (C++) (/tf/Tutorials/tf%20and%20Time%20%28C%2B%2B%29) teaches the basic principles of tf and time. The advanced tutorial about tf and time (Python) (/tf/Tutorials/Time%20travel%20with%20tf%20%28Python%29) (C++) (/tf/Tutorials/Time%20travel%20with%20tf%20%28C%2B%2B%29) teaches the principles of time traveling with tf.

4. Code API Overview

tf API Overview (/tf/Overview):

- Data Types (/tf/Overview/Data%20Types)

- Transformations and Frames (/tf/Overview/Transformations)
- Broadcasting Transforms (/tf/Overview/Broadcasting%20Transforms)
- Using Published Transforms (/tf/Overview/Using%20Published%20Transforms)
- Exceptions (/tf/Overview/Exceptions)

5. Frequently asked questions

- Frequently Asked Questions (/tf/FAQ)
- Rotation methods (/geometry/RotationMethods)
- Coordinate Frame Conventions (/geometry/CoordinateFrameConventions)

6. Command-line Tools

Although tf is mainly a code library meant to be used within ROS nodes (/Nodes), it comes with a large set of command-line tools that assist in the debugging and creation of tf coordinate frames. These tools include:

- view_frames (/tf#view_frames): visualizes the full tree of coordinate transforms.
- tf_monitor (/tf#tf_monitor): monitors transforms between frames.
- tf_echo (/tf#tf_echo): prints specified transform to screen
- roswtf (/roswtf): with the `tfwtf` plugin, helps you track down problems with tf.
- static_transform_publisher (/tf#static_transform_publisher) is a command line tool for sending static transforms.

You may also wish to use the tf_remap (/tf#tf_remap) node, which is a utility node for remapping coordinate transforms.

6.1 tf_monitor

`tf_monitor`

Print information about the current coordinate transform tree to console. For example:

```
$ rosrun tf tf_monitor
RESULTS: for all Frames

Frames:
Frame: /base_footprint published by /robot_pose_ekf Average Delay: 0.0469324 Max
 Delay: 0.0501503
Frame: /base_laser_link published by /robot_state_publisher Average Delay: 0.0089
1066 Max Delay: 0.009591
Frame: /base_link published by /robot_state_publisher Average Delay: 0.00891147 M
ax Delay: 0.009592
0.00891431 Max Delay: 0.009595

... editing for the sake of brevity ...

Broadcasters:
Node: /realtime_loop 94.7371 Hz, Average Delay: 0.000599916 Max Delay: 0.001337
Node: /robot_pose_ekf 30.8259 Hz, Average Delay: 0.0469324 Max Delay: 0.0501503
Node: /robot_state_publisher 25.8099 Hz, Average Delay: 0.0089224 Max Delay: 0.00
960276
```

`tf_monitor <source_frame> <target_target>`

Monitor a specific transform. For example, to monitor the transform from `/base_footprint` to `/odom`:

```
$ rosrun tf tf_monitor /base_footprint /odom
RESULTS: for /base_footprint to /odom
Chain currently is: /base_footprint -> /odom
Net delay     avg = 0.00371811: max = 0.012472

Frames:
Frame: /base_footprint published by /robot_pose_ekf Average Delay: 0.0465218 Max
 Delay: 0.051754
Frame: /odom published by /realtime_loop Average Delay: 0.00062444 Max Delay: 0.0
01553

Broadcasters:
Node: /realtime_loop 95.3222 Hz, Average Delay: 0.00062444 Max Delay: 0.001553
Node: /robot_pose_ekf 30.9654 Hz, Average Delay: 0.0465218 Max Delay: 0.051754
Node: /robot_state_publisher 25.9839 Hz, Average Delay: 0.00903061 Max Delay: 0.0
0939562
```

6.2 tf_echo

`tf_echo <source_frame> <target_frame>`

Print information about a particular transformation between a `source_frame` and a `target_frame`. For example, to echo the transform between `/map` and `/odom`:

```
$ rosrun tf tf_echo /map /odom
At time 1263248513.809
- Translation: [2.398, 6.783, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.707, 0.707]
in RPY [0.000, -0.000, -1.570]
```

6.3 static_transform_publisher

`static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms`

Publish a static coordinate transform to tf using an x/y/z offset in meters and yaw/pitch/roll in radians. (yaw is rotation about Z, pitch is rotation about Y, and roll is rotation about X). The period, in milliseconds, specifies how often to send a transform. 100ms (10hz) is a good value.

`static_transform_publisher x y z qx qy qz qw frame_id child_frame_id  period_in_ms`

Publish a static coordinate transform to tf using an x/y/z offset in meters and quaternion. The period, in milliseconds, specifies how often to send a transform. 100ms (10hz) is a good value.

`static_transform_publisher` is designed both as a command-line tool for manual use, as well as for use within roslaunch (/roslaunch) files for setting static transforms. For example:

Toggle line numbers

```
   1 <launch>
   2 <node pkg="tf" type="static_transform_publisher" name="link1_broadcaster"
 args="1 0 0 0 0 0 1 link1_parent link1 100" />
   3 </launch>
```

6.4 view_frames

`view_frames` is a graphical debugging tool that creates a PDF graph of your current transform tree.

```
$ rosrun tf view_frames
```

You probably want to view the graph when you are done, so a typical usage on Ubuntu systems is:

```
$ rosrun tf view_frames
$ evince frames.pdf
```

Therefore an helpful shortcut to add in your `.bashrc` is:

```
alias tf='cd /var/tmp && rosrun tf view_frames && evince frames.pdf &'
```

NOTE: See also rqt_tf_tree (/rqt_tf_tree) that allows dynamic introspection of the frames.

6.5 roswtf plugin

`roswtf` tf comes with a plugin for roswtf (/roswtf) that automatically runs whenever you run `roswtf`. This plugin will analyze your current tf configuration and attempt to find common problems. To run, just invoke `roswtf` normally:

```
$ roswtf
```

7. Nodes

7.1 tf_remap

tf_remap listens to the `/tf_old` topic and republishes transforms on the `/tf` topic. This is mainly used with out-of-date bag files (/Bags) that need their coordinate frame IDs updated. The typical operation for this node is to play the bag file with `/tf:=/tf_old`. The `tf_remap` node is run with a `~mappings` parameter that describes the mapping of frame IDs from old to new.

7.1.1 Subscribed Topics

`/tf_old` (tf/tfMessage (http://docs.ros.org/api/tf/html/msg/tfMessage.html))
: Old transform tree. This is usually published by remapping playback of a bag file (/Bags). You may need to set use_sim_time true for the bag file transforms to be accepted.

7.1.2 Published Topics

`/tf` (tf/tfMessage (http://docs.ros.org/api/tf/html/msg/tfMessage.html))
: Current transform tree. This is the normal `/tf` topic.

7.1.3 Parameters

`~mappings` (`[ {str: str} ]`, default: [])
: List of dictionaries that map old tf frame IDs to new frame IDs. Each dictionary in the list must have an "old" and "new" key.

7.2 change_notifier

`change_notifier` listens to `/tf` and periodically republishes any transforms that have changed by a given amount on the `/tf_changes` topic.

7.2.1 Subscribed Topics

`/tf` (tf/tfMessage (http://docs.ros.org/api/tf/html/msg/tfMessage.html))
: Transform tree.

7.2.2 Published Topics

`/tf_changes` (tf/tfMessage (http://docs.ros.org/api/tf/html/msg/tfMessage.html))
: Reduced transform tree.

7.2.3 Parameters

`~polling_frequency` (`float`, default: 10.0)
: Frequency (hz) at which to check for any changes to the transform tree.

`~translational_update_distance` (`float`, default: 0.1)
: Minimum distance between the origin of two frames for the transform to be considered changed.

`~angular_update_distance` (`float`, default: 0.1)
: Minimum angle between the rotation of two frames for the transform to be considered changed.

Keywords Transformation, Transformations, coordinate transform

Wiki: tf (last edited 2017-10-02 13:40:32 by jarvisschultz (/jarvisschultz))