ROS-I Basic Training "Mobility"
ROS tf2 Tutorial - Turtlebot 3

Instructor: MASCOR Institute

2017ff

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Contents

1 Introduction

This tutorial explains static and dynamic transforms. ROS is using the package tf2 as an easy accessible and intuitive API to describe the relationship between coordinate frames. For a detailed explanation, see the online documentation http://wiki.ros.org/tf2. In addition, you will get used to the simulated Turtlebot3 2D laser scanner. You will create a fake localization application representing a motion of the robot.

- Lines beginning with $ are terminal commands
- Lines beginning with # indicate the syntax of the commands

2 Terminal usage

- opening a new terminal : ctrl+alt+t
- opening a new tab inside an existing terminal : ctrl+shift+t
- killing an active process inside a terminal: ctrl+c

3 Creating a static transform

A static transform is an invariant coordinate transformation and describes the relationship between two frames. A definition can be done through launch files by using the ***static_transform_publisher*** node, which contains a publisher to the */tf* topic. Provide the following arguments to describe the relationship between a parent and a child frame:

```
Arguments: x y z yaw pitch roll parent_frame child_frame
```

The translation is in meters and the rotation in radians.

H2020 funded GA no. 732287

Create a new package called **turtlebot3_sim** and in there a launch file called *turtlebot_tf.launch*. Use the given example as a skeleton:

```
<launch>

    <node pkg="tf2_ros" type="static_transform_publisher" name="
       ↪ parent_to_child" args="0 0 0 0 0 0 parent child" />

</launch>
```

Create the transformation tree of the turtlebot by starting multiple running processes of the node ***static_transform_publisher***. Take care to name them differently:

- Relationship between the frames base_footprint and base_link:
  - Translation: t = (x, y, z) = (0, 0, 0.01)
  - Rotation: r = (roll, pitch, yaw) = (0, 0, 0)
- Relationship between the frames base_link and camera_link:
  - Translation: t = (0.06, 0, 0.11)
  - Rotation: r = (-1.57, 0, -1.57)
- Relationship between the frames base_link and imu_link:
  - Translation: t = (-0.032, 0, 0.068)
  - Rotation: r = (0, 0, 0)
- Relationship between the frames base_link and laser_link:
  - Translation: t = (-0.032, 0, 0.0171)
  - Rotation: r = (0, 0, 0)

Visualize the tf tree in RViz:

```
$ roslaunch turtlebot3_sim turtlebot_tf.launch
$ rosrun rviz rviz
```

Click the **Add** button and choose **TF**, to visualize the tf information.

Hint: Switch your fixed_frame in RVIZ to the start frame of your tf tree, what is right now base_footprint.

→ You should see the different coordinate frames of your turtlebot, based on your static transforms.

4 Creating a dynamic transform

Dynamic transforms are variant transformations, which change over time. The convenient way is to use the Python API to define frame relationships.

4.1 Exercise

Write a transform Broadcaster and define a dynamic relationship between the frames *map* and *odom*. Therefore, create a node named ***simulated_localization.py*** in the package **turtlebot3_sim**. By providing changes in the translation values x and y, the Turtlebot takes a simulated localization. Usually the relation between map and odom is used to calculate the position of the robot. In this example we will fake the localization process by simply providing new positions of the *odom* frame (relative to *map*). Use the following mathematical definition and the Python code example.

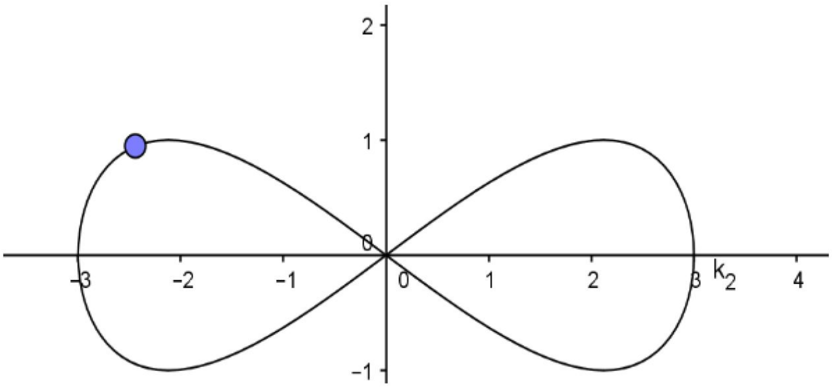

Figure 1: Virtual Turtlebot Path

$$\overrightarrow{Turtlebot} = \begin{pmatrix} 3 * cos(t) \\ sin(2t) \end{pmatrix} with\ 0 \leq t \leq 2\pi \tag{1}$$

```
#!/usr/bin/env python

import rospy
import math
import tf
import tf2_ros
import tf_conversions
import geometry_msgs

br = tf2_ros.TransformBroadcaster()
cnt = 0
position_odom = [0.,0.]

def calculate_position():
    global cnt
    if cnt > 300:
        cnt = 0
```

```
    cnt = cnt+1
    x = 3* math.cos(cnt*0.02)
    y = math.sin(2*cnt*0.02)

position = (x,y)
    return position

def calculate_tf():
    t = geometry_msgs.msg.TransformStamped()
    #getting actual position
    position_odom = calculate_position()
    #calculating transform
    t.header.stamp = rospy.Time.now()
    t.header.frame_id = "map"
    t.child_frame_id = "odom"
    t.transform.translation.x = position_odom[0]
    t.transform.translation.y = position_odom[1]
    t.transform.translation.z = 0
    q = tf.transformations.quaternion_from_euler(0,0,0)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]
    return t

if __name__ == '__main__':
    rospy.init_node("dynamic_tf_broadcaster")
    rate = rospy.Rate(100)

    while not rospy.is_shutdown():
        odom_to_base_footprint = calculate_tf()
        #broadcasting transform
        br.sendTransform(odom_to_base_footprint)
        rate.sleep()
```

To create a new python node perform as follows:

H2020 funded GA no. 732287

- Create a new file in your package in the "scripts" or "nodes" subfolder:

```
$ roscd turtlebot3_sim
$ mkdir scripts
$ cd scripts
$ gedit simulated_localization.py
```

- After you created your node make sure that it is executable:

```
$ chmod +x simulated_localization.py
```

4.2 Explanation

The basic structure of a simple node including Publisher and Subscriber is already well known. So, only the tf specific lines will be explained.

```
import tf2_ros
import tf_conversions
```

These lines import the essential TF modules. The tf2_ros package provides ROS bindings to tf2. tf_conversions provides the popular transformations.py, which was included in tf but not in tf2, in order to have a cleaner package. Creates an object br of the TransformBroadcaster class.

```
br = tf2_ros.TransformBroadcaster()
```

This will initialize a ROS TF Broadcaster, which allows to send transforms from one frame to another one.

```
map_to_odom = calculate_tf()
```

Calculates the transform from map to odom

```
position_odom = calculate_position()
```

Returns the virtual x and y position of the turtlebot.

```
br.sendTransform(t)
```

The handler function *sendTransform()* for the turtlebot pose broadcasts the turtlebot's translation and rotation, and publishes it as a transform from frame "map" to frame "odom".
The sendTransform function is from the message type StampedTransform, which was set up in the subscriber callback:

- Header
- Timestamp (Determine the moment when this transform is happening. This is mainly rospy.Time.now() when you want to send the actual transform. This means the transform can change over time to generate a dynamic motion.)
- Frame_ID (The frame ID of the Origin Frame)
- Child Frame ID (Frame ID to which the transform is happening)
- Transform
- Position in m (X, Y and Z)
- Orientation in Quaternion (You can use the TF Quaternion from Euler function to use the roll, pitch and yaw angles in rad instead)

```
sendTransform(translation, rotation, time, child, parent)
```

parent	–	parent frame as string
translation	–	the translation of the transformation as a tuple (x, y, z)
rotation	–	the rotation of the transformation as a tuple (x, y, z, w)
time	–	the time of the transformation, as a rospy.Time()
child	–	child frame as string

4.3 Exercise

Run your node and visualize the dynamic tf tree in RViz:

```
$ rosrun turtlebot3_sim simulated_localization.py
$ rosrun rviz rviz
```

Make sure to visualize "TF" in RViz and to setup a proper fixed frame.
Note: This node can also be run while the turtlebot3_gazebo roundTrack_simulation.launch is running.