TF Tutorial

tf Tools

1. view_frames:创建正在被广播的 frame 图表

rosrun tf view_frames

可以看到能够生成一个 tf tree 的 pdf 文件,打开该文件:

evince frames.pdf

2. rqt_tf_tree:显示 tf tree 的实时工具

rosrun rqt_tf_tree rqt_tf_tree

或 *rqt &*

3. tf_monitor:监督 frame 之间的变换

rosrun tf tf_monitor

rosrun tf tf_monitor /base_link /odom

4. tf_echo:显示两个坐标之间的变换关系

rosrun tf tf_echo [reference_frame] [target_frame]

5. static_transform_publisher:发送静态变换的命令行工具，也可用在 launch 文件中，例

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

6. roswtf:打开 ftwtf 插件，追踪 tf 错误

roswtf

Writing a tf broadcaster(Python)

创建包:

```
cd ~/test/src
catkin_create_pkg learning_tf roscpp rospy turtlesim tf
cd ~/test
catkin_make
source ./devel/setup.bash
```

创建 tf broadcaster 节点:

```
roscd learning_tf
mkdir nodes
cd nodes
gedit turtle_tf_broadcaster.py
```

```python
#!/usr/bin/env python
import roslib
roslib.load_manifest('learning_tf')
import rospy

import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")
                    #The handler function for the turtle pose message broadcasts this turtle's translation and rotation,
                    #and publishes it as a transform from frame "world" to frame "turtleX".

if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
                     #This node takes a single parameter "turtle", which specifies a turtle name, e.g. "turtle1" or "turtle2".
    rospy.Subscriber('/%s/pose' % turtlename, turtlesim.msg.Pose, handle_turtle_pose, turtlename)
                     # Topic                    type            callback function   arguments(needed in  callback function)
                     #The node subscribes to topic "turtleX/pose" and runs function handle_turtle_pose on every incoming message.
    rospy.spin()
```

```
chmod +x turtle_tf_broadcaster.py
```

运行 broadcaster:

```
roscd learning_tf
mkdir launch
```

cd launch
gedit start_demo.launch

```
<launch>
   <!-- Turtlesim Node-->
   <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
   <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

   <node name="turtle1_tf_broadcaster" pkg="learning_tf" type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
     <param name="turtle" type="string" value="turtle1" />
   </node>
   <node name="turtle2_tf_broadcaster" pkg="learning_tf" type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
     <param name="turtle" type="string" value="turtle2" />
   </node>

 </launch>
```

roslaunch learning_tf start_demo.launch
出现一只乌龟，可以通过键盘操作
检查结果：
使用 tf_echo 工具来检查乌龟的姿态是否已经播放到 tf：
rosrun tf tf_echo /world /turtle1
可以看到实时输出的 turte1 相对于 world 的姿态

Writing a tf listener(Python)

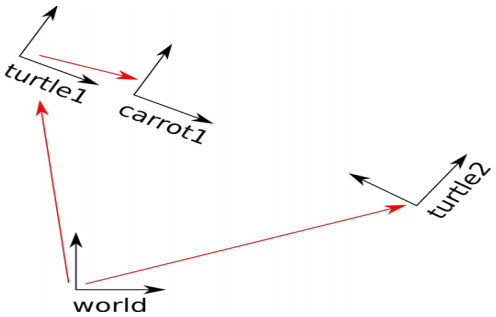
cd ../nodes
gedit turtle_tf_listener.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import roslib
roslib.load_manifest('learning_tf')
import rospy
import math
import tf #The tf package provides an implementation of a tf.TransformListener to help make the task of receiving transforms easier.
import geometry_msgs.msg
import turtlesim.srv
if __name__ == '__main__':
    rospy.init_node('tf_turtle')
    listener = tf.TransformListener()
       #we create a tf.TransformListener object. Once the listener is created, it starts receiving tf transformations over the wire.
    rospy.wait_for_service('spawn') #等待服务spawn,这个服务可以创建另一只乌龟
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn) #client端用proxy代理建立同service的通信
    spawner(4, 2, 0, 'turtle2') #位置跟名字
    turtle_vel = rospy.Publisher('turtle2/cmd_vel', geometry_msgs.msg.Twist,queue_size=1)
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))
            #we query the listener for a specific transformation by lookupTransform. Let's take a look at the arguments:
            #(We want the transform from this frame , to this frame , The time at which we want to transform. Providing rospy.Time(0) will
just get us the latest available transform. )
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
            continue
        angular = 4 * math.atan2(trans[1], trans[0])
        linear = 0.5 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
        cmd = geometry_msgs.msg.Twist()
        cmd.linear.x = linear
        cmd.angular.z = angular
        turtle_vel.publish(cmd)
        rate.sleep()
```

chmod +x turtle_tf_listener.py
gedit ../launch/start_demo.launch
在 start_demo.launch 文件中添一句：
<node pkg="learning_tf" type="turtle_tf_listener.py" name="listener" />
此时在运行 launch file：
roslaunch learning_tf start_demo.launc
出现两只乌龟，可以通过键盘操作一只，另一只自动跟随。

Adding a frame(Python)

现在 tf tree 包括三个坐标：world,turtle1,turtle2，后两个坐标是 world 的自坐标。如我们再添加一个 world 的自坐标:carrot1，总是相对于 turtle1 偏移 2 个单位。如下所示：



Add new frame:

roscd learning_tf

gedit nodes/fixed_tf_broadcaster.py

```
#!/usr/bin/env python
import roslib
roslib.load_manifest('learning_tf')

import rospy
import tf

if __name__ == '__main__':
    rospy.init_node('my_tf_broadcaster')
    br = tf.TransformBroadcaster()
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        br.sendTransform((0.0, 2.0, 0.0),
         #Broadcasting a moving frame:   br.sendTransform((2.0 * math.sin(t), 2.0 * math.cos(t), 0.0),
                         (0.0, 0.0, 0.0, 1.0),
                         rospy.Time.now(),
                         "carrot1",
                         "turtle1")  # we create a new transform, from the parent "turtle1" to the new child "carrot1".
                                     # The carrot1 frame is 2 meters offset from the turtle1 frame.
        rate.sleep()
```

chmod +x nodes/fixed_tf_broadcaster.py

roslaunch learning_tf start_demo.launch

在 start_demo.launch 文件中添一句：

<node pkg="learning_tf" type="fixed_tf_broadcaster.py" name="broadcaster_fixed" />

打开 nodes/turtle_tf_listener.py，用 carrot1 替换 turtle1，如下所示：

(trans,rot) = listener.lookupTransform("/turtle2", "/carrot1", rospy.Time(0))

运行 launch 文件：

roslaunch learning_tf start_demo.launch

此时，用键盘控制 turtle1 运动，由于 carrot1（没有显示）相对于 turtle1 偏移 2m，因此可以看到 turtle2 一直随着 turtle1 的运动，往其偏移 2m 的位置运动。

上边的例子 carrot1 的 frame 相对于 turtle1 是固定的，也可以规定变化的 frame,见 fixed_tf_broadcaster.py 的注释。

Time travel with tf(Python)

试想我们这次需要的是让 turtle2 朝着 carrot1 5s 前的位置移动，turtle2 朝着 carrot1 运动是由 turtle_tf_listener.py 程序规定的，打开这个程序

roscd learning_tf

重写异常处理语句：

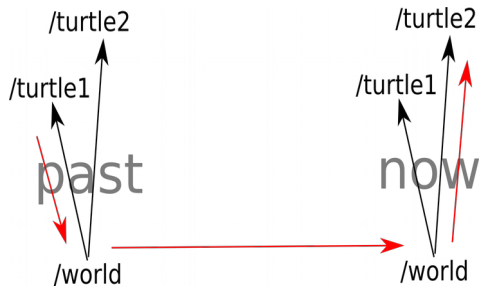
```
        try:
            now = rospy.Time.now() - rospy.Duration(5.0)
            listener.waitForTransform("/turtle2", "/carrot1", now, rospy.Duration(1.0))
# We spawned turtle2 but tf may have not ever seen the /turtle2 frame before waiting for the transform. The first waitForTransform() will
wait until the /turtle2 frame is broadcast on tf before trying to waitForTransform() at time now.
# waitForTransform() takes four arguments:
#   (Wait for the transform from this frame,to this frame,at this time,timeout: don't wait for longer than this maximum duration)
            (trans, rot) = listener.lookupTransform("/turtle2", "/carrot1", now)
        except (tf.Exception, tf.LookupException, tf.ConnectivityException):
            continue
```

保存后运行 launch 文件，通过在命令终端控制 turtle1 运动，观察 5s 后 turtle2 朝着 carrot1 的方向（偏移 turtle1 2m）运动。但是结过并非如预期一样，整个过程 turtle2 的运动特别没有规律。回头再看改写的程序，我们用了 lookupTransform 语句，虽然时间设到了 5s 前，但是整个语句的真实作用为：5s 前的 turtle2 朝着 5s 前的 carrot1 运动，而我们真正想达到的是现在的 turtle1 朝着 5s 前的 carrot 运

动。重新修改代码：

```
            try:
              now = rospy.Time.now()
              past = now - rospy.Duration(5.0)
              listener.waitForTransformFull("/turtle2", now, "/turtle1", past, "/world", rospy.Duration(1.0))
              (trans, rot) = listener.lookupTransformFull("/turtle2", now, "/turtle1", past, "/world")
        #The advanced API for lookupTransform() takes six arguments:
        #   (Give the transform from this frame, at this time, to this frame, at this time, Specify the frame that does not change over time,the
        variable to store the result in)
            except (tf.Exception, tf.LookupException, tf.ConnectivityException):
              continue
```

Notice that waitForTransform() also has a basic and and advanced API, just like lookupTransform().



此时，再运行 launch 文件：

roslaunch learning_tf start_demo.launch

控制 turtle1,观察 turtle2，结果和预想的一样。

C++版

Writing a tf broadcaster

```
. ~/test/devel/setup.bash
roscd learning_tf
gedit src/turtle_tf_broadcaster.cpp
```

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
        //The tf package provides an implementation of a TransformBroadcaster to help make the task of publishing transforms easier.
#include <turtlesim/Pose.h>

std::string turtle_name;



void poseCallback(const turtlesim::PoseConstPtr& msg){
  static tf::TransformBroadcaster br; //create a TransformBroadcaster object that we'll use later to send the transformations over the wire.
  tf::Transform transform;
  transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );
  tf::Quaternion q;
  q.setRPY(0, 0, msg->theta);
  transform.setRotation(q); //Here we create a Transform object, and copy the information from the 2D turtle pose into the 3D transform.
  br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));
          //This is where the real work is done. Sending a transform with a TransformBroadcaster requires four arguments:
//(pass in the transform itself, give the transform being published a timestamp, pass the name of the parent frame, pass the name of the
child frame)
}

int main(int argc, char** argv){
  ros::init(argc, argv, "my_tf_broadcaster");
  if (argc != 2){ROS_ERROR("need turtle name as argument"); return -1;};
  turtle_name = argv[1];

  ros::NodeHandle node;
  ros::Subscriber sub = node.subscribe(turtle_name+"/pose", 10, &poseCallback);

  ros::spin();
  return 0;
};
```

打开 learning_tf 包下的 CMakeLists.txt，添加下面两行代码

```
add_executable(turtle_tf_broadcaster src/turtle_tf_broadcaster.cpp)
target_link_libraries(turtle_tf_broadcaster ${catkin_LIBRARIES})
```

Writing a tf listener

gedit src/turtle_tf_listener.cpp

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
        //The tf package provides an implementation of a TransformListener to help make the task of receiving transforms easier.
#include <turtlesim/Velocity.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv){
  ros::init(argc, argv, "my_tf_listener");

  ros::NodeHandle node;

  ros::service::waitForService("spawn");
  ros::ServiceClient add_turtle =
    node.serviceClient<turtlesim::Spawn>("spawn");
  turtlesim::Spawn srv;
  add_turtle.call(srv);

  ros::Publisher turtle_vel =
    node.advertise<turtlesim::Velocity>("turtle2/command_velocity", 10);

  tf::TransformListener listener;
     //create a TransformListener object. Once the listener is created, it starts receiving tf transformations over the wire, and buffers
them for up to 10 seconds.

  ros::Rate rate(10.0);
  while (node.ok()){
    tf::StampedTransform transform;
    try{
      listener.lookupTransform("/turtle2", "/turtle1",
                               ros::Time(0), transform);
    }
```

打开 learning_tf 包下的 CMakeLists.txt，添加下面两行代码

```
add_executable(turtle_tf_listener src/turtle_tf_listener.cpp)
target_link_libraries(turtle_tf_listener ${catkin_LIBRARIES})
```

编译

```
cd ~/test
catkin_make
```

然后在 learning_tf 包下的 launch 文件夹下建立 start_demo_C.launch

```
    <launch>
      <!-- Turtlesim Node-->
      <node pkg="turtlesim" type="turtlesim_node" name="sim"/>

      <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
      <!-- Axes -->
      <param name="scale_linear" value="2" type="double"/>
      <param name="scale_angular" value="2" type="double"/>

      <node pkg="learning_tf" type="turtle_tf_broadcaster"
            args="/turtle1" name="turtle1_tf_broadcaster" />
      <node pkg="learning_tf" type="turtle_tf_broadcaster"
            args="/turtle2" name="turtle2_tf_broadcaster" />

       <node pkg="learning_tf" type="turtle_tf_listener" name="listener" />

    </launch>
```

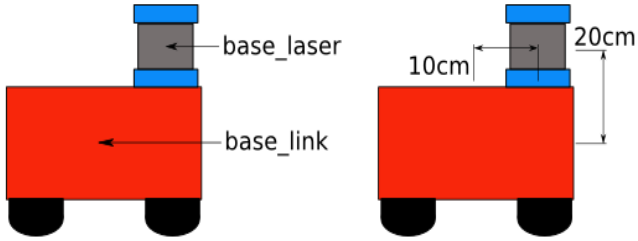
执行：
roslaunch learning_tf start_demo_C.launch
可以看到出现两只乌龟，在命令终端操作键盘控制 turtle1，turtle2 随之运动。刚开始可能会出现上图说示的错误：

```
[ERROR] [1490708342.329733210]: "turtle2" passed to lookupTransform argument tar
get_frame does not exist.
```

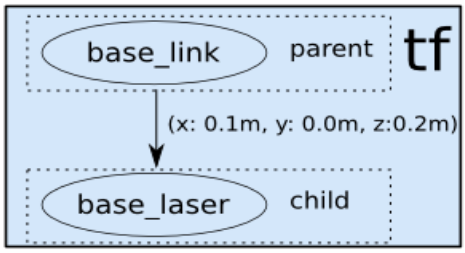
This happens because our listener is trying to compute the transform before messages about turtle 2 have been received because it takes a little time to spawn in turtlesim and start broadcasting a tf frame.

Debugging tf problems

Setting up your robot using tf



如图所示，在一个小车上安装一个激光雷达，激光雷达感知环境信息，而小车根据采集到的信息进行壁障，有必要对激光雷达采集到的坐标值进行变换为相对于小车的坐标值，如下所示：



编程实现：
cd ~/test/src
catkin_create_pkg robot_tf roscpp tf geometry_msgs
Broadcasting a Transform
在 robot_tf 功能包 src 文件夹下编写 tf_broadcaster.cpp

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
    //The tf package provides an implementation of a tf::TransformBroadcaster to help make the task of publishing transforms easier. To use the TransformBroadcaster, we need to include the tf/transform_broadcaster.h header file.

int main(int argc, char** argv){
  ros::init(argc, argv, "robot_tf_publisher");
  ros::NodeHandle n;

  ros::Rate r(100);

  tf::TransformBroadcaster broadcaster;
    //Here, we create a TransformBroadcaster object that we'll use later to send the base_link → base_laser transform over the wire.

  while(n.ok()){
    broadcaster.sendTransform(
      tf::StampedTransform(
        tf::Transform(tf::Quaternion(0, 0, 0, 1), tf::Vector3(0.1, 0.0, 0.2)),
        ros::Time::now(),"base_link", "base_laser"));
// This is where the real work is done. Sending a transform with a TransformBroadcaster requires five arguments. First, we pass in the rotation transform, which is specified by a btQuaternion for any rotation that needs to occur between the two coordinate frames. In this case, we want to apply no rotation, so we send in a btQuaternion constructed from pitch, roll, and yaw values equal to zero. Second, a btVector3 for any translation that we'd like to apply. We do, however, want to apply a translation, so we create a btVector3 corresponding to the laser's x offset of 10cm and z offset of 20cm from the robot base. Third, we need to give the transform being published a timestamp, we'll just stamp it with ros::Time::now(). Fourth, we need to pass the name of the parent node of the link we're creating, in this case "base_link." Fifth, we need to pass the name of the child node of the link we're creating, in this case "base_laser."
    r.sleep();
  }
}
```

Using a Transform

在 robot_tf 功能包 src 文件夹下编写 tf_listener.cpp

```
#include <ros/ros.h>
#include <geometry_msgs/PointStamped.h>
#include <tf/transform_listener.h>

void transformPoint(const tf::TransformListener& listener){
  //we'll create a point in the base_laser frame that we'd like to transform to the base_link frame
  geometry_msgs::PointStamped laser_point;
  laser_point.header.frame_id = "base_laser";

  //we'll just use the most recent transform available for our simple example
  laser_point.header.stamp = ros::Time();

  //just an arbitrary point in space
  laser_point.point.x = 1.0;
  laser_point.point.y = 0.2;
  laser_point.point.z = 0.0;

  try{
    geometry_msgs::PointStamped base_point;
    listener.transformPoint("base_link", laser_point, base_point);

    ROS_INFO("base_laser: (%.2f, %.2f. %.2f) -----> base_link: (%.2f, %.2f, %.2f) at time %.2f",
        laser_point.point.x, laser_point.point.y, laser_point.point.z,
        base_point.point.x, base_point.point.y, base_point.point.z, base_point.header.stamp.toSec());
  }
  catch(tf::TransformException& ex){
    ROS_ERROR("Received an exception trying to transform a point from \"base_laser\" to \"base_link\": %s", ex.what());
  }
}
int main(int argc, char** argv){
  ros::init(argc, argv, "robot_tf_listener");
  ros::NodeHandle n;

  tf::TransformListener listener(ros::Duration(10));

  //we'll transform a point once every second
  ros::Timer timer = n.createTimer(ros::Duration(1.0), boost::bind(&transformPoint, boost::ref(listener)));

  ros::spin();

}
```

打开功能包下的CmakeLists.txt,粘贴下面语句:

```
add_executable(tf_broadcaster src/tf_broadcaster.cpp)
add_executable(tf_listener src/tf_listener.cpp)
target_link_libraries(tf_broadcaster ${catkin_LIBRARIES})
target_link_libraries(tf_listener ${catkin_LIBRARIES})
```

编译:

```
cd ~/test
catkin_make
```

运行:

```
roscore
source ~/test/devel/setup.bash
rosrun robot_tf tf_broadcaster
rosrun robot_tf tf_listener
```

可以看到输出的转换信息:

```
[ INFO] [1490708901.337630326]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1490708901.33
[ INFO] [1490708902.337303850]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1490708902.33
[ INFO] [1490708903.337791192]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1490708903.34
[ INFO] [1490708904.337334347]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1490708904.33
```

当机器人比较复杂，涉及的坐标变换很多时，ROS 提供了 state publisher 来简少这一工作量。具体可见 Using the robot state publisher on your own robot

This tutorial gives a full example of a robot model with URDF that uses robot_state_publisher