清华大学无人机课程第二次作业 在第一次作业中，我们能够用话题topic和服务service两种服务方式传输数据。在ROS实际应用中，来自相机传感器的图片经常作为传输所需的数据类型，本次作业首先讲解图片的传递&RVIZ可视化，随后学习ROS系统中的tf的使用，最终手动实现2个turtle追踪同一个turtle的代码设计。

实验准备：安装OpenCV库

```
sudo apt install libopencv-dev
```

任务一：怎么使用ROS的话题传输图片？

如果想使用话题topic传输图片，则需要首先定义图片的数据类型，而ROS系统中内建了一个图片数据类型sensor_msgs/Image：其中：

Header记录的图片的时间信息、坐标系信息等等；

height记录图片的行数；width记录图片的列数；

encoding为图排尿的编码方式，可以中include/sensor_msgs/image_encodings.h查看候选数值；

step为图片中的一整行的数据长度(以Byte为单位)。

data记录了具体的像素信息（按照行进行索引）。

我们可以读取外部图片每个像素点的数值并且手动定义每一个内部成员变量，然后创建一个以该消息为内容的消息 topic将其publish出去，但这样操作过于繁琐。一个较为简单的做法是使用开源库OpenCV读取外部图片。由于OpenCV读取的外部图片为内建的Mat类型，该类型与sensor_msgs/Image类型不同，因此需要从Mat转换称为sensor_msgs/Image。ROS内部也提供了格式转换所需的可函数CvBridge。因此接下来我们采用OpenCV读取外部图片存储为Mat格式，通过CvBridge将其转换为sensor_msgs/Image并创建消息topic将其发出。具体代码如下：

```
$ cd ~/catkin_ws/src/
$ catkin_create_pkg image_tran rospy roscpp std_msgs image_transport cv_bridge
$ cd image_tran
$ mkdir scripts && cd scripts
$ gedit image_pub_node.py
$ sudo chmod +x ~/catkin_ws/src/image_tran/scripts/image_pub_node.py
```

将下列内容写入image_pub_node.py

```
#!/usr/bin/env python
#coding:utf-8

import rospy
import sys
import cv2
import os
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

def pubImage():
    rospy.init_node('pubImage', anonymous = True)
    pub = rospy.Publisher('ShowImage', Image, queue_size = 10)
    rate = rospy.Rate(10)
    bridge = CvBridge()
    get_image_name = []
    path = "/home/jimmy/work/ROS_work/test_ws/src/image_tran/dataset/"
    for item in os.listdir(path):
        get_image_name.append(os.path.join(path, item))
    while not rospy.is_shutdown():
        for imagepath in get_image_name:
            image = cv2.imread(imagepath)
            image = cv2.resize(image, (900, 450))
            pub.publish(bridge.cv2_to_imgmsg(image, "bgr8"))

            rate.sleep()

if __name__ == '__main__':
    try:
        pubImage()
    except rospy.ROSInterruptException:
        pass
```

修改文件中path为同学电脑中图片的路径位置。 代码解读：

1. 修改下面的路径为本地存储图片的路径，图片在资料中picture_data.zip中。

```
path = "/home/jimmy/work/ROS_work/test_ws/src/image_tran/dataset/"
```

2. 以下代码实现从path指定的路径中获取所有图片的文件名称。

```
for item in os.listdir(path)：
    get_image_name.append(os.path.join(path,item))
```

3. 以下代码从path指定的路径中读取每一张图片并将图片放缩成宽900，高450的格式。

```
for imagepath in get_image_name:
    image = cv2.imread(imagepath)
    image = cv2.resize(image, (900, 450))
```

4. bridge创建了CvBridge()的对象，并且通过调用cv2_to_imgmsg函数将OpenCV读取的Mat类型图片转换为sensor_msgs/Image类型，最终发布。

```
bridge = CvBridge()
pub.publish(bridge.cv2_to_imgmsg(image, "bgr8"))
```

至此，我们完成了话题发布者脚本的撰写。 话题接受者如下：

```
$ cd ~/catkin_ws/src/image_tran/scripts
$ gedit image_sub_node.py
$ sudo chmod +x ~/catkin_ws/src/image_tran/scripts/image_sub_node.py
```

将下列内容输入image_sub_node.py

```
#!/usr/bin/env python
#coding:utf-8

import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge,CvBridgeError

def callback(data):
    bridge = CvBridge()
    cv_image = bridge.imgmsg_to_cv2(data, "bgr8")
    cv2.imshow("view", cv_image)

def showImage():
    rospy.init_node('showImage', anonymous = True)
    cv2.namedWindow("view", cv2.WINDOW_AUTOSIZE)
    cv2.startWindowThread()
    rospy.Subscriber('ShowImage', Image, callback)
    rospy.spin()
    cv2.destroyWindow("view")

if __name__ == '__main__':
    showImage()
```

代码解读：

1. 主函数调用进入showImage()函数。该函数中下面两行代码创建了OpenCV用于显示图片名为“view”的窗口并创建一个新的线程运行窗口的更新。该窗口一直存在直至调用 `cv2.destroyWindow("view")`

```
cv2.namedWindow("view", cv2.WINDOW_AUTOSIZE)
cv2.startWindowThread()
```

2. 回调函数中的代码创建了一个CvBridge()并且调用imgmsg_to_cv2将接受的sensor_msgs/Image类型消息转化为Mat并用于后续imshow的图片显示。imshow第一参数指定了显示的窗口名称。

```
    bridge = CvBridge()
    cv_image = bridge.imgmsg_to_cv2(data, "bgr8")
    cv2.imshow("view", cv_image)
```

至此，所有准备工作就绪。编译运行

终端1

```
$ cd ~/catkin_ws
$ catkin_make
$ roscore
```

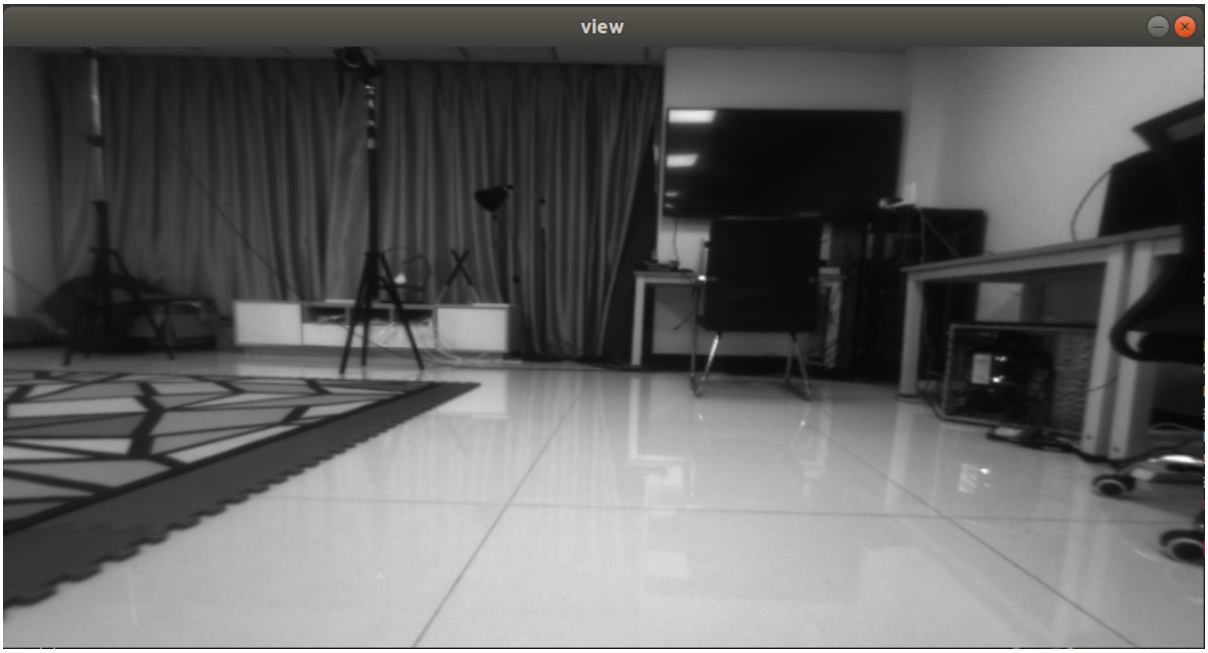
终端2

```
$ source ~/catkin_ws/devel/setup.bash
$ rosrun image_tran image_pub_node.py
```

终端3

```
$ source ~/catkin_ws/devel/setup.bash
$ rosrun image_tran image_sub_node.py
```

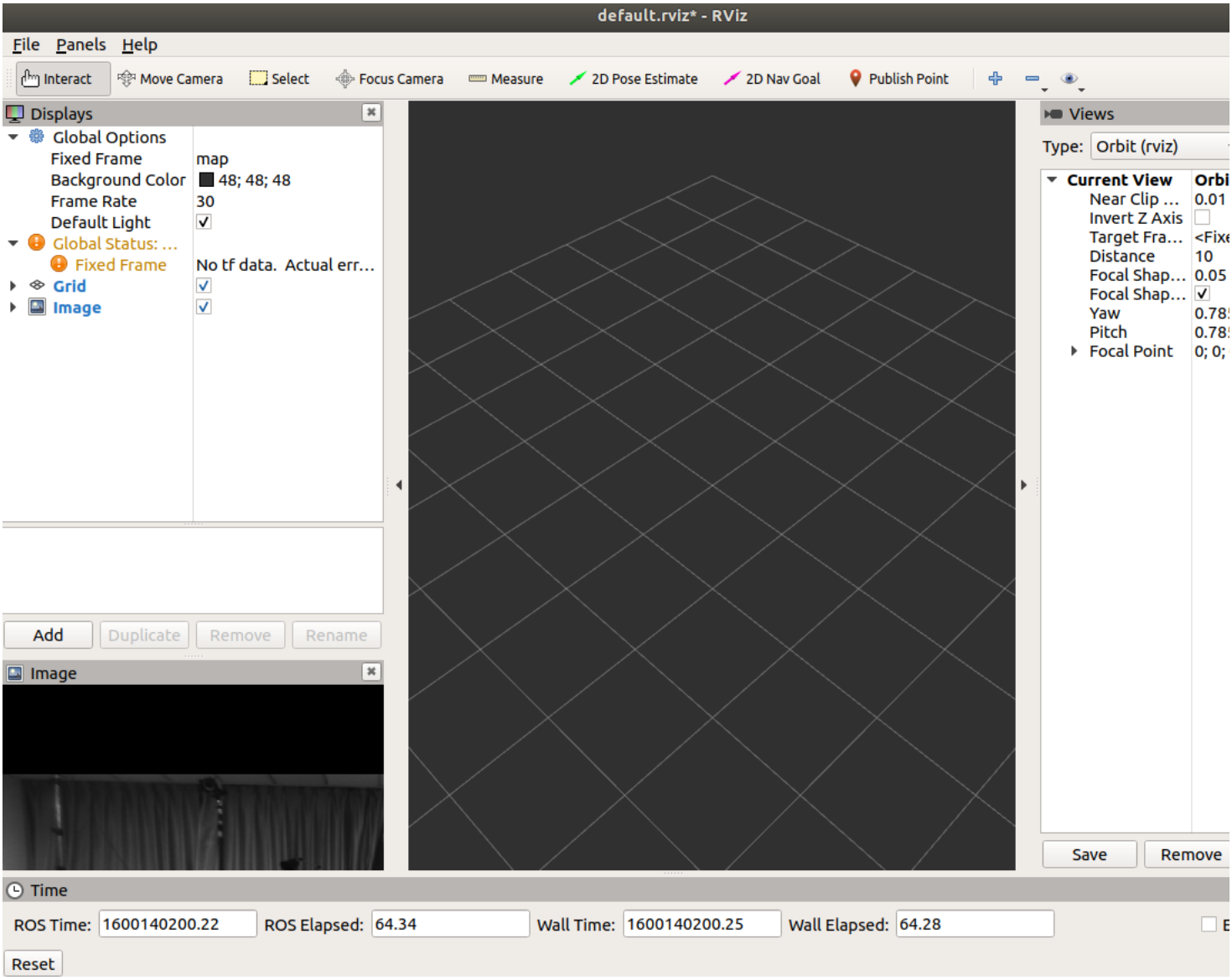
此时，可以看到终端3运行后会新创建一个窗口循环显示path路径下的图片。



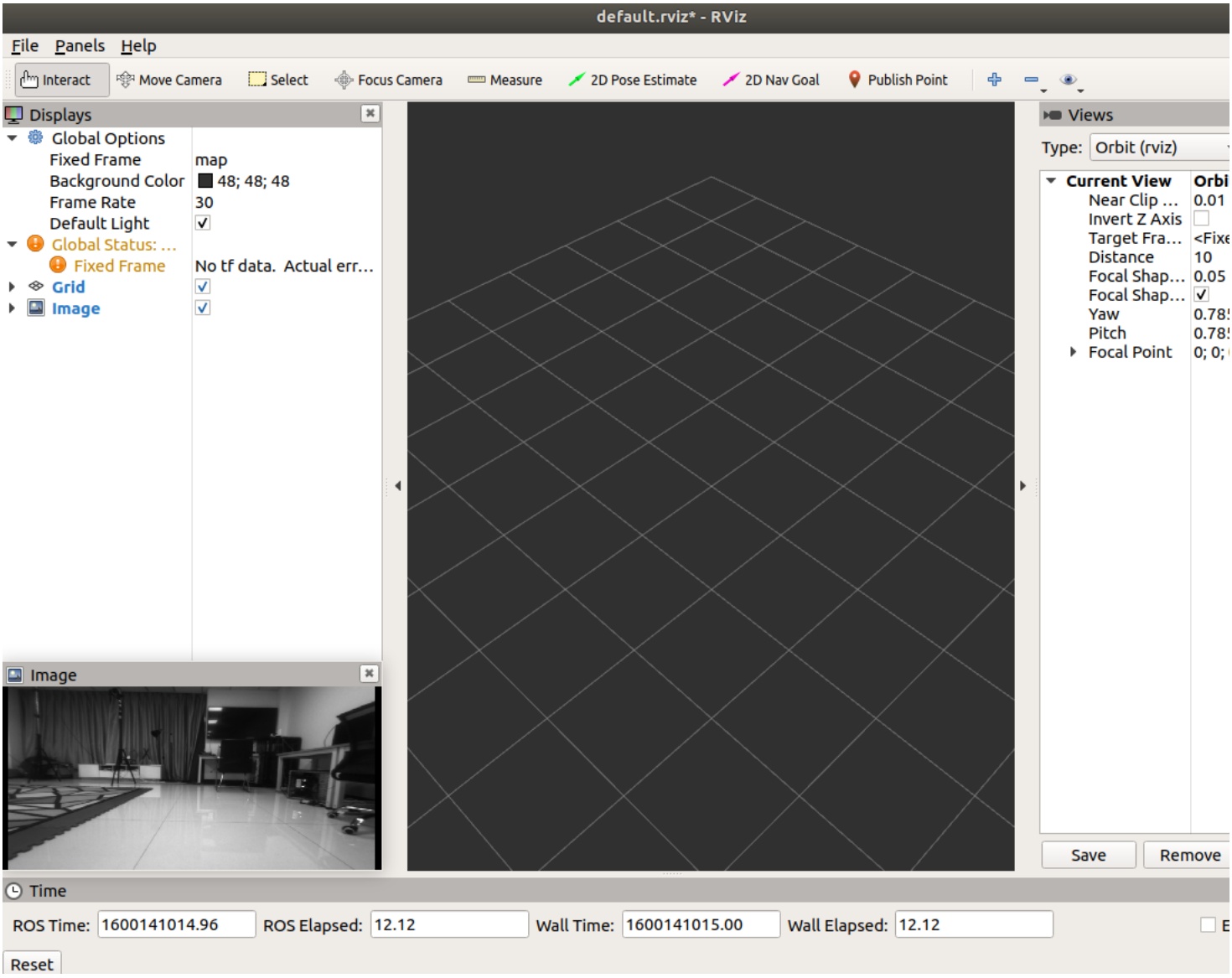
我们也可以使用RVIZ来观察当前ROS系统中传递的图片消息。

```
rosrun rviz rviz
```

选中窗口中左下角add选项->By topic->/ShowImage->Image，左下角会显示传输的图片。



双击图片的Image状态栏可视图片自动放缩到合适的大小。



任务二：tf能做什么？

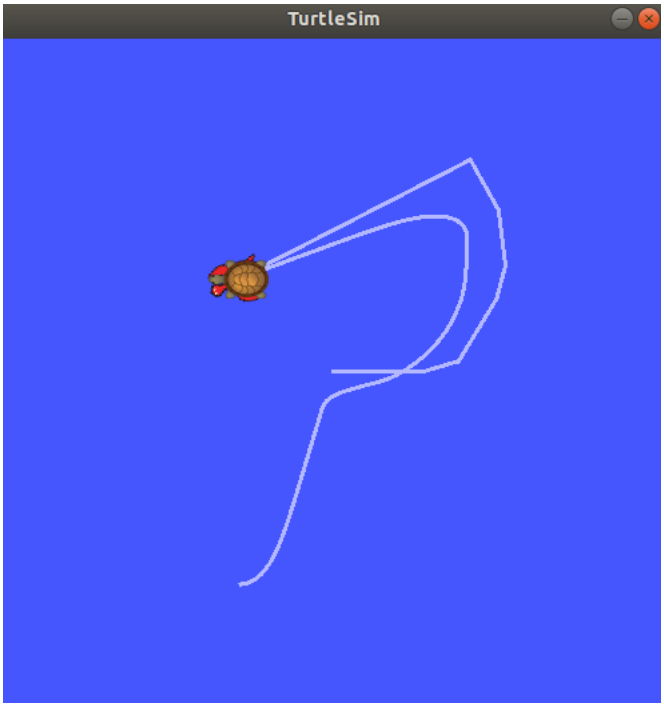
本任务通过一系列演示程序旨在帮助大家理解tf能够完成什么功能。

安装tf相关的依赖包，如下：

```
$ sudo apt-get install ros-melodic-ros-tutorials ros-melodic-geometry-tutorials ros-melodic-rviz ros-melodic-rosbash ros-melodic-rqt-tf-tree
```

终端1运行demo

```
$ roslaunch turtle_tf turtle_tf_demo.launch
```



在终端1中使用键盘方向键操作乌龟移动，发现另一只乌龟紧随其后进行跟踪。

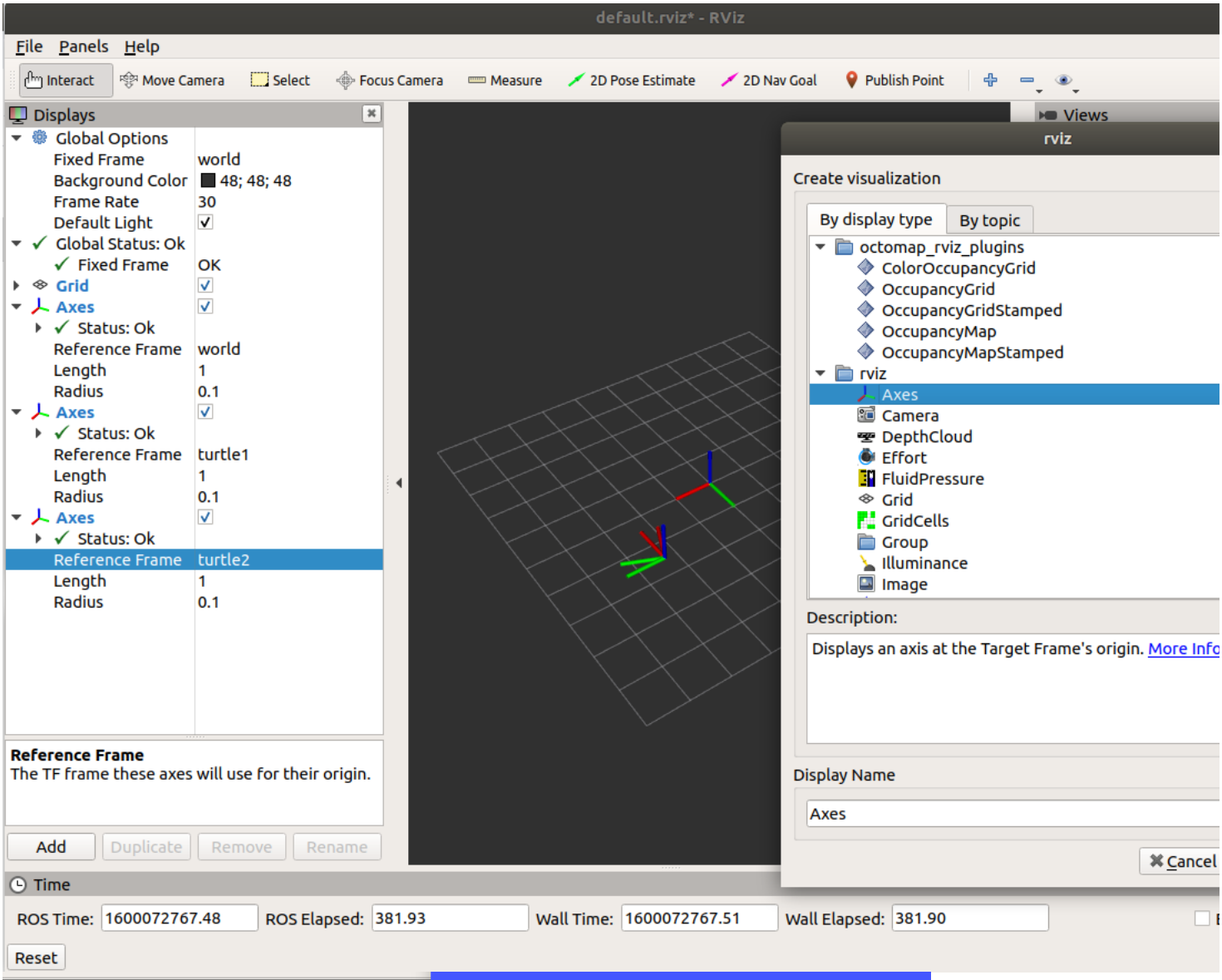
这个demo是如何做到的呢？ 此demo使用的是tf包提供的独特的通信方式（广播/监听：broadcaster/listener），即tf中的话题。 为了实现turtle2追踪turtle1，需要控制turtle2向turtle1的位置移动。那么问题来了，如何使turtle2得知turtle1的位置？这就需要用到tf包。

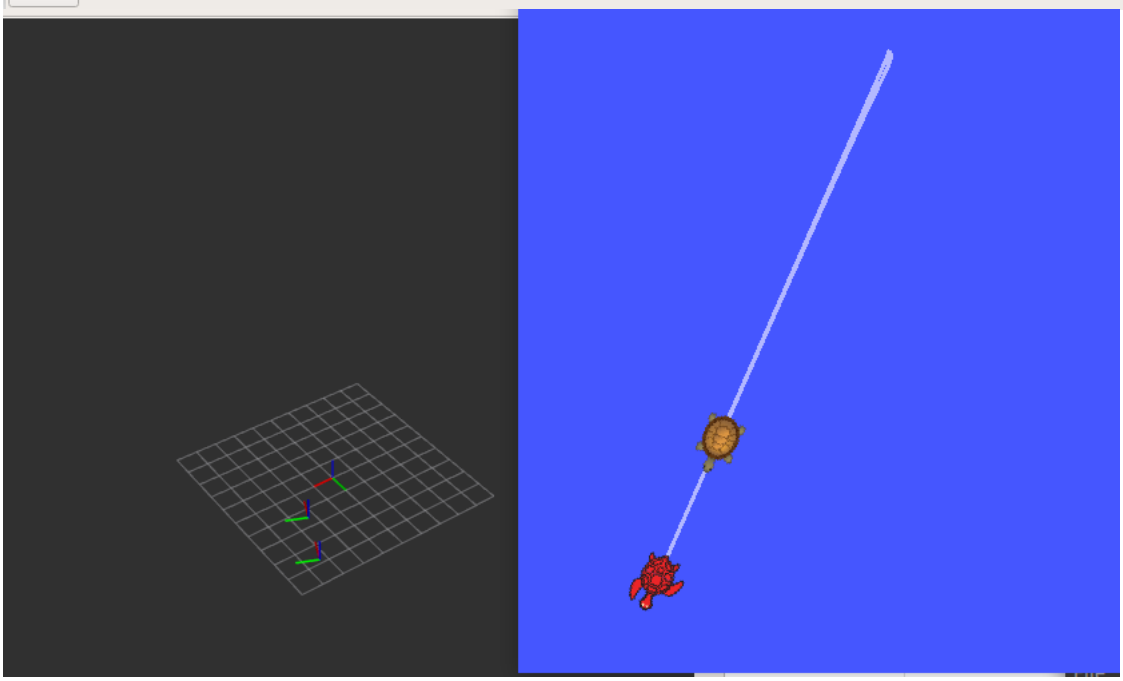
tf全称叫做transform，该库提供了坐标变换所需的常用函数。在上述例子中一共存在3个坐标系: ROS内建了名为RVIZ的可视化工具

```
rosrun rviz rviz
```

打开后左侧便是当前显示的话题名称和内容。Global Options中的Fixed Frame需要更改为world。因为Fixed Frame为了当前RVIZ显示的参考系。初始化时为map frame，但在本例中并不存在map。

通过左下角add按钮添加By display type中的rviz文件夹中的Axes并按照下图中配置每个Axes中的Refernce Frame分别为turtle1, turtle2和world，我们可以直观地看到一下三个坐标系。





- turtle1的坐标系(frame ID: turtle1)：以turtle1中心为原点，turtle前方为x轴正方向，turtle左手边为y轴正方向，采用右手螺旋定则，四指从x正向转到y轴正向时拇指的指向为z轴正方向。
- turtle2的坐标系(frame ID: turtle2)：以turtle2中心为原点，正方向同上。
- 世界坐标系(frame ID: world)。

由于Global Options中Fixed Frame选择为world，因此只有世界坐标系固定，而两只turtle的坐标系都在移动。

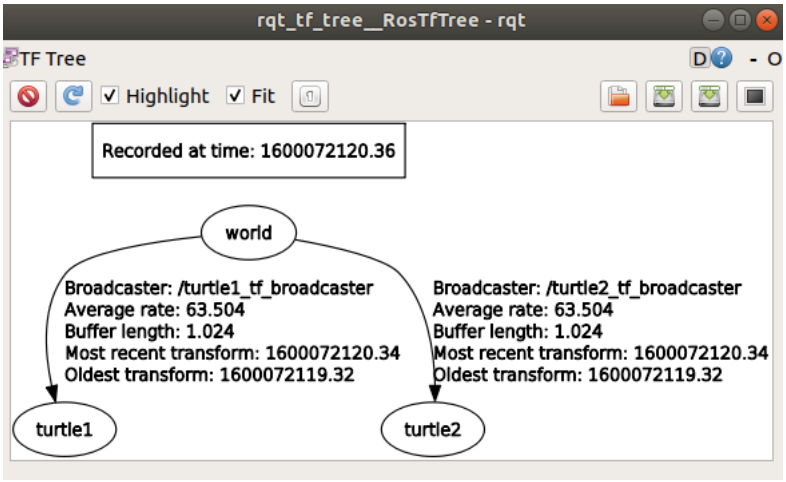
注意：turtle1的世界坐标系下坐标 就是 world frame->turtle1 frame的坐标转换。而上述例子中turtle2追踪turtle1即计算了turtle1在turtle2的坐标，即turtle2->turtle1的坐标变换。

为了更直观地展示turtle2->turtle1的坐标变换，我们还可以监听tf的数据 `$ rosrun tf tf_echo [reference_frame] [target_frame]`

```
$ rosrun tf tf_echo turtle2 turtle1
```

已知turtle1在turtle2坐标系中的坐标（即turtle2->turtle1的坐标变换），我们即可控制turtle2直接向turtle1位置移动，从而实现追踪。 最后，我们也可以通过下述命令查看当前ROS系统中的各个坐标系之间的变换关系

```
rosrun rqt_tf_tree rqt_tf_tree
```



任务三：手写tf脚本实现turtle跟踪

任务一底层的实现方式为：turtle1通过tf广播/监听通信方式广播world->turtle1的坐标变换（turtle1在世界坐标系下的坐标）；同时turtle2也通过tf广播/监听通信方式广播world->turtle2的坐标变换（turtle2在世界坐标系下的坐标）。turtle1/turtle2 tf广播的作用就是将turtle1/turtle2发送到tf树中。tf树就是一个存储各个坐标系之间的变换关系的数据库，因为其采用树型组织，因此又称为transform tree简称tf树。而另外编写一个监听脚本，该脚本负责在tf树中监听是否存在turtle2->turtle1的坐标变换（`rosrun rqt_tf_tree rqt_tf_tree` 中存在turtle2节点 和 turtle1节点的链接 即存在turtle2->turtle1的坐标变换）。

部分一：广播脚本编写

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg learning_tf rospy roscpp tf turtlesim
$ cd learning_tf
$ mkdir scripts
$ gedit scripts/turtle_tf_broadcaster.py
```

将下列内容放入该python脚本

```
#!/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")

if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

为python脚本文件增加可执行权限。

```
$ chmod +x scripts/turtle_tf_broadcaster.py
```

代码解读（仅解释先前任务中未涉及的代码）：

1. `turtlename = rospy.get_param('~turtle')` 即从参数服务器中获取名为turtle参数的数值，该数值用于配置后续监听话题名称。
2. `rospy.Subscriber('/%s/pose' % turtlename, turtlesim.msg.Pose, handle_turtle_pose, turtlename)` 使该节点监听名为'<turtlename>/pose'的话题（其中<turtlename>为参数服务器中该参数的数值），每当从该话题中收到一个消息后即调用一次回调函数handle_turtle_pose。
3. ```
   br.sendTransform((msg.x, msg.y, 0), tf.transformations.quaternion_from_euler(0, 0, msg.theta), rospy.Time.now(), turtlename, "world")```广播了<turtlename>所指定world->turtlename的坐标变换（平移+旋转）。

完成了广播脚本的撰写，下面准备完成启动文件的撰写，即启动乌龟仿真节点，键盘控制节点以及两个坐标广播节点。

```
$ mkdir -p ~/catkin_ws/src/learning_tf/launch/
$ gedit ~/catkin_ws/src/learning_tf/launch/start_demo.launch
```

将下列内容写入start_demo.launch 文件

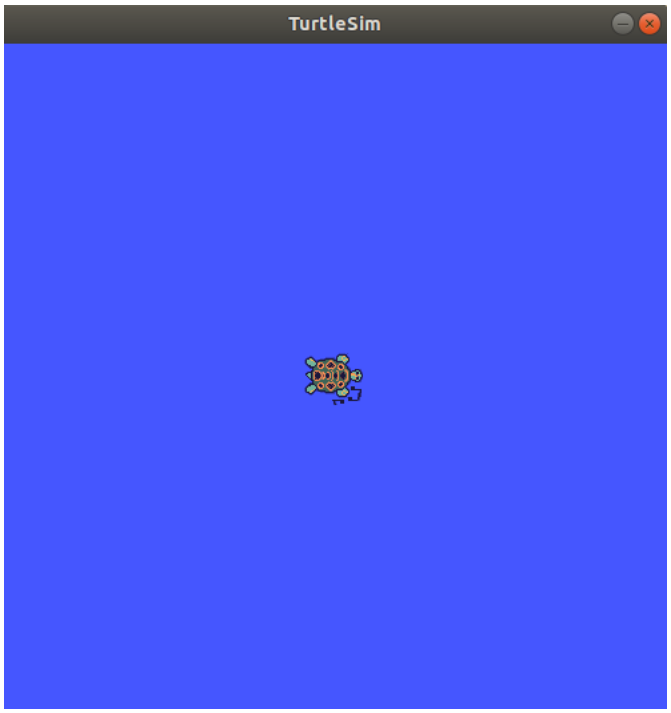
```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

  <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>
```

编译运行

```
$ cd  ~/catkin_ws
$ catkin_make
$ source ./devel/setup.bash
$ roslaunch learning_tf start_demo.launch
```

用户可以看到如下结果 

并且可以通过下面代码来查看turtle1是否成功广播插入tf树中。

```
$ rosrun tf tf_echo /world /turtle1
```

部分二：监听脚本撰写

```
$ gedit ~/catkin_ws/src/learning_tf/scripts/turtle_tf_listener.py
```

将下列内容写入文件中

```
#!/usr/bin/env python
import roslib
roslib.load_manifest('learning_tf')
import rospy
import math
import tf
import geometry_msgs.msg
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('turtle_tf_listener')

    listener = tf.TransformListener()

    rospy.wait_for_service('spawn')
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    spawner(4, 2, 0, 'turtle2')

    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))
        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()
```

增加文件的执行权限。

```
$ chmod +x scripts/turtle_tf_listener.py
```

代码解读：

1. `import tf` 与 `listener = tf.TransformListener()` 创建tf的监听，一旦tf完成创建，就开始接受tf变换，并且在本地缓存10s。
2. `void tf::TransformListener::lookupTransform (const std::string &target_frame, const std::string &source_frame, const ros::Time &time, StampedTransform &transform) const` lookupTransform的API如此所示，在本例中监听turtle2->turtle1的坐标变换。
3. `rospy.Time(0)` 控制返回当前系统中可使用的最新的变换。

将下列代码加入刚刚撰写start_demo.launch中

```
<launch>
    ...
    <node pkg="learning_tf" type="turtle_tf_listener.py"
          name="listener" />
  </launch>
```

启动launch文件

```
$ roslaunch learning_tf start_demo.launch
```

成功后应该在窗口中具有两个turtle，turtle2跟踪turtle1. 至此，我们完成了tf跟踪设计。

任务四：自主任务

任务三中，我们采用python脚本编写了turtle2追踪turtle1节点：该节点实现 创建turtle2、 使turtle2追踪turtle1两个功能； 并编写launch脚本启动turtle2追踪节点 和 turtle仿真器启动&turtle1创建节点 和通过终端按键控制turtle1移动节点。

请同学们施展才华在任务三的基础上编写python脚本实现turtle3追踪turtle2节点： 该节点实现 创建turtle3、 使turtle3追踪turtle2两个功能。 并修改launch文件启动turtle3追踪turtle2节点，turtle2追踪turtle1节点， turtle仿真器启动&turtle1创建节点 ， 通过终端按键控制turtle1移动节点。

python代码放在 catkin_ws/src/learning_tf/scripts 文件夹中。launch 文件放在 catkin_ws/src/learning_tf/launch 文件夹中。并且将 catkin_ws/src/learning_tf 文件夹压缩为 learning_tf.zip 或者 learning_tf.tar.gz 压缩包（不要压缩成 rar）。

请在 10月16日晚 24 点之前将相应的实验报告（运行截图和感受）+ 文件压缩包 提交到网络学堂。

因为本课程只有两次需要提交的作业，所以请同学们在国庆期间好好自学本实验指导书内容，并且按时提交作业。以免影响课程成绩。