Chapter 3
TF

Wende Ke

Department of Mechanical and Energy Engineering

Southern University of Science and Technology

Introduction to tf

$ sudo apt-get install ros-noetic-ros-tutorials ros-noetic-geometry-tutorials ros-noetic-rviz ros-noetic-rosbash ros-noetic-rqt-tf-tree	
$ roslaunch turtle_tf turtle_tf_demo.launch	Let's run the demo.
$ rosrun tf view_frames	*view_frames* creates a diagram of the frames being broadcast by tf over ROS.
$ evince frames.pdf	Draw a tree of how the frames are connected.
$ rosrun rqt_tf_tree rqt_tf_tree	Visualize the tree of frames being broadcast over ROS.
$ rosrun tf tf_echo turtle1 turtle2	The transform of the turtle2 frame with respect to turtle1 frame.
$ rosrun rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz	

https://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf

More links:

https://wiki.ros.org/rqt_tf_tree

Writing a tf broadcaster (C++)

```
$ cd catkin_ws/src
```

```
$ catkin_create_pkg learning_tf tf roscpp rospy turtlesim
```

```
$ cd catkin_ws/
```

```
$ catkin_make
```

```
$ source ./devel/setup.bash
```

```
$ roscd learning_tf
```

https://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28C%2B%2B%29

More links:

https://wiki.ros.org/turtlesim

Create a file called ***turtle_tf_broadcaster.cpp*** in /src and paste the following codes:

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <turtlesim/Pose.h>
std::string turtle_name;

void poseCallback(const turtlesim::PoseConstPtr& msg){
  static tf::TransformBroadcaster br;
  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);
  br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));
}
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;
};
```

Simply add these few lines to the bottom of your *CMakeLists.txt*:

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

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

```
$ source ./devel/setup.bash
```

Create a launch file called *start_demo.launch* and paste the following:

```
<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" />

 </launch>
```

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

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

If you run tf_echo for the transform between the world and turtle 2, you should not see a transform, because the second turtle is not there yet.

Create a file called ***turtle_tf_listener.cpp*** in /src and paste the following:

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.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<geometry_msgs::Twist>("turtle2/cmd_vel", 10);

  tf::TransformListener listener;

  ros::Rate rate(10.0);
```

Create a file called ***turtle_tf_listener.cpp*** in /src and paste the following:

```
  while (node.ok()){
   tf::StampedTransform transform;
   try{
    listener.lookupTransform("/turtle2", "/turtle1",
                             ros::Time(0), transform);
   }
   catch (tf::TransformException &ex) {
    ROS_ERROR("%s",ex.what());
    ros::Duration(1.0).sleep();
    continue;
   }

   geometry_msgs::Twist vel_msg;
   vel_msg.angular.z = 4.0 * atan2(transform.getOrigin().y(),
                                   transform.getOrigin().x());
   vel_msg.linear.x = 0.5 * sqrt(pow(transform.getOrigin().x(), 2) +
                                 pow(transform.getOrigin().y(), 2));
   turtle_vel.publish(vel_msg);

   rate.sleep();
  }
  return 0;
};
```

If you get an error "Lookup would require extrapolation into the past" while running, you can try this alternative code to call the listener:

```
try {
    listener.waitForTransform("/turtle2", "/turtle1", ros::Time(0), ros::Duration(10.0)
);
    listener.lookupTransform("/turtle2", "/turtle1", ros::Time(0), transform);
} catch (tf::TransformException ex) {
    ROS_ERROR("%s",ex.what());
}
```

https://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28C%2B%2B%29

Simply add these few lines to the bottom of your *CMakeLists.txt*:

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

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

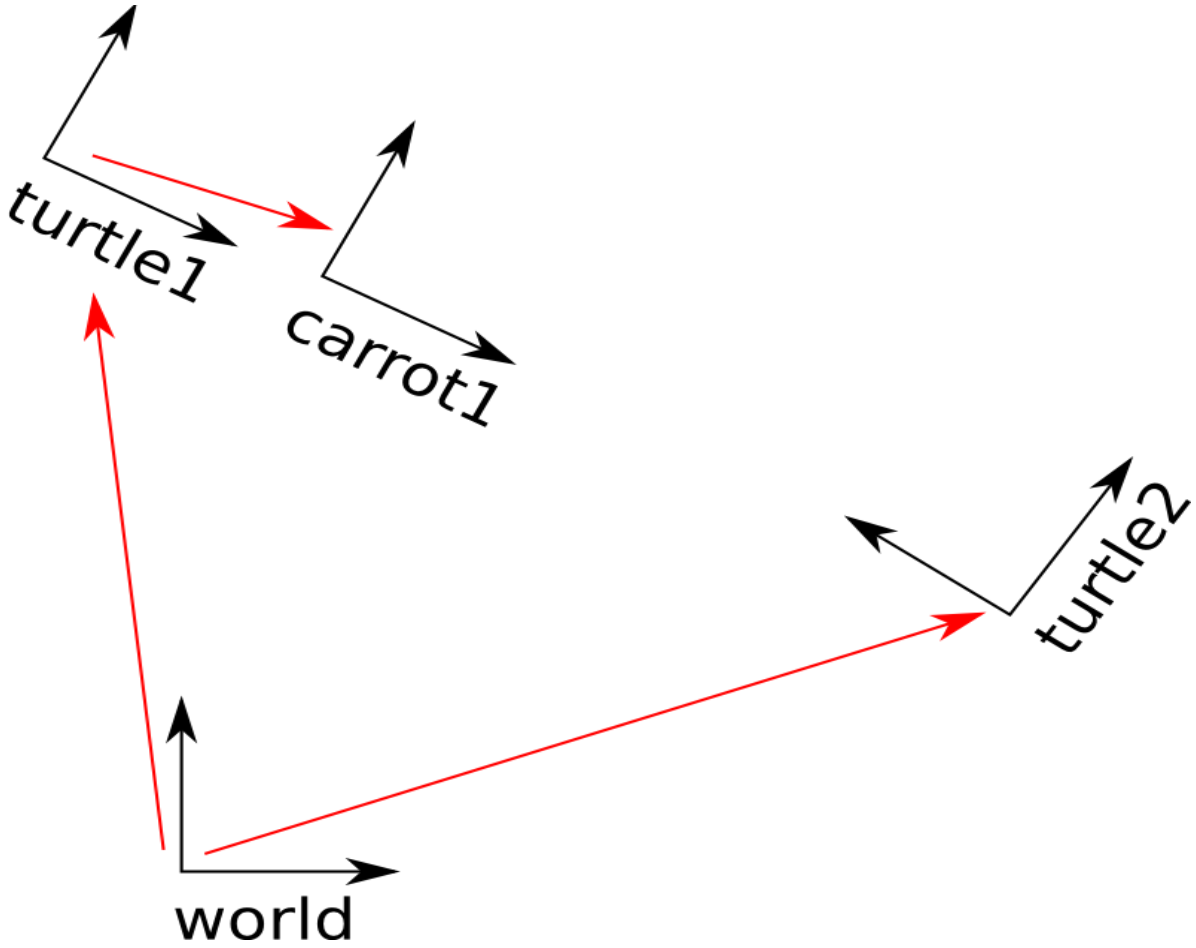
```
$ source ./devel/setup.bash
```

Open the launch file *start_demo.launch* and merge the node block below inside the <launch> block:

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

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

Adding a frame (C++)



https://wiki.ros.org/tf/Tutorials/Adding%20a%20frame%20%28C%2B%2B%29

Create a new file called src/*frame_tf_broadcaster.cpp* in /src and paste the following:

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>

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

  tf::TransformBroadcaster br;
  tf::Transform transform;

  ros::Rate rate(10.0);
  while (node.ok()){
    transform.setOrigin( tf::Vector3(0.0, 2.0, 0.0) );
    transform.setRotation( tf::Quaternion(0, 0, 0, 1) );
    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "turtle1", "carrot1"));
    rate.sleep();
  }
  return 0;
};
```

Simply add these few lines to the bottom of your *CMakeLists.txt*:

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

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

```
$ source ./devel/setup.bash
```

Open the launch file *start_demo.launch* and merge the node block below inside the <launch> block:

```
<launch>
   ...
   <node pkg="learning_tf" type="frame_tf_broadcaster"
         name="broadcaster_frame" />
 </launch>
```

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

Open the src/*turtle_tf_listener.cpp* file, and simple replace "*/turtle1*" with "*/carrot1*" in lines 26-27:

```
listener.lookupTransform("/turtle2", "/carrot1",
                         ros::Time(0), transform);
```

```
$ catkin_make
```

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

If you want to publish a moving frame you can change the broadcaster to change over time. Let's modify the file *frame_tf_broadcaster.cpp* with */carrot1* frame to change relative to */turtle1* over time:

```
transform.setOrigin( tf::Vector3(2.0*sin(ros::Time::now().toSec()),
2.0*cos(ros::Time::now().toSec()), 0.0) );
    transform.setRotation( tf::Quaternion(0, 0, 0, 1) );
```

```
$ catkin_make
```

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

Learning about tf and time (C++)

```
$ roscd learning_tf
```

Open the file src/*turtle_tf_listener.cpp*. Take a look at lines 25-27:

```
try{
    listener.lookupTransform("/turtle2", "/carrot1",
                             ros::Time(0), transform);
```

Let's make the second turtle follow the first turtle, and not the carrot. Change your code to the following:

```
try{
   listener.lookupTransform("/turtle2", "/turtle1",
                            ros::Time::now(), transform);
```

```
$ catkin_make
```

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

tf provides a nice tool that will wait until a transform becomes available. In *turtle_tf_listener.cpp*, let's look at what the code would look like:

```
try{
    ros::Time now = ros::Time::now();
    listener.waitForTransform("/turtle2", "/turtle1",
                              now, ros::Duration(3.0));
    listener.lookupTransform("/turtle2", "/turtle1",
                             now, transform);
```

```
$ catkin_make
```

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

https://wiki.ros.org/tf/Tutorials/tf%20and%20Time%20%28C%2B%2B%29