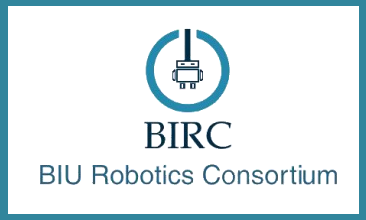
October 2016



ROS – Lecture 6

ROS tf system

Get robot's location on map

Lecturer: Roi Yehoshua

roiyeho@gmail.com

What is tf?

- A robotic system typically has many coordinate frames that change over **time**, such as a world frame, base frame, gripper frame, head frame, etc.
- tf is a transformation system that allows making computations in one frame and then transforming them to another at any desired point in time
- tf allows you to ask questions like:
  - What is the current pose of the base frame of the robot in the map frame?
  - What is the pose of the object in my gripper relative to my base?
  - Where was the head frame relative to the world frame, 5 seconds ago?

Pros of tf

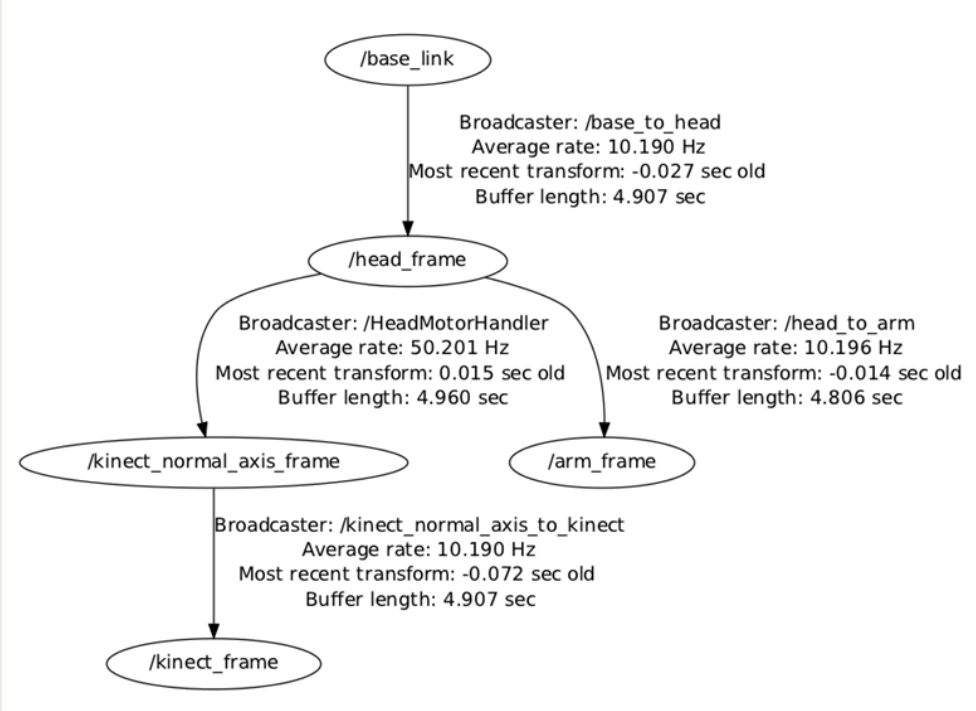
- Distributed system – no single point of failure
- No data loss when transforming multiple times
- No computational cost of intermediate data transformations between coordinate frames
- The user does not need to worry about which frame their data started
- Information about past locations is also stored and accessible (after local recording was started)

tf Nodes

- There are two types of tf nodes:
  - **Publishers** – publish transforms between coordinate frames on /tf
  - **Listeners** – listen to /tf and cache all data heard up to cache limit

Transform Tree

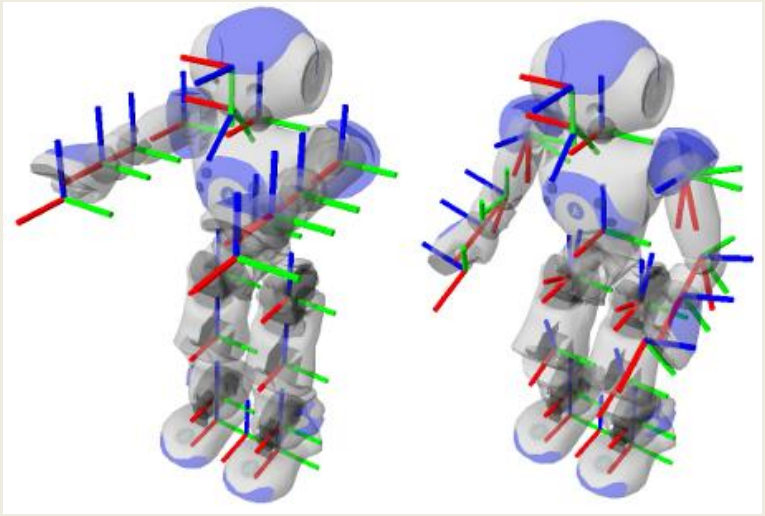
- TF builds a tree of transforms between frames

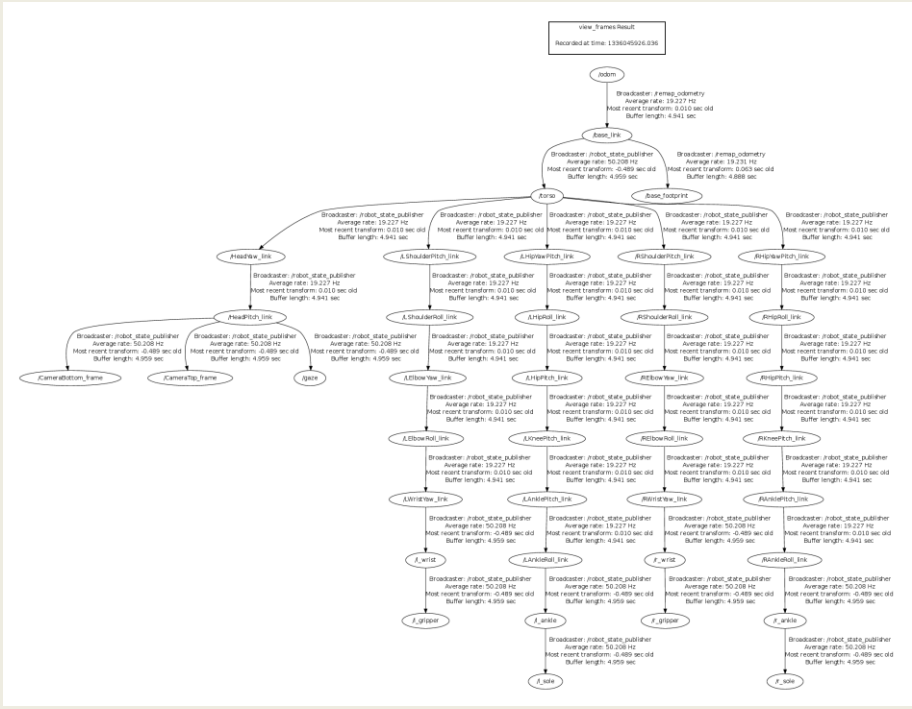


- Can support multiple disconnected trees
- Transforms only work within the same tree

Transform Tree Example

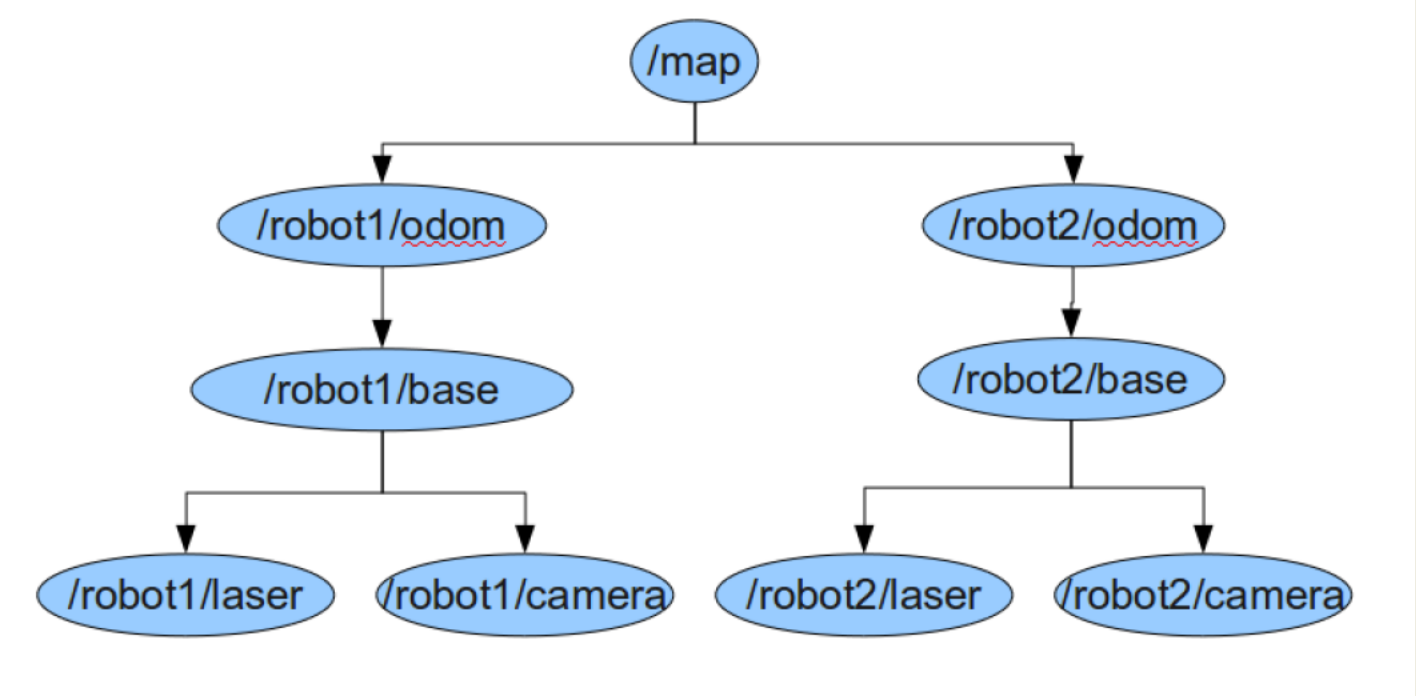
- Nao's TF tree



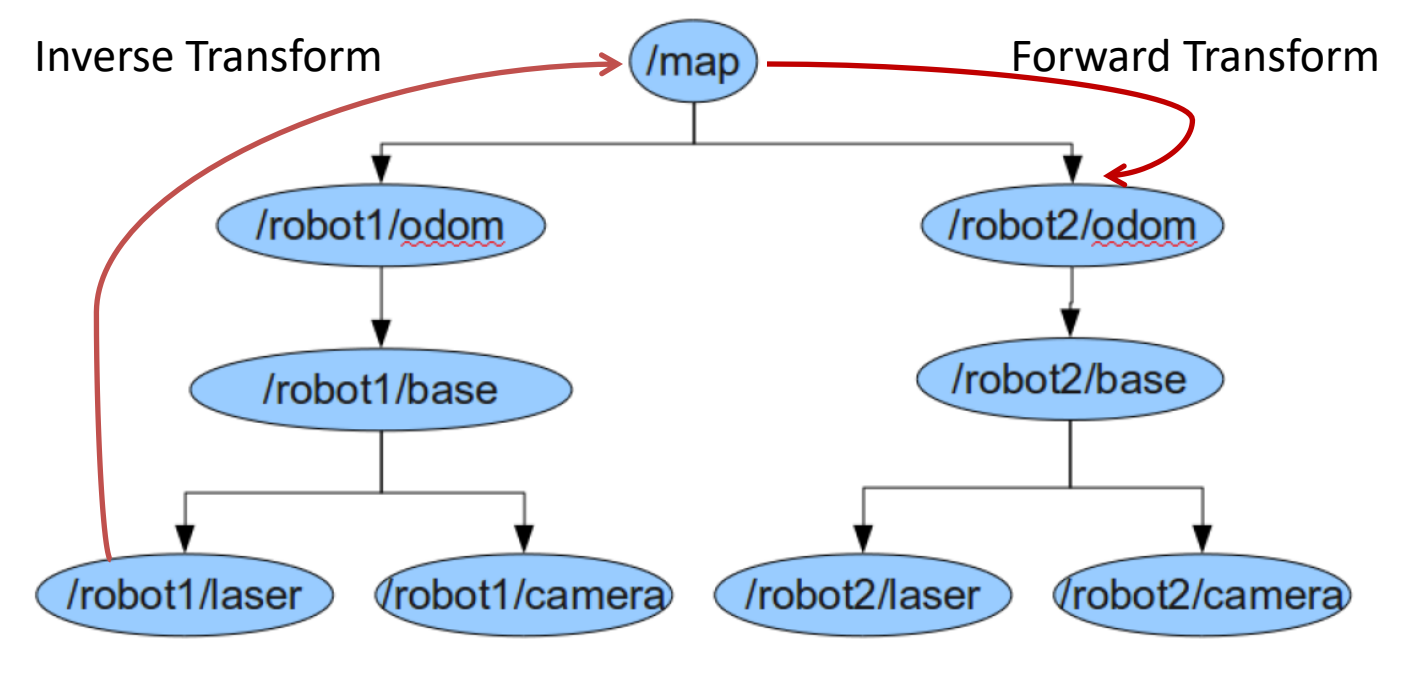


How To Use the TF Tree

- Given the following TF tree, let's say we want robot2 to navigate based on the laser data of robot1



How To Use the TF Tree



tf Demo

- Launch the turtle_tf_demo by typing:

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

- In another terminal run the turtle_tf_listener

```
$ rosrun turtle_tf turtle_tf_listener
```

- Now you should see a window with two turtles where one follows the other
- You can drive the center turtle around in the turtlesim using the keyboard arrow keys

```
$ rosrun turtlesim turtle_teleop_key
```

tf Demo

tf Demo

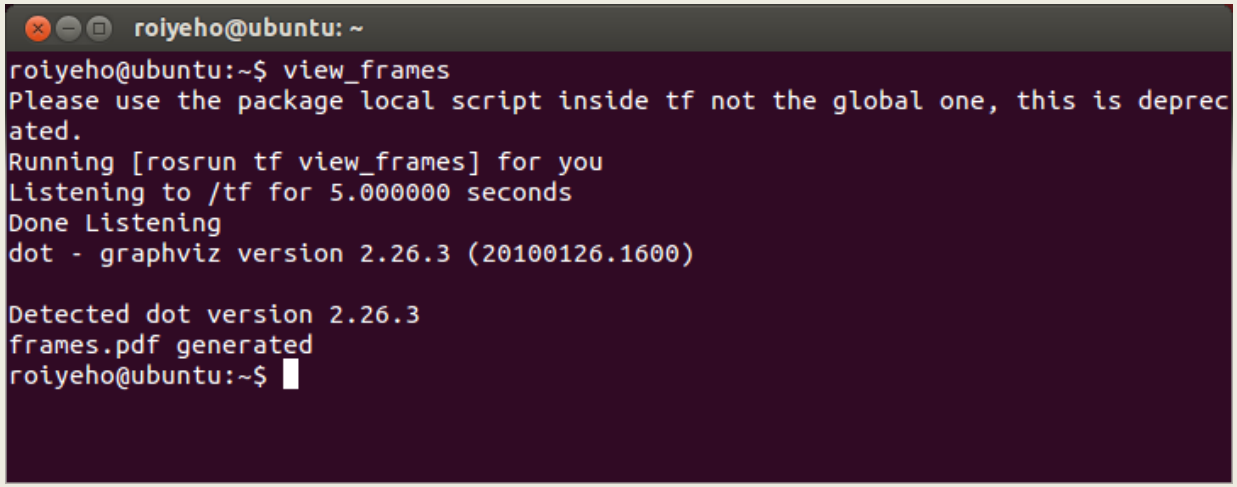
- This demo is using the tf library to create three coordinate frames: a world frame, a turtle1 frame, and a turtle2 frame.
- A **tf broadcaster** publishes the turtle coordinate frames and a **tf listener** computes the distance between the turtle frames to follow the other turtle

tf Command-line Tools

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

view_frames

- view_frames creates a diagram of the frames being broadcast by tf over ROS

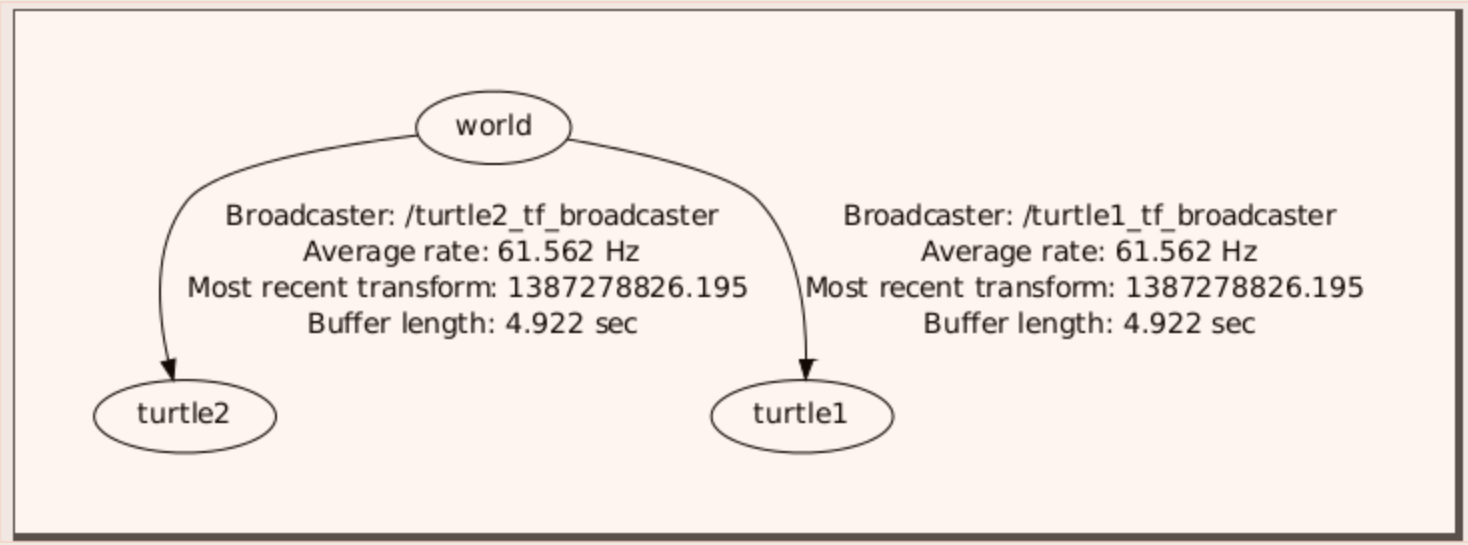


- To view the tree write:

```
$ evince frames.pdf
```

view_frames

- The TF tree:



tf_echo

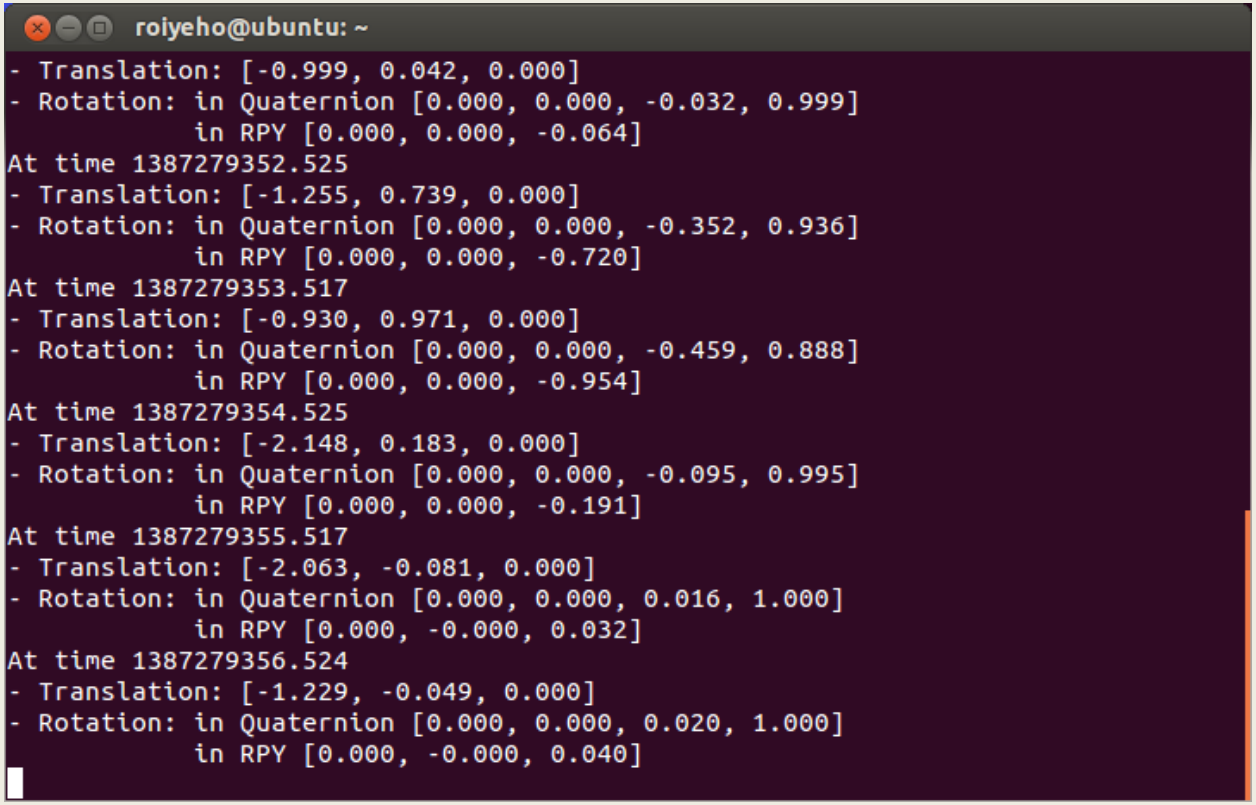
- tf_echo reports the transform between any two frames broadcast over ROS
- Usage:

```
$ rosrun tf tf_echo [reference_frame] [target_frame]
```

- Let's look at the transform of the turtle2 frame with respect to turtle1 frame which is equivalent to: $T_{turtle1\_turtle2} = T_{turtle1\_world} * T_{world\_turtle2}$

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

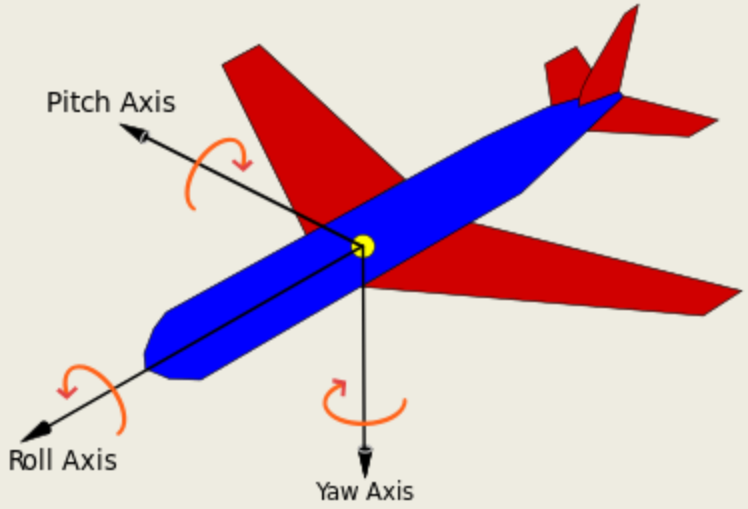
tf_echo

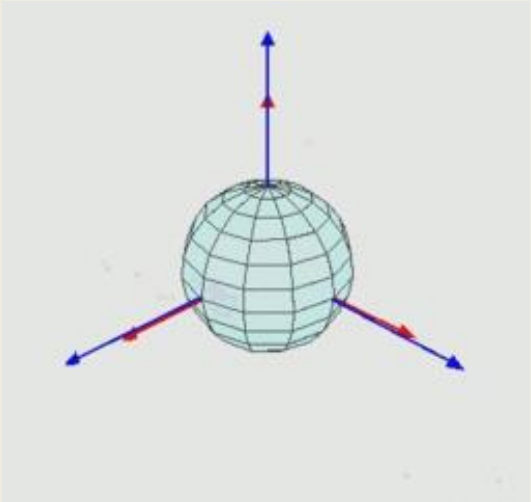


- As you drive your turtle around you will see the transform change as the two turtles move relative to each other

Rotation Representation

- There are many ways to represent rotations:
  - Euler angles yaw, pitch, and roll about Z, Y, X axes respectively
  - Rotation matrix
  - Quaternions





Quaternions

- In mathematics, quaternions are a number system that extends the complex numbers
- The fundamental formula for quaternion multiplication (Hamilton, 1843):

$$i^2 = j^2 = k^2 = ijk = -1$$

- Quaternions find uses in both theoretical and applied mathematics, in particular for calculations involving 3D rotations such as in computers graphics and computer vision

Quaternions and Spatial Rotation

- Any rotation in 3D can be represented as a combination of a vector u (the Euler axis) and a scalar θ (the rotation angle)
- A rotation with an angle of rotation θ around the axis defined by the unit vector

$$\vec{u} = (u_x, u_y, u_z) = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$$

is represented by

$$\mathbf{q} = e^{\frac{1}{2}\theta(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\tfrac{1}{2}\theta + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\tfrac{1}{2}\theta$$

Quaternions and Spatial Rotation

- Quaternions give a simple way to encode this axis–angle representation in 4 numbers
- Can apply the corresponding rotation to a position vector using a simple formula
  - http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
- Advantages of using quaternions:
  - Nonsingular representation
    - there are 24 different possibilities to specify Euler angles
  - More compact (and faster) than matrices.

tf_monitor

- Print information about the current coordinate transform tree to console

```
$ rosrun tf tf_monitor
```

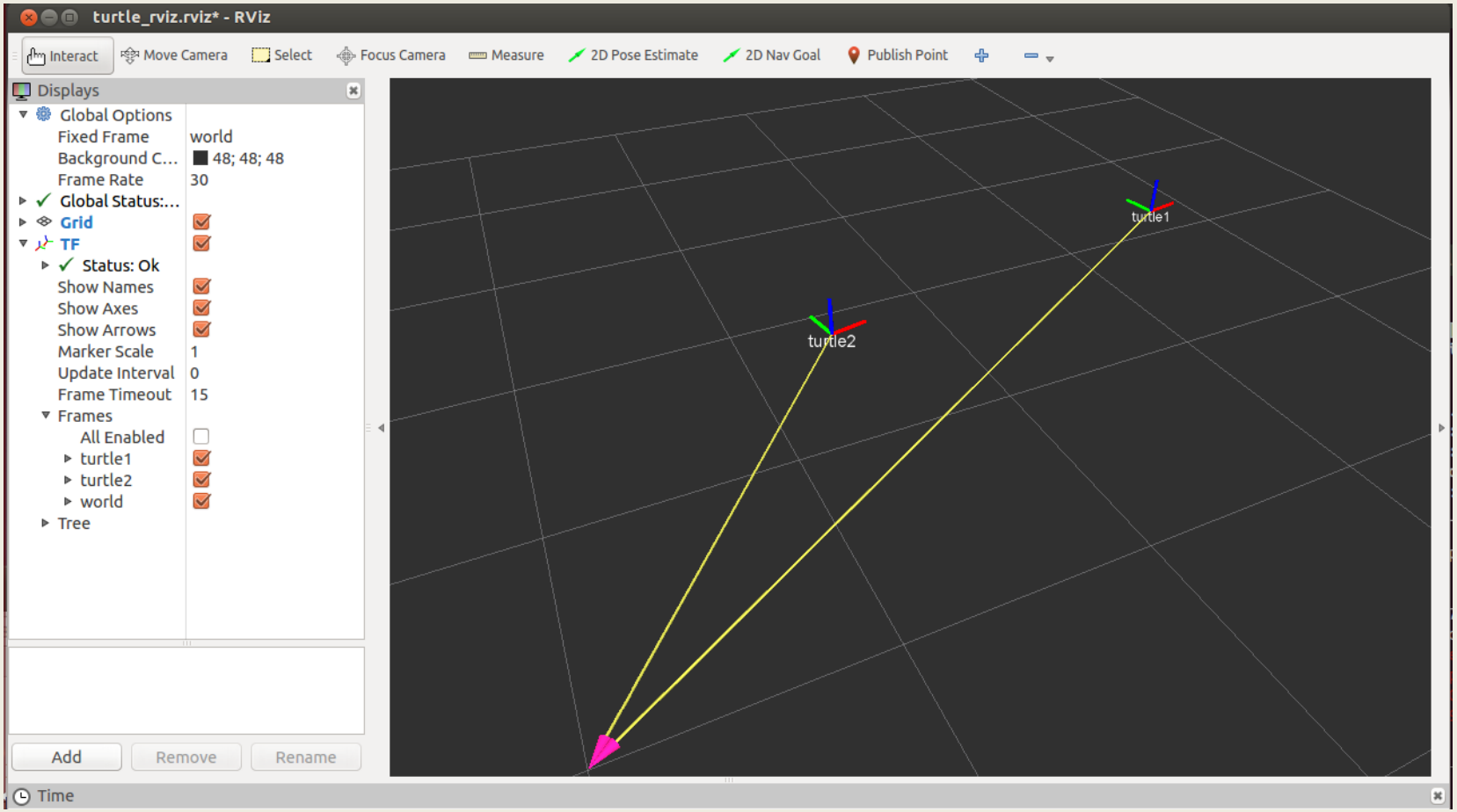
rviz and tf

- Let's look at our turtle frames using rviz
- Start rviz with the turtle_tf configuration file using the -d option for rviz:

```
$ rosrun rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz
```

- On the left side bar you will see the frames broadcast by tf
- Note that the fixed frame is /world
  - The fixed frame is assumed not to be moving over time
- As you drive the turtle around you will see the frames change in rviz

rviz and tf



Broadcasting Transforms

- A tf broadcaster sends out the relative pose of coordinate frames to the rest of the system
- A system can have many broadcasters, each provides information about a different part of the robot
- We will now write the code to reproduce the tf demo

Writing a tf broadcaster

- First create a new package called tf_demo that depends on tf, roscpp, rospy and turtlesim

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tf_demo tf roscpp rospy turtlesim
```

- Build the package by calling catkin_make
- Open the package in Eclipse and add a new source file called tf_broadcaster.cpp

tf_broadcaster.cpp (1)

```
#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 quaternion;
    transform.setRotation(tf::createQuaternionFromYaw(msg->theta));
    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(),
"world", turtle_name));
}
```

tf_broadcaster.cpp (2)

```
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;
};
```

Sending Transforms

```
br.sendTransform(tf::StampedTransform(transform,
ros::Time::now(), "world", turtle_name));
```

- Sending a transform with a TransformBroadcaster requires 4 arguments:
  - The transform object
  - A timestamp, usually we can just stamp it with the current time, ros::Time::now()
  - The name of the parent frame of the link we're creating, in this case "world"
  - The name of the child frame of the link we're creating, in this case this is the name of the turtle itself

Running the Broadcaster

- Create tf_demo.launch in the /launch subfolder

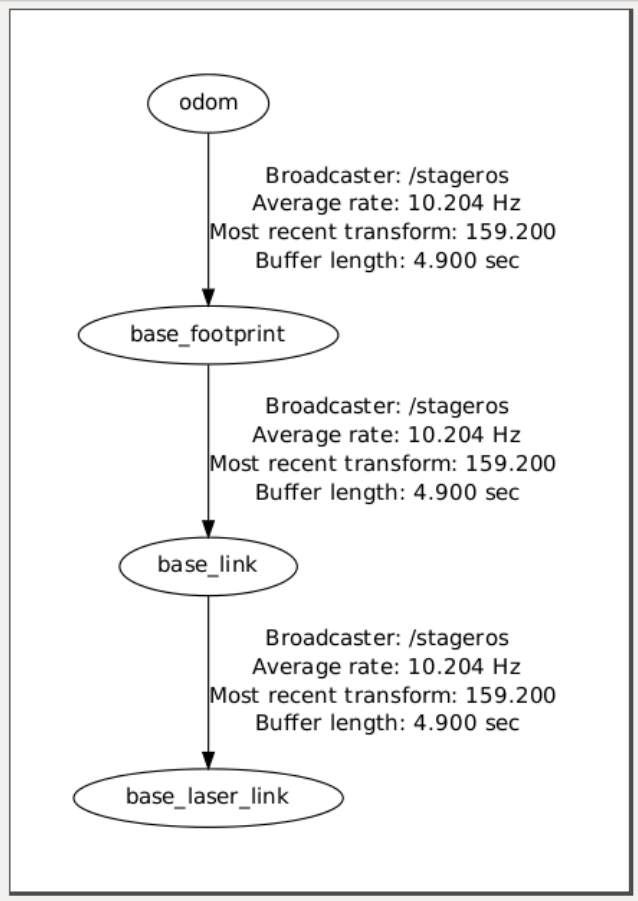
```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

  <!-- tf broadcaster node -->
  <node pkg="tf_demo" type="turtle_tf_broadcaster"
      args="/turtle1" name="turtle1_tf_broadcaster" />
 </launch>
```

- Run the launch file

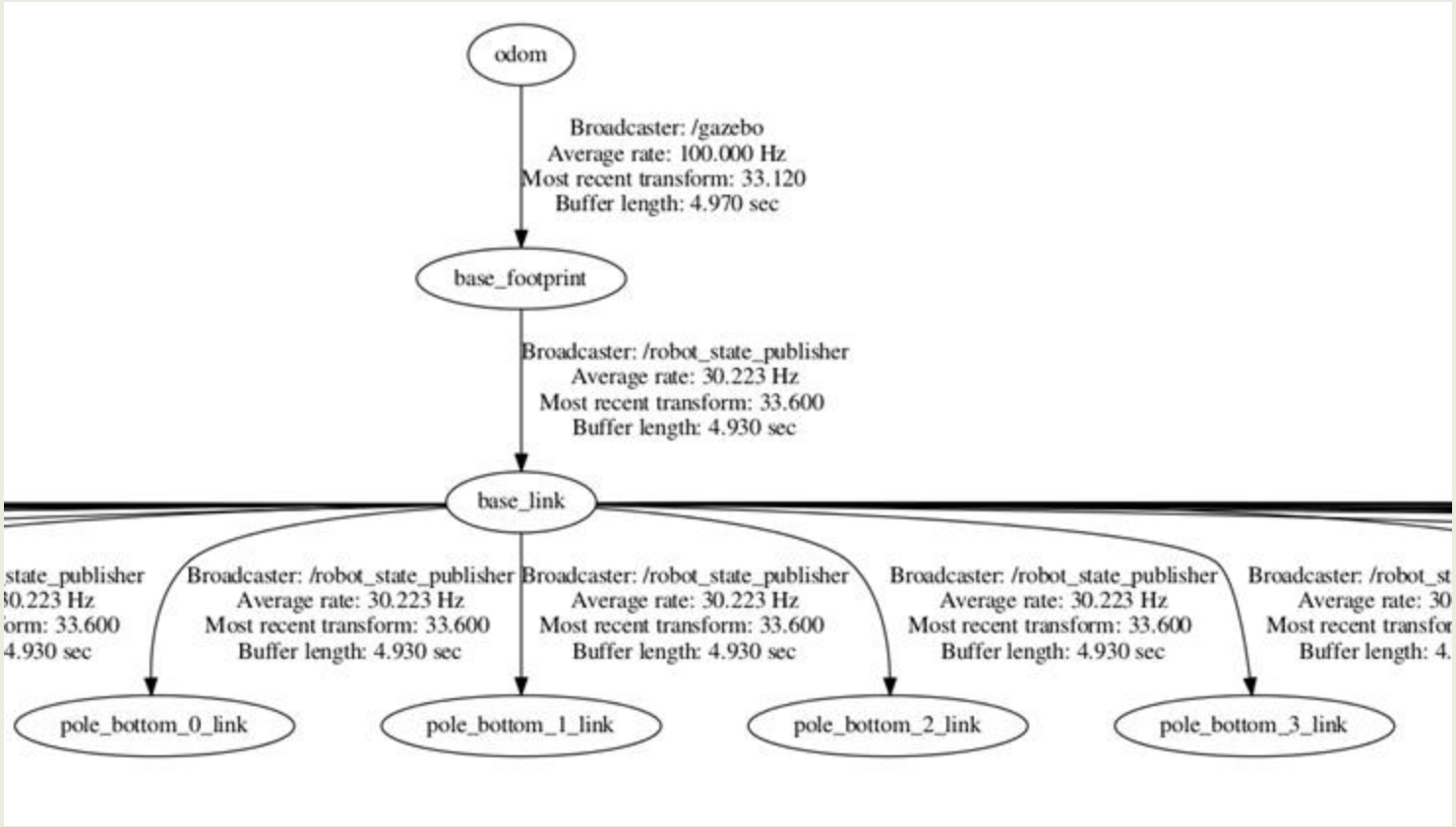
```
$ cd ~/catkin_ws/src
$ roslaunch tf_demo tf_demo.launch
```

Typical TF Frames



- **odom** – the self consistent coordinate frame using the odometry measurements only
- **base_footprint** – the base of the robot at zero height above the ground
- **base_link** – the base link of the robot, placed at the rotational center of the robot
- **base_laser_link** – the location of the laser sensor

Turtlebot TF Frames



Ex. 6

- Write functions that translate the robot's current position in the world (x, y) to the cell in the grid map (i, j) in which the robot is located and vice versa
  - that take into account the map resolution
- Print the initial cell that the robot is located at the start