Com S 476/576 Lab 3

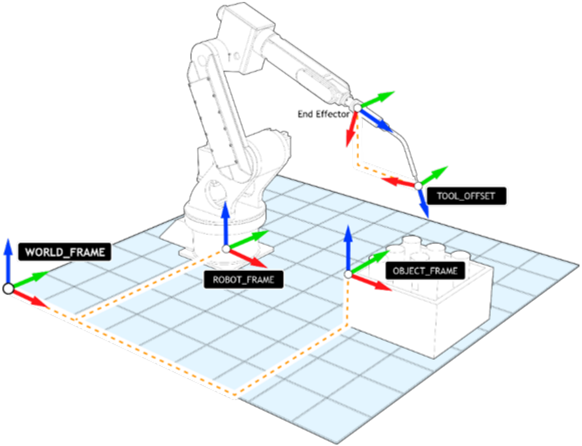
Coordinates and Transformation (ROS)

Yuechuan Xue
yuechuan@iastate.edu

Com S 476/576, Spring 2021

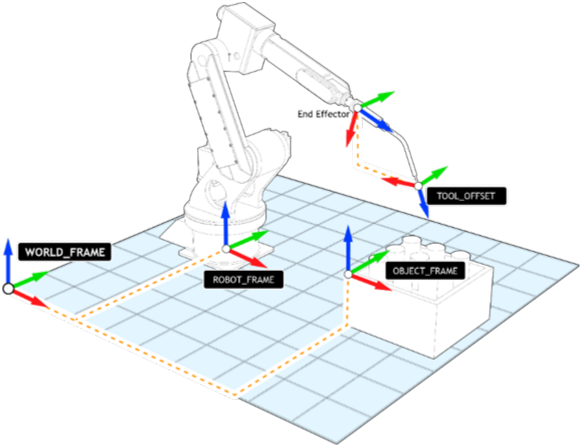
Frames of Reference

- Robots sense objects in the physical world
    - need positions and orientations over time
- A frame of reference provides a physical representation for observations
    - provides a coordinate system
    - may move with an observer or sensor



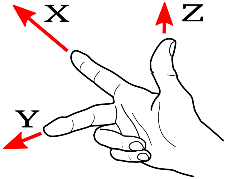
Cartesian Coordinates

- Three numbers describe any point in 3-dimensional space
- The frame of reference defines an origin for those coordinates
- The “right hand rule” determines their signs



Right Hand Rule

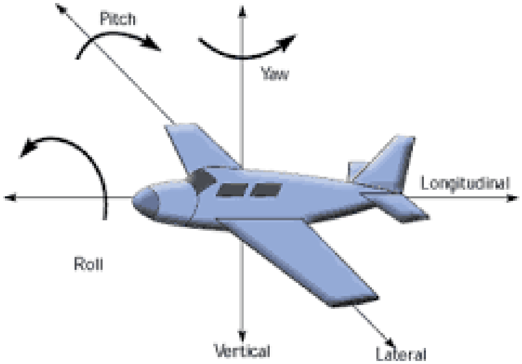
- There are several variants
- “Right hand rule” is the one used in robotics.



Poses

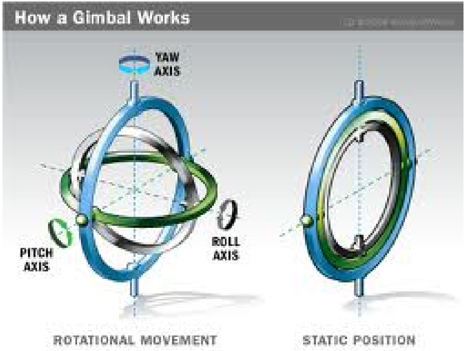
- A **point** has no orientation
- A **rigid body** has a pose with both position and orientation
- **orientation** uses several representations
  - Euler angles: roll, pitch and yaw
  - Rotation matrix
  - Quaternion

Yaw, Pitch, and Roll



Problems with Euler Angles

- The order of the angles affects the rotation
- Some sequences do not work for some angles ("gimbal lock")
- YouTube video on gimbal lock (https://www.youtube.com/watch?v=zc8b2Jo7mno)



Other Orientation Representations

- Rotation matrix
  - 3×3 matrix
  - No gimbal lock
- Quaternion
  - Four numbers: $[qw, qx, qy, qz]$
  - No gimbal lock

ROS Conventions

Described in REP-0103, key concepts:

- All angles are in radians
- All coordinate frames are right-handed
- Standard axis
  - x forward
  - y left
  - z up
- Orientations are quaternions

ROS Message Header

- Many ROS messages have a **header**
- `std_msgs/Header` fields:
  - `time stamp`
  - `string frame_id`
- These fields describe the time and frame of reference for the message data

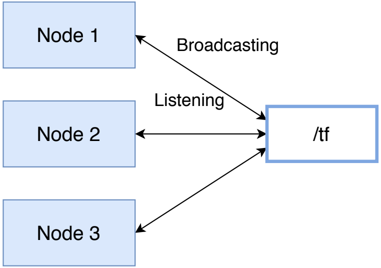
ROS Frames of Reference

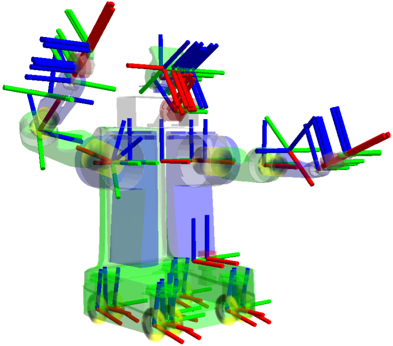
- Robots have many coordinate frames
- Each frame identifier is a string
  - map
  - odom
  - base_footprint
  - base_link
  - end_effector
- Frames for specific robots may use a prefix
  - `robot1/base_link`
  - `robot2/base_link`
- REP-0105 specifies standard coordinate frames for mobile robots
  - `map` is fixed in the world
  - `odom` is the world as measured by odometry
  - `base_link` is rigidly attached to the robot base
- The graph of reference frames is a **tree**
  - not always natural, but efficient to process
  - `map` → `odom` → `base_link`

ROS Transformation Library

TF Transformation System

- Tool for keeping track of coordinate frames over time
- Maintains relationship between coordinate frames in a tree structure buffered in time
- Lets the user transform points, vectors, etc. between coordinate frames at desired time
- Implemented as publisher/subscriber model on the topics `/tf` and `/tf_static`

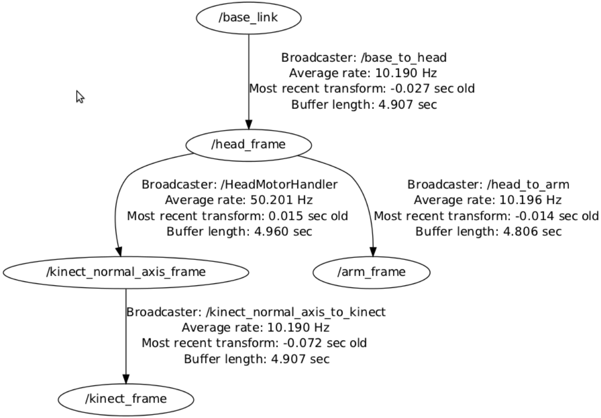




TF Transformation System

Transform Tree

- TF listeners use a buffer to listen to all broadcasted transforms
- Query for specific transforms from the transform tree



Broadcaster: /base_to_head
Average rate: 10.190 Hz
Most recent transform: -0.027 sec old
Buffer length: 4.907 sec

Broadcaster: /HeadMotorHandler
Average rate: 50.201 Hz
Most recent transform: 0.015 sec old
Buffer length: 4.960 sec

Broadcaster: /head_to_arm
Average rate: 10.196 Hz
Most recent transform: -0.014 sec old
Buffer length: 4.806 sec

Broadcaster: /kinect_normal_axis_to_kinect
Average rate: 10.190 Hz
Most recent transform: -0.072 sec old
Buffer length: 4.907 sec

tf2_msgs/TFMessage.msg

```
geometry_msgs/TransformStamped[] transforms
    std_msgs/Header header
        uint32 seqtime stamp
        string frame_id
    string child_frame_id
    geometry_msgs/Transform transform
        geometry_msgs/Vector3 translation
        geometry_msgs/Quaternion rotation
```

TF Transformation System
Tools

Command Line

Print information about the current transform tree

```
> rosrun tf tf_monitor
```

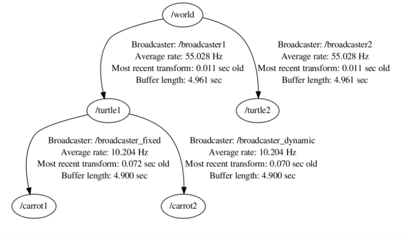
Print information about the transform between two frames

```
> rosrun tf tf_echo
      src_frame tar_frame
```

View Frames

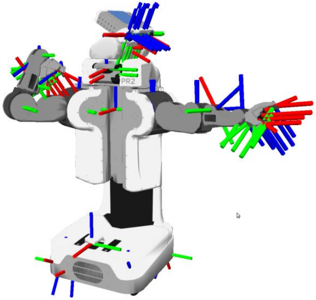
Creates a visual graph (PDF) of the transform tree

```
> rosrun tf view_frames
```


/world
Broadcaster: /broadcaster1
Average rate: 55.028 Hz
Most recent transform: 0.011 sec old
Buffer length: 4.961 sec
Broadcaster: /broadcaster2
Average rate: 55.028 Hz
Most recent transform: 0.011 sec old
Buffer length: 4.961 sec
/turtle1
/turtle2
Broadcaster: /broadcaster_fixed
Average rate: 10.204 Hz
Most recent transform: 0.072 sec old
Buffer length: 4.900 sec
Broadcaster: /broadcaster_dynamic
Average rate: 10.204 Hz
Most recent transform: 0.070 sec old
Buffer length: 4.900 sec
/carrot1
/carrot2

RViz

3D Visualization



TF2 Static Broadcaster Demo

```python
import rospy

# to get commandline arguments
import sys

# because of transformations
import tf

import tf2_ros
import geometry_msgs.msg

if __name__ == '__main__':
    rospy.init_node('my_static_tf2_broadcaster')
    broadcaster = tf2_ros.StaticTransformBroadcaster()
    static_transformStamped = geometry_msgs.msg.TransformStamped()

    ...
```

TF2 Static Broadcaster Demo

```python
if __name__ == '__main__':
    rospy.init_node('my_static_tf2_broadcaster')
    broadcaster = tf2_ros.StaticTransformBroadcaster()
    static_transformStamped = geometry_msgs.msg.TransformStamped()

    static_transformStamped.header.stamp = rospy.Time.now()
    static_transformStamped.header.frame_id = "world"
    static_transformStamped.child_frame_id = sys.argv[1]

    static_transformStamped.transform.translation.x = float(sys.argv[2])
    static_transformStamped.transform.translation.y = float(sys.argv[3])
    static_transformStamped.transform.translation.z = float(sys.argv[4])

    quat = tf.transformations.quaternion_from_euler(
               float(sys.argv[5]),float(sys.argv[6]),float(sys.argv[7]))
    static_transformStamped.transform.rotation.x = quat[0]
    static_transformStamped.transform.rotation.y = quat[1]
    static_transformStamped.transform.rotation.z = quat[2]
    static_transformStamped.transform.rotation.w = quat[3]

    broadcaster.sendTransform(static_transformStamped)
    rospy.spin()
```

TF2 Static Broadcaster

- First we need to mark the file as executable.

```
> chmod +x ./scripts/static_broadcaster.py
```

- Compile it with

```
> catkin build
```

- Start a roscore in a separate terminal

```
> source devel/setup.bash
> roscore
```

- Run with

```
> rosrun my_tf2_static_broadcaster static_broadcaster.py
                          mystaticframe 0 0 1 0 0 0
```

- Check result in another terminal with

```
> rostopic echo /tf_static
```

TF2 Static Broadcaster

- Employ ROS built-in TF2 static broadcaster

  ```
  > static_transform_publisher x y z qx qy qz qw
                       frame_id child_frame_id
  ```

  x/y/z offset in meters

- For use within roslaunch files for setting static transforms

  ```
  <launch>
      <node pkg="tf2_ros"
            type="static_transform_publisher"
            name="myframe_broadcaster"
            args="1 0 0 0 0 0 1 frame_id child_frame_id" />
  </launch>
  ```