Lab 03: ROS Tutorial 1

By Sean Lu, last modified on 09/25 2018.

The objective of this tutorial is to introduce to the ROS concept, nodes, publishers and subscribers, and also demonstrate some commonly used command line tools. Spatial transformation between each component is an important thing in your robot, luckily, tf helps us to do the job. We have assumed you have the basic coding ability to both C++ and Python. You should also understand homogeneous transformation matrix, if not, Prob. Yang's Robotics would be a good material for you. You should have installed ROS in your laptop, for those who use native Ubuntu please refer to the Hardware and Software Setup part.

Hardware and Software Setup

We have already installed ROS in the VirtualBox image we provided, for those who use native Ubuntu, please follow the step here.

Clone the repo and make the workspace

```
laptop $ cd ~/
laptop $ git clone https://github.com/Sensing-Intelligent-System/sis_lab_all_2020
# You will be asked to type your Github username and password
laptop $ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws
laptop $ catkin_make # This may take awhile
laptop $ source devel/setup.bash # Make ROS knows where your packages are, do this every time you open a new terminal
```

Download the bags we prepared here
temperature_pressure_bag
[Download in your VirtualBox]
Put the bag in **sis_lab_all_2020/03-ROS_tutorial_1/bag**

```
laptop $ cd ~/Downloads
laptop $ mkdir ~/sis_lab_all_2020/03-ROS_tutorial_1/bag && mv *.bag ~/sis_lab_all_2020/03-ROS_tutorial_1/bag/
```

Overview

Estimated Time to Finish: 2 hours

After completing this tutorial you should

- understand the basic ROS concept: nodes, publishers and subscribers
- understand how to use tf APIs
- be able to write a simple node to publish/ subscribe the specific topic

Topics and Activities

Topic/Activity 1 Command Line Tools

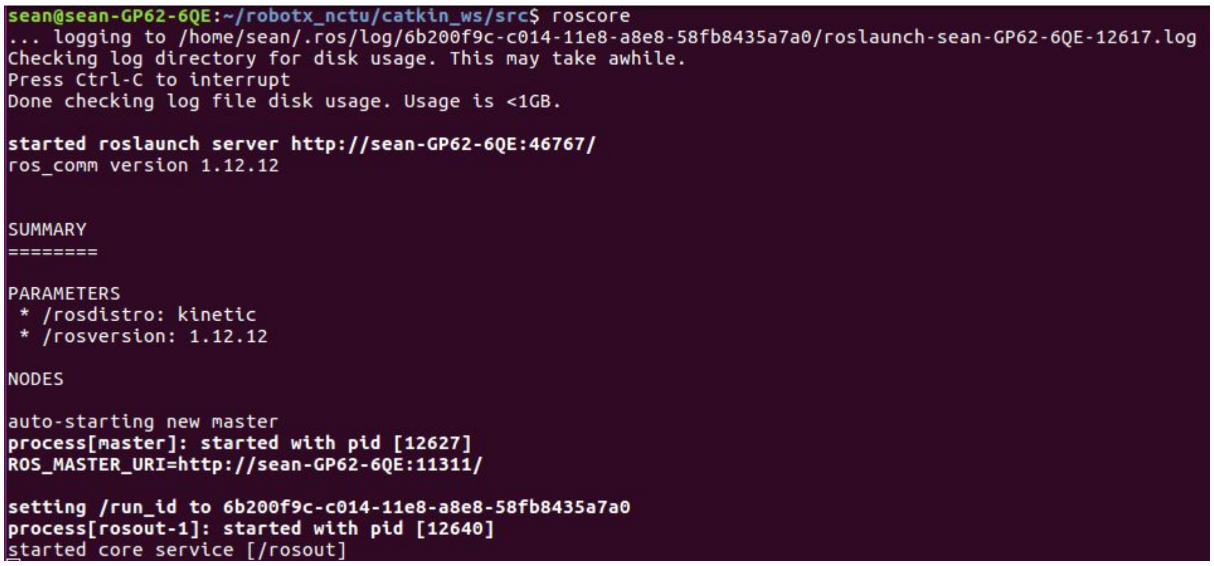
ROS provides many convenient command line tools, let's get familiar with them.

Topic 1.1 Initialize your ROS master

Open a new terminal, type

laptop $ roscore

You will see something like the following

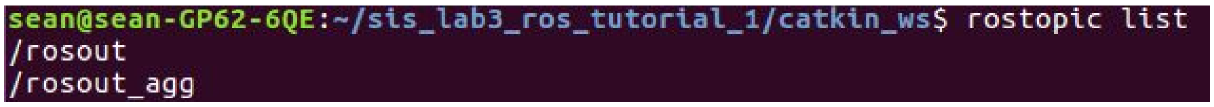


```
sean@sean-GP62-6QE:~/robotx_nctu/catkin_ws/src$ roscore
... logging to /home/sean/.ros/log/6b200f9c-c014-11e8-a8e8-58fb8435a7a0/roslaunch-sean-GP62-6QE-12617.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://sean-GP62-6QE:46767/
ros_comm version 1.12.12


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES

auto-starting new master
process[master]: started with pid [12627]
ROS_MASTER_URI=http://sean-GP62-6QE:11311/

setting /run_id to 6b200f9c-c014-11e8-a8e8-58fb8435a7a0
process[rosout-1]: started with pid [12640]
started core service [/rosout]
```

Open another terminal, type

laptop $ rostopic list

You will see there are two topics, /rosout and /rosout_agg



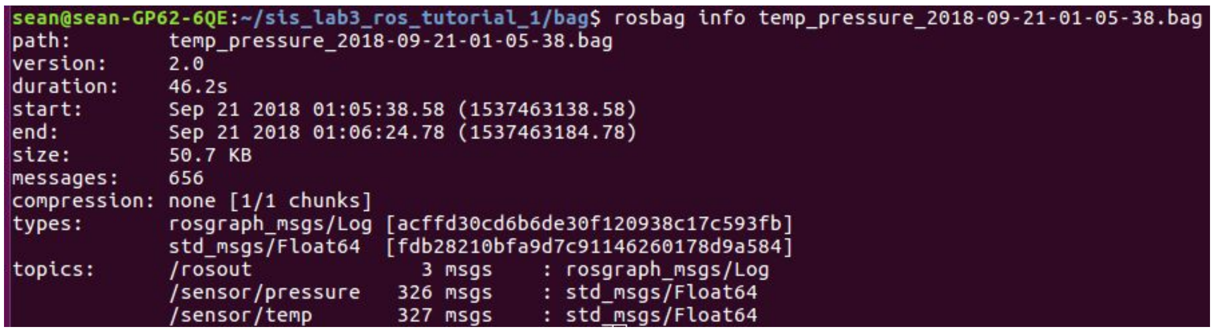
```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rostopic list
/rosout
/rosout_agg
```

Topic 1.2 Play the bag

Let's see what surprise inside the bag,

laptop $ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/bag

laptop $ rosbag info temp_pressure_2018-09-21-01-05-38.bag



```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/bag$ rosbag info temp_pressure_2018-09-21-01-05-38.bag
path:        temp_pressure_2018-09-21-01-05-38.bag
version:     2.0
duration:    46.2s
start:       Sep 21 2018 01:05:38.58 (1537463138.58)
end:         Sep 21 2018 01:06:24.78 (1537463184.78)
size:        50.7 KB
messages:    656
compression: none [1/1 chunks]
types:       rosgraph_msgs/Log [acffd30cd6b6de30f120938c17c593fb]
             std_msgs/Float64  [fdb28210bfa9d7c91146260178d9a584]
topics:      /rosout              3 msgs    : rosgraph_msgs/Log
             /sensor/pressure   326 msgs    : std_msgs/Float64
             /sensor/temp       327 msgs    : std_msgs/Float64
```

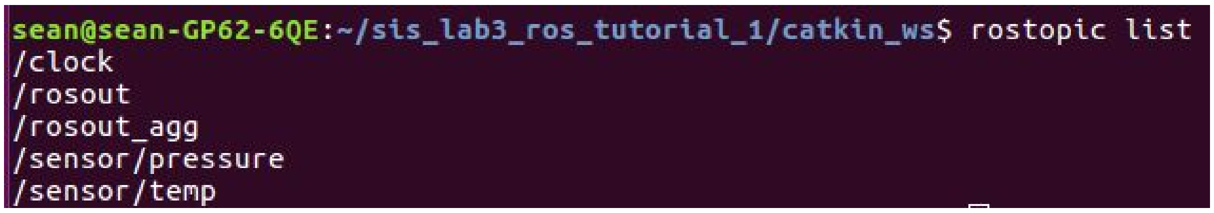
As you can see, the information include the duration, start and end time, bag size, total number of messages, the message type and topic name.

laptop $ rosbag play temp_pressure_2018-09-21-01-05-38.bag -l # -l means play in loops

Open another terminal, type

laptop $ rostopic list

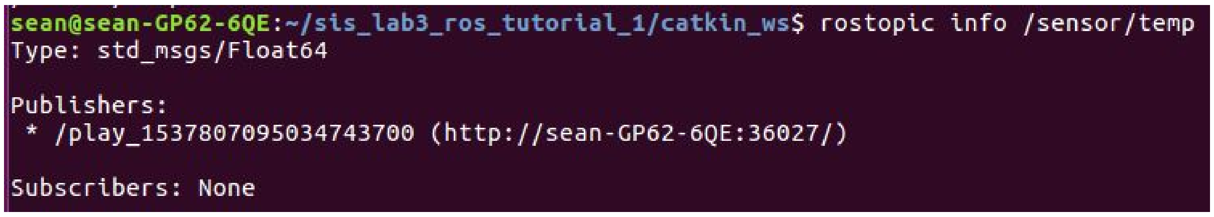
You will see two new topics as expected



```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rostopic list
/clock
/rosout
/rosout_agg
/sensor/pressure
/sensor/temp
```

Info helps us see who publishes and subscribes the topic

laptop $ rostopic info /sensor/temp

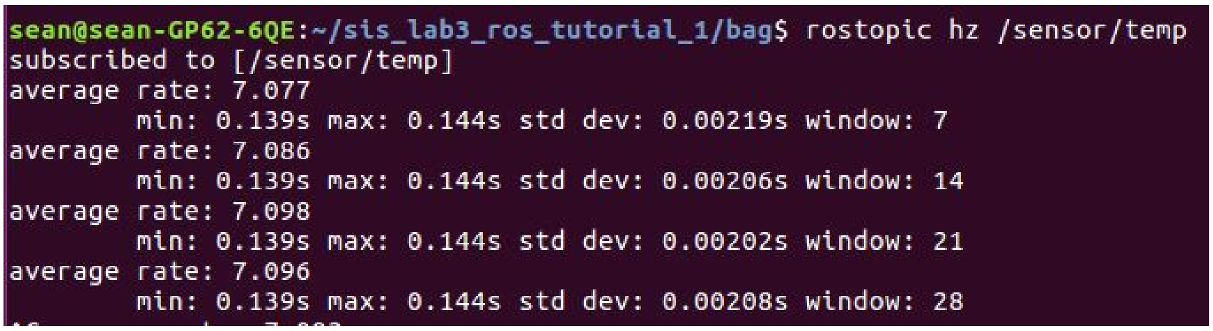


```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rostopic info /sensor/temp
Type: std_msgs/Float64

Publishers:
 * /play_1537807095034743700 (http://sean-GP62-6QE:36027/)

Subscribers: None
```

/sensor/temp is published by the node play, and no subscriber yet.

If we are interested in the rate of the topic:

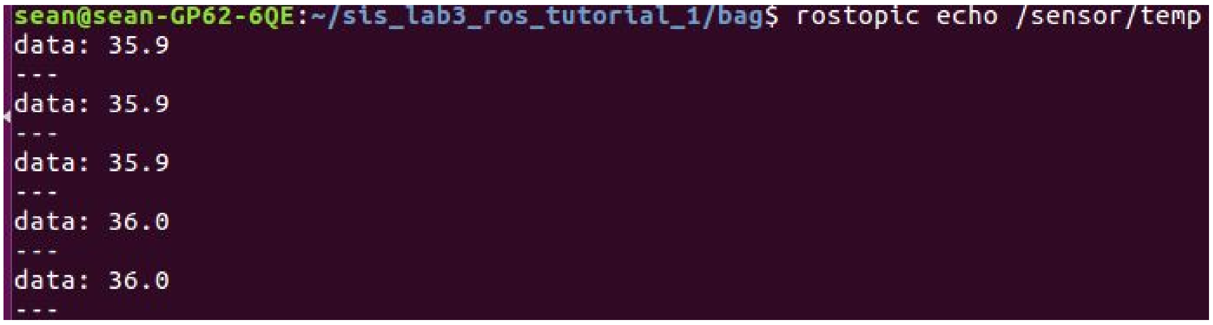
laptop $ rostopic hz /sensor/temp



```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/bag$ rostopic hz /sensor/temp
subscribed to [/sensor/temp]
average rate: 7.077
        min: 0.139s max: 0.144s std dev: 0.00219s window: 7
average rate: 7.086
        min: 0.139s max: 0.144s std dev: 0.00206s window: 14
average rate: 7.098
        min: 0.139s max: 0.144s std dev: 0.00202s window: 21
average rate: 7.096
        min: 0.139s max: 0.144s std dev: 0.00208s window: 28
```

The terminal keeps printing data until you press Ctrl+C.
Echo prints the data of the topic:

laptop $ rostopic echo /sensor/temp

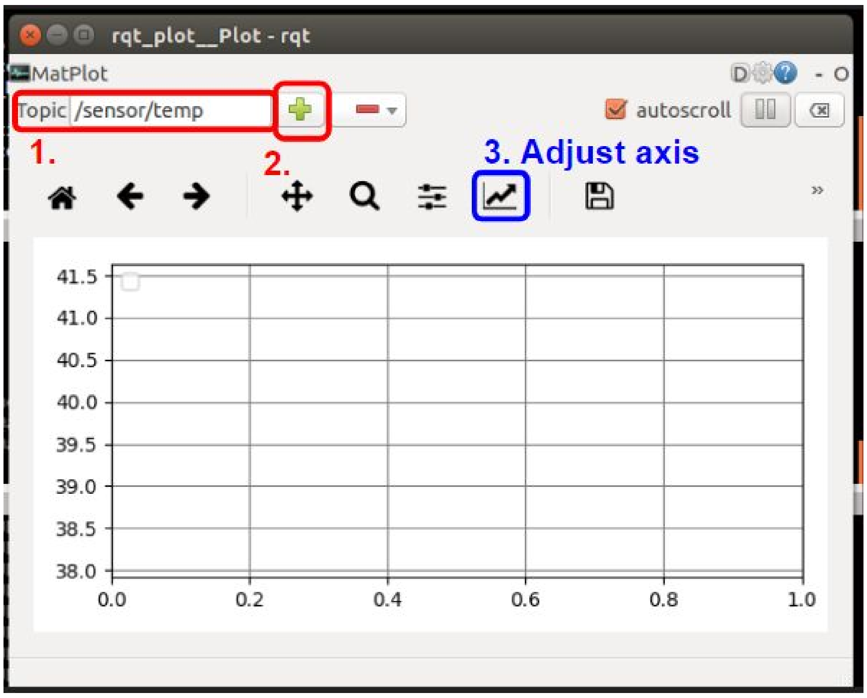


```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/bag$ rostopic echo /sensor/temp
data: 35.9
---
data: 35.9
---
data: 35.9
---
data: 36.0
---
data: 36.0
---
```

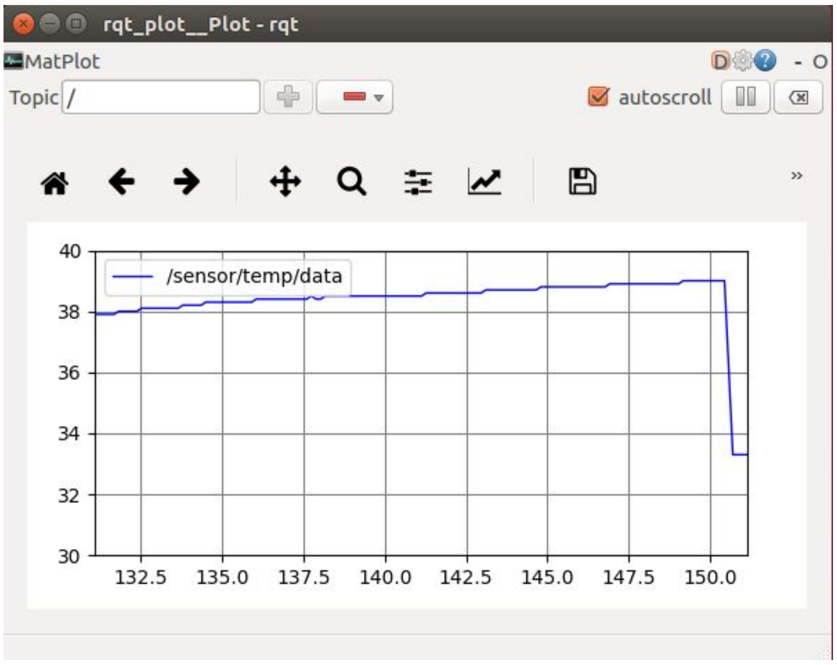
Press Ctrl+C to stop.
We can also visualize the temperature-time curve,

laptop $ rqt_plot

A window with name rqt_plot will show with empty canvas, in Topic field, add /sensor/temp



Then you will see the curve as following,



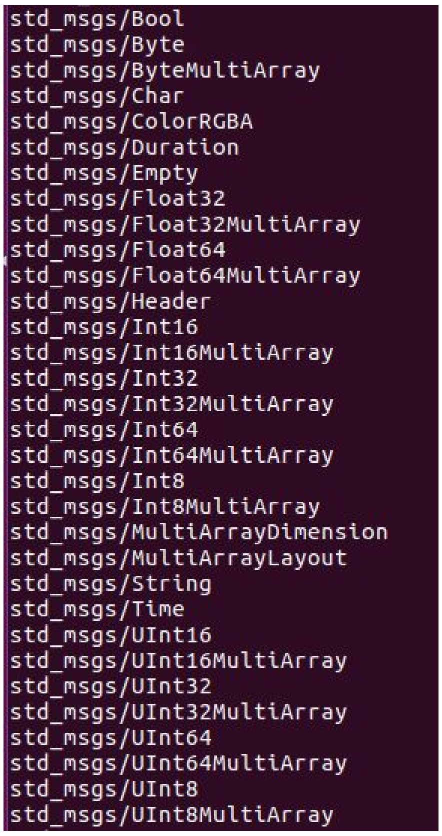
you can adjust the axes range in the blue box in above picture. Press ‘X’ to turn off the interface.

Topic 1.3 What messages and services I have?

There are many default messages and services in ROS, let’s take a look.

laptop $ rosmsg list

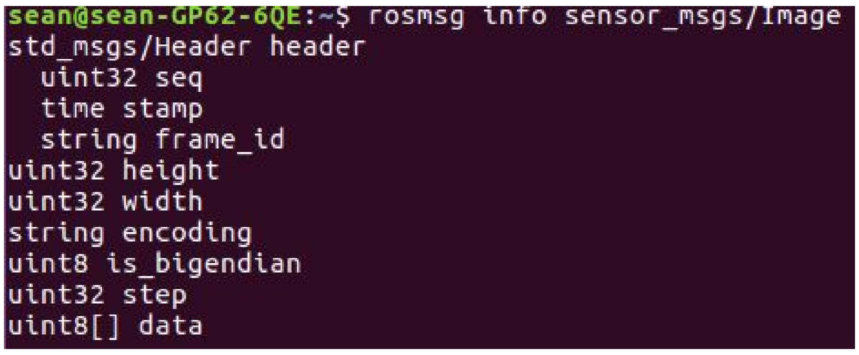
There will be several lines, for prefix ‘s’, you will see a family std_msgs:



```
std_msgs/Bool
std_msgs/Byte
std_msgs/ByteMultiArray
std_msgs/Char
std_msgs/ColorRGBA
std_msgs/Duration
std_msgs/Empty
std_msgs/Float32
std_msgs/Float32MultiArray
std_msgs/Float64
std_msgs/Float64MultiArray
std_msgs/Header
std_msgs/Int16
std_msgs/Int16MultiArray
std_msgs/Int32
std_msgs/Int32MultiArray
std_msgs/Int64
std_msgs/Int64MultiArray
std_msgs/Int8
std_msgs/Int8MultiArray
std_msgs/MultiArrayDimension
std_msgs/MultiArrayLayout
std_msgs/String
std_msgs/Time
std_msgs/UInt16
std_msgs/UInt16MultiArray
std_msgs/UInt32
std_msgs/UInt32MultiArray
std_msgs/UInt64
std_msgs/UInt64MultiArray
std_msgs/UInt8
std_msgs/UInt8MultiArray
```

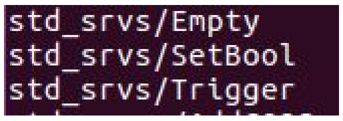
These are the most basically message types in ROS, you can find them in /opt/ros/kinetic/share/std_msgs/msg.

Other usually used message family include geometry_msgs, sensor_msgs and visualization_msgs.
If you want to see what information inside the message, for example, sensor_msgs/Image, you can type
laptop $ rosmsg info sensor_msgs/Image
The result is as following



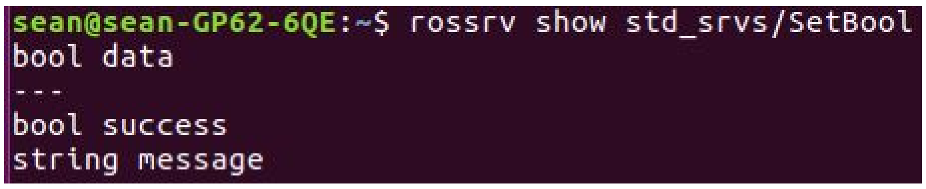
```
sean@sean-GP62-6QE:~$ rosmsg info sensor_msgs/Image
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

Just like rosmsg,
laptop $ rossrv list
You will see many available services in your system, a screenshot of the result is as below



```
std_srvs/Empty
std_srvs/SetBool
std_srvs/Trigger
```

So what SetBool do?
laptop $ rossrv show std_srvs/SetBool



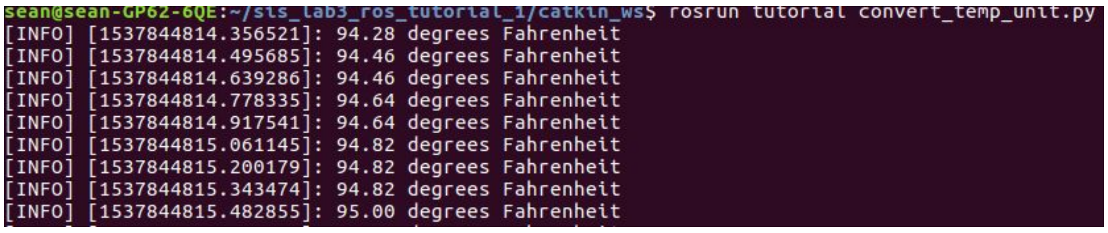
```
sean@sean-GP62-6QE:~$ rossrv show std_srvs/SetBool
bool data
---
bool success
string message
```

You call the service with ‘data’, the service will return a response to you whether if it is ‘success’ and a string ‘message’.

Checkpoint 1: Show TA your rqt_plot

Topic/Activity 2 Publisher/ Subscriber

Now we are going to run a ROS node, open a terminal
laptop $ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws && source devel/setup.bash
laptop $ rosrun tutorial convert_temp_unit.py
The result should be as following



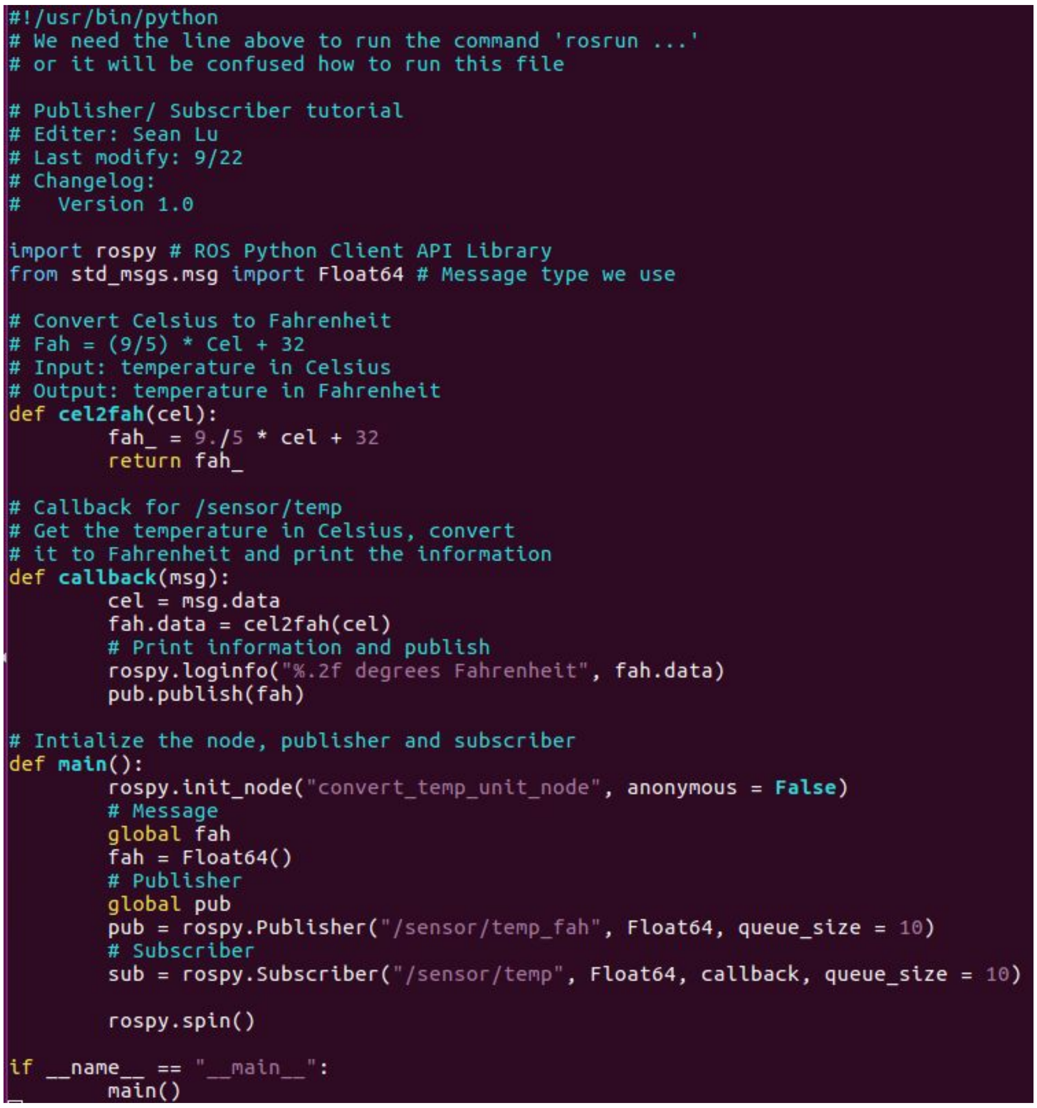
```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rosrun tutorial convert_temp_unit.py
[INFO] [1537844814.356521]: 94.28 degrees Fahrenheit
[INFO] [1537844814.495685]: 94.46 degrees Fahrenheit
[INFO] [1537844814.639286]: 94.46 degrees Fahrenheit
[INFO] [1537844814.778335]: 94.64 degrees Fahrenheit
[INFO] [1537844814.917541]: 94.64 degrees Fahrenheit
[INFO] [1537844815.061145]: 94.82 degrees Fahrenheit
[INFO] [1537844815.200179]: 94.82 degrees Fahrenheit
[INFO] [1537844815.343474]: 94.82 degrees Fahrenheit
[INFO] [1537844815.482855]: 95.00 degrees Fahrenheit
```

The number after INFO is the time now, you can get similar string in Python if you print time.time(). The string after colon is our information, what degrees Fahrenheit were at that time.
Let’s see what contents in the file, open a new terminal

laptop $ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws && source devel/setup.bash

laptop $ roscd tutorial/src

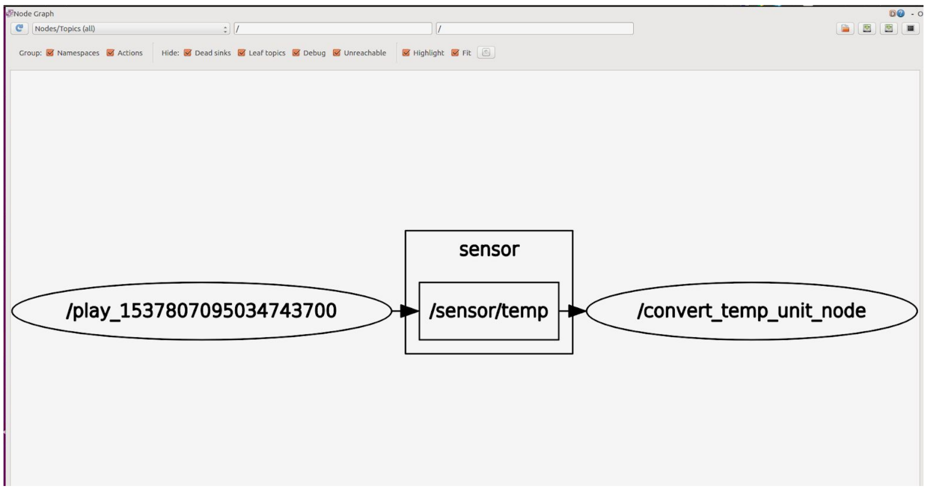
laptop $ vim convert_temp_unit.py (or $ code convert_temp_unit.py)



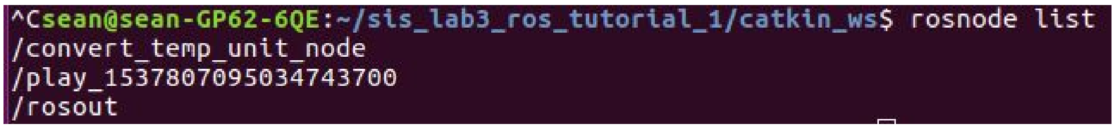
```python
#!/usr/bin/python
# We need the line above to run the command 'rosrun ...'
# or it will be confused how to run this file

# Publisher/ Subscriber tutorial
# Editer: Sean Lu
# Last modify: 9/22
# Changelog:
#   Version 1.0

import rospy # ROS Python Client API Library
from std_msgs.msg import Float64 # Message type we use

# Convert Celsius to Fahrenheit
# Fah = (9/5) * Cel + 32
# Input: temperature in Celsius
# Output: temperature in Fahrenheit
def cel2fah(cel):
        fah_ = 9./5 * cel + 32
        return fah_

# Callback for /sensor/temp
# Get the temperature in Celsius, convert
# it to Fahrenheit and print the information
def callback(msg):
        cel = msg.data
        fah.data = cel2fah(cel)
        # Print information and publish
        rospy.loginfo("%.2f degrees Fahrenheit", fah.data)
        pub.publish(fah)

# Intialize the node, publisher and subscriber
def main():
        rospy.init_node("convert_temp_unit_node", anonymous = False)
        # Message
        global fah
        fah = Float64()
        # Publisher
        global pub
        pub = rospy.Publisher("/sensor/temp_fah", Float64, queue_size = 10)
        # Subscriber
        sub = rospy.Subscriber("/sensor/temp", Float64, callback, queue_size = 10)

        rospy.spin()

if __name__ == "__main__":
        main()
```

In this node, we subscribe to /sensor/temp, convert its unit to Fahrenheit and then publish as topic /sensor/temp_fah, we also print the information as above.
What is our nodes/ topics flow now? Exit vim and type

laptop $ rqt_graph

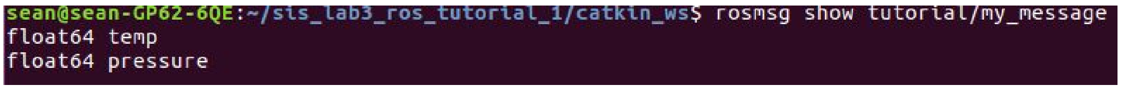

Node Graph
Nodes/Topics (all)
Group: Namespaces Actions
Hide: Dead sinks Leaf topics Debug Unreachable
Highlight Fit
sensor
/play_1537807095034743700
/sensor/temp
/convert_temp_unit_node

/play publishes /sensor/temp and /convert_temp_unit_node subscribes to it, the topics with no subscribers will not shown in rqt_graph.
With command
laptop $ rosnode list



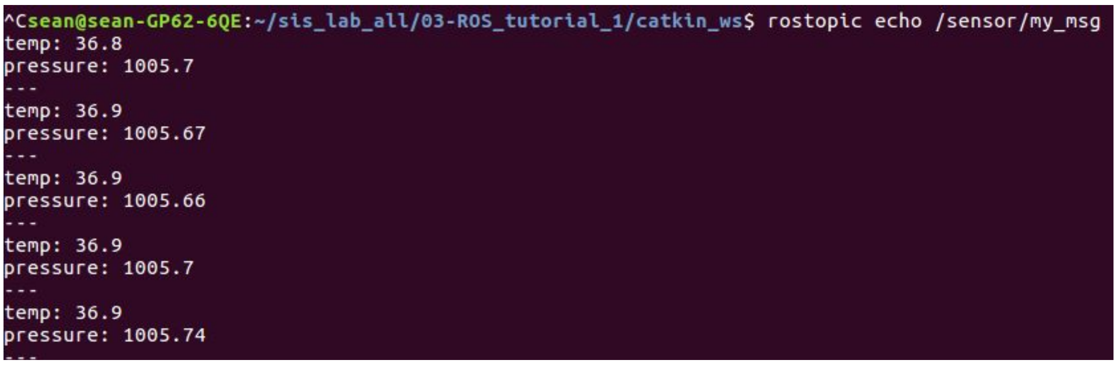
```
^Csean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rosnode list
/convert_temp_unit_node
/play_1537807095034743700
/rosout
```

we can list all active nodes just like "rostopic list"
How to create an user-defined message if we want to publish temperature and pressure simultaneously? Follow the step here to create the new message type, my_message, with two float64: temp and pressure.



```
sean@sean-GP62-6QE:~/sis_lab3_ros_tutorial_1/catkin_ws$ rosmsg show tutorial/my_message
float64 temp
float64 pressure
```

Next,
laptop $ rosrun tutorial test_my_message.py
In another terminal,
laptop $ rostopic echo /sensor/my_msg
You will see



```
^Csean@sean-GP62-6QE:~/sis_lab_all/03-ROS_tutorial_1/catkin_ws$ rostopic echo /sensor/my_msg
temp: 36.8
pressure: 1005.7
---
temp: 36.9
pressure: 1005.67
---
temp: 36.9
pressure: 1005.66
---
temp: 36.9
pressure: 1005.7
---
temp: 36.9
pressure: 1005.74
---
```

If you want to know what this node has done, see
sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws/src/tutorial/src/test_my_message.py
We subscribe to both /sensor/temp and /sensor/pressure, get the data and publish our new type message.

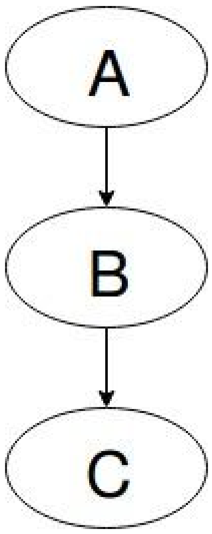
Checkpoint 2: Show TA the picture above

Topic/Activity 3 TF

Notation: T_A^B : *homogeneous transformation matrix from frame A to frame B*

Topic 3.1 Publish a transform

Given $T_A^B = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $T_B^C = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, how to build the transform tree like the figure below with the help of TF?

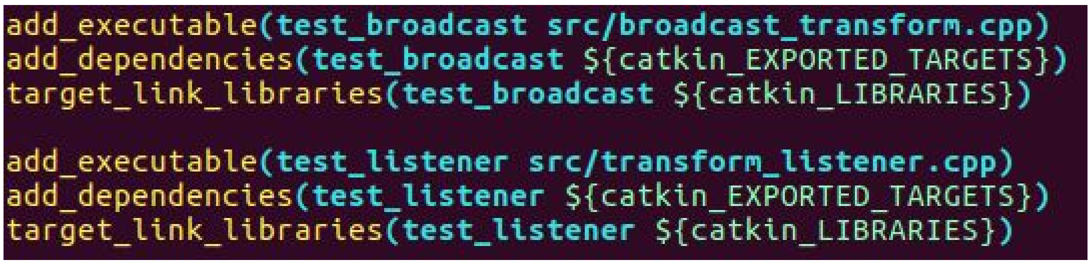


$$\begin{tikzcd} A \arrow[d] \\ B \arrow[d] \\ C \end{tikzcd}$$

In this tutorial, we use C++ to construct our node, while don't forget that 'rosrun' must be feed with an executable file, so you have to make your C++ source code be executable. In ROS Kinetic, we use **CMake** and **catkin** to lighten our burden. Take a look at **sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws/src/tutorial/CMakeLists.txt**

laptop **$ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws && source devel/setup.sh**
laptop **$ roscd tutorial**
laptop **$ vim CMakeLists.txt**

At Line 155, you will see



```
add_executable(test_broadcast src/broadcast_transform.cpp)
add_dependencies(test_broadcast ${catkin_EXPORTED_TARGETS})
target_link_libraries(test_broadcast ${catkin_LIBRARIES})

add_executable(test_listener src/transform_listener.cpp)
add_dependencies(test_listener ${catkin_EXPORTED_TARGETS})
target_link_libraries(test_listener ${catkin_LIBRARIES})
```

For more detail what the function of each tag, please refer to here. Don't edit the other part of CMakeLists.txt manually unless you know what you are doing. Every time you create a new C++ file, add these three tags to your CMakeLists.txt; or if you have modified your C++ file, back to **~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws**, and then

laptop **$ catkin_make # You can skip this step until Assignment 2**

Run the node, stop all your terminal unless roscore and in one terminal

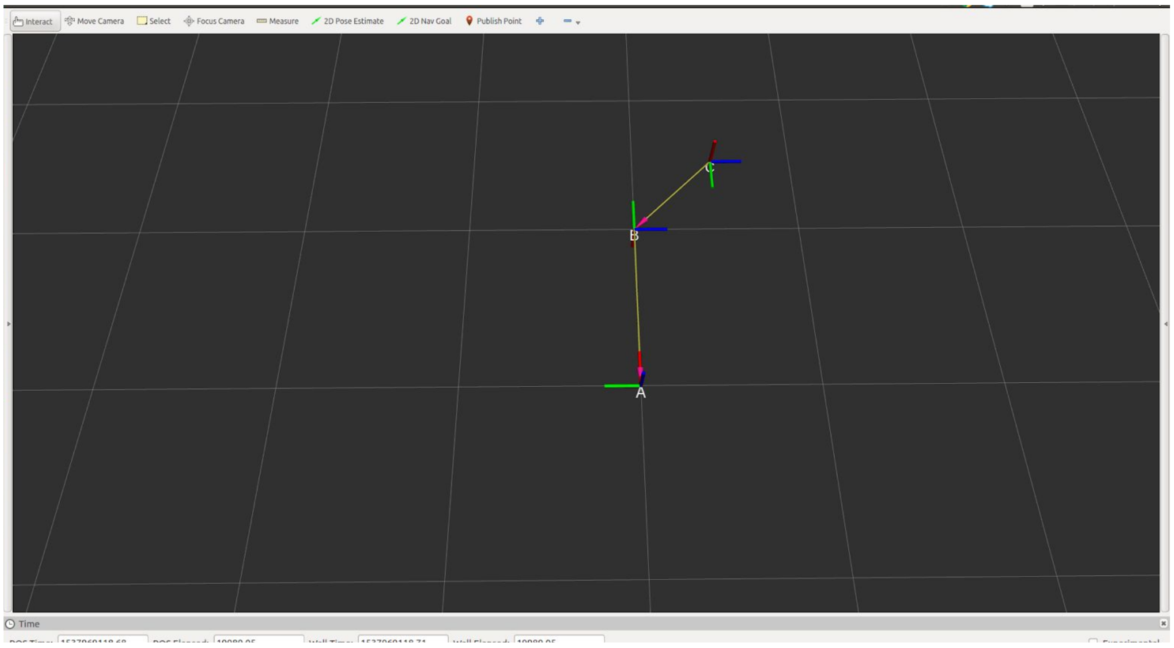
laptop **$ rosrun tutorial test_broadcast**

And in another terminal,

laptop **$ cd ~/sis_lab_all_2020/03-ROS_tutorial_1/rviz**

laptop **$ rviz -d tf.rviz**

You will see something like below

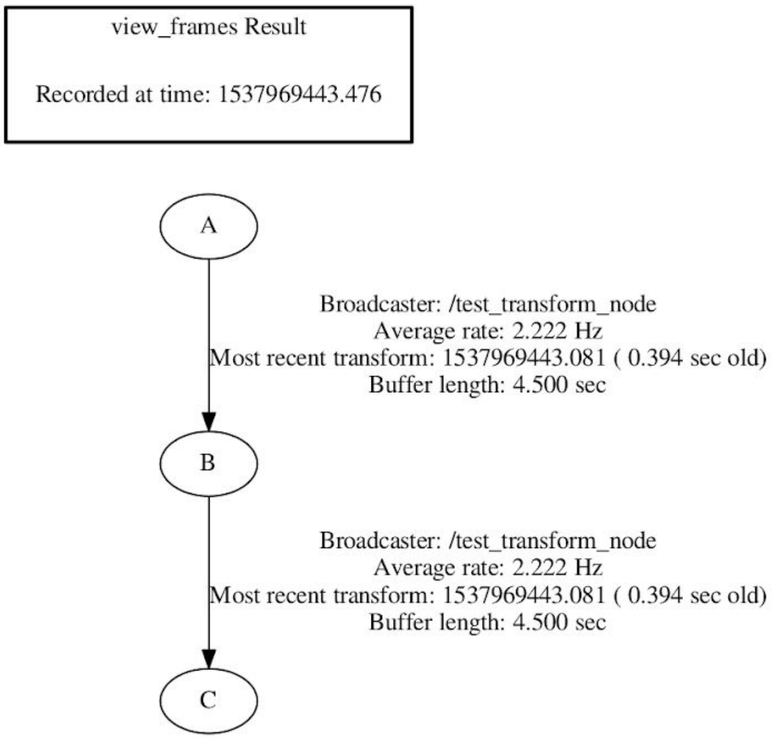


Let's see the transform tree, in another terminal

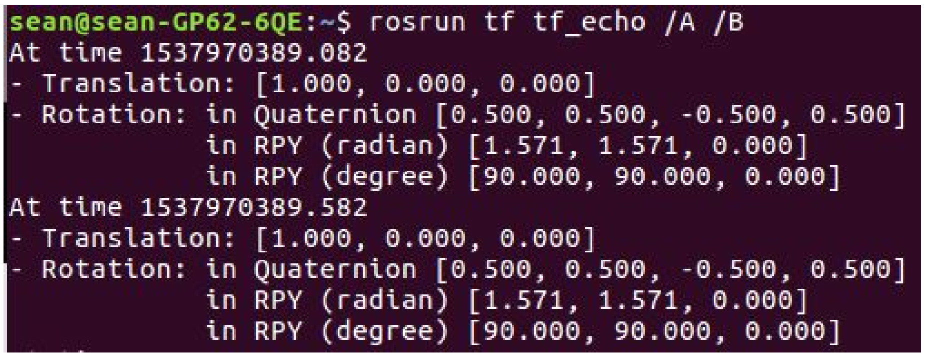
laptop **$ cd ~/ && rosrun tf view_frames**

It will produce an PDF file named frames.pdf, use evince to take a look

laptop **$ evince frames.pdf**


view_frames Result
Recorded at time: 1537969443.476
A
Broadcaster: /test_transform_node
Average rate: 2.222 Hz
Most recent transform: 1537969443.081 (0.394 sec old)
Buffer length: 4.500 sec
B
Broadcaster: /test_transform_node
Average rate: 2.222 Hz
Most recent transform: 1537969443.081 (0.394 sec old)
Buffer length: 4.500 sec
C

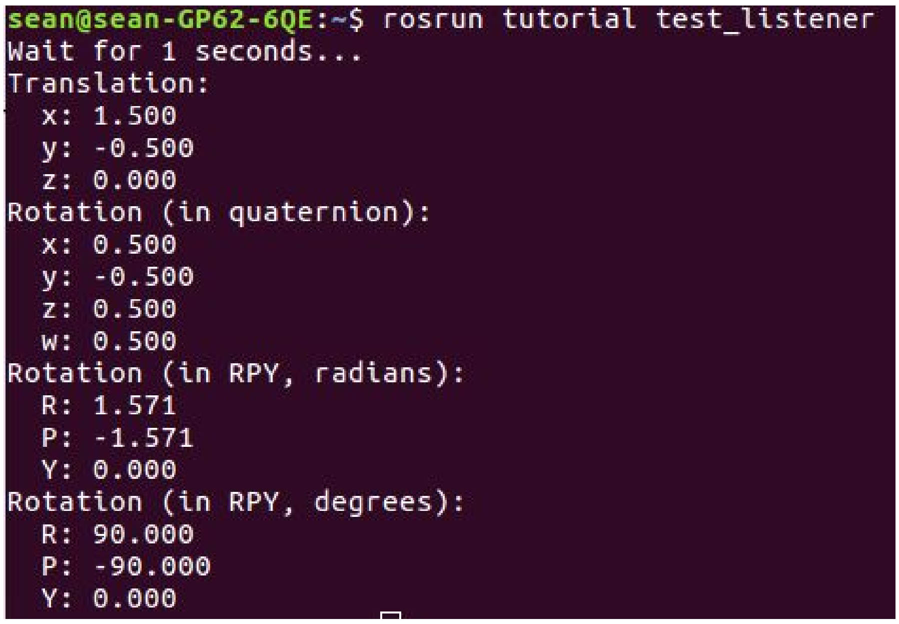
Just like the structure we expected.
Print the transform between frame A and frame B by
laptop $ rosrun tf tf_echo /A /B
Expected result



```
sean@sean-GP62-6QE:~$ rosrun tf tf_echo /A /B
At time 1537970389.082
- Translation: [1.000, 0.000, 0.000]
- Rotation: in Quaternion [0.500, 0.500, -0.500, 0.500]
            in RPY (radian) [1.571, 1.571, 0.000]
            in RPY (degree) [90.000, 90.000, 0.000]
At time 1537970389.582
- Translation: [1.000, 0.000, 0.000]
- Rotation: in Quaternion [0.500, 0.500, -0.500, 0.500]
            in RPY (radian) [1.571, 1.571, 0.000]
            in RPY (degree) [90.000, 90.000, 0.000]
```

Topic 3.2 Listen to the transform

Sometimes we have to know the transformation between two frames, for pick and place scenario, we already know the transformation between the camera and the object, but what we want to know is the one from the base of the robot arm to the object, so that we could pick it with the tool on the arm. TF provides listener API which complete this job, let's study it. In one terminal,
laptop $ rosrun tutorial test_listener
This node will print the transformation between frame A and frame C in one shot rather than tf_echo. The expected result will be looked like the following


```
sean@sean-GP62-6QE:~$ rosrun tutorial test_listener
Wait for 1 seconds...
Translation:
  x: 1.500
  y: -0.500
  z: 0.000
Rotation (in quaternion):
  x: 0.500
  y: -0.500
  z: 0.500
  w: 0.500
Rotation (in RPY, radians):
  R: 1.571
  P: -1.571
  Y: 0.000
Rotation (in RPY, degrees):
  R: 90.000
  P: -90.000
  Y: 0.000
```

You may take a look at
sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws/src/tutorial/src/transform_listener.cpp

Checkpoint 3: Show TA your rviz, and echo the transform from frame A to frame C

Assignment Tasks

Students should do this by themselves

Task 1 Publisher/ Subscriber

In this assignment, we ask you to do the similar task with Topic 2, convert temperature from Celsius to Fahrenheit, while this time we want to know the pressure in mmHg rather than in hPa. Edit
~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws/src/tutorial/src/assignment1.py
such that you have the result as following
Expected Result:
Fig. 1.1 $ rosrun tutorial assignment1.py
[Hint: Don't forget to make your .py file executable by
'$ chmod +x assignment1.py']
Fig. 1.2 $ rostopic list

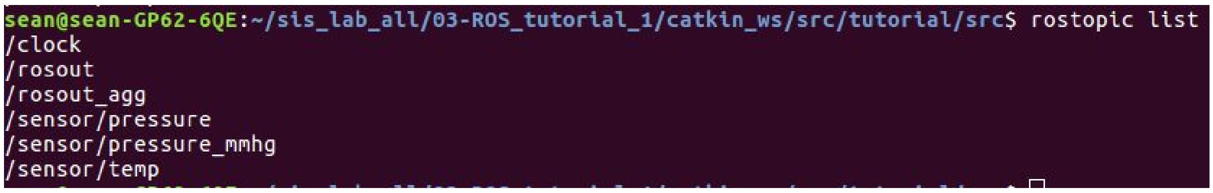
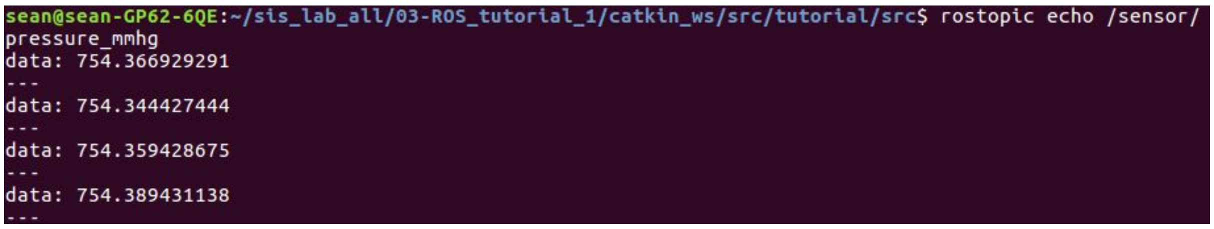

```
sean@sean-GP62-6QE:~/sis_lab_all/03-ROS_tutorial_1/catkin_ws/src/tutorial/src$ rostopic list
/clock
/rosout
/rosout_agg
/sensor/pressure
/sensor/pressure_mmhg
/sensor/temp
```

Fig. 1.3 $ rostopic echo /sensor/pressure_mmhg



```
sean@sean-GP62-6QE:~/sis_lab_all/03-ROS_tutorial_1/catkin_ws/src/tutorial/src$ rostopic echo /sensor/
pressure_mmhg
data: 754.366929291
---
data: 754.344427444
---
data: 754.359428675
---
data: 754.389431138
---
```

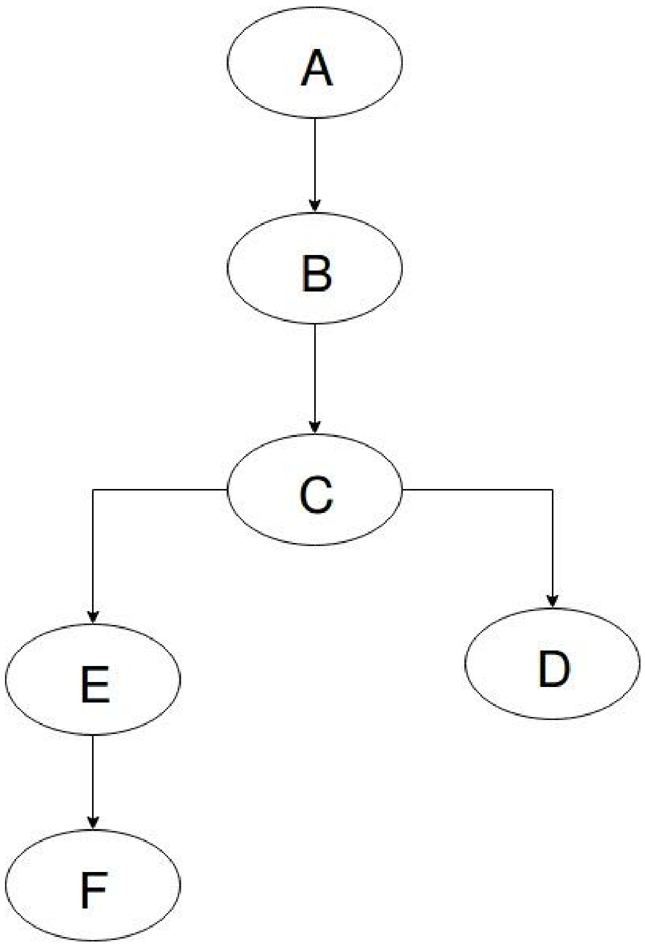
Task 2 TF

In this assignment, we ask you to do the similar task with Topic 3, while this time with more complicated transform tree as following, where

$$T_B^A = \begin{bmatrix} 0 & -1 & 0 & -0.3 \\ 1 & 0 & 0 & 0.4 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ T_A^C = \begin{bmatrix} 0 & 1 & 0 & 0.6 \\ 0 & 0 & -1 & 0.8 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_E^D = \begin{bmatrix} 0.7500 & -0.2165 & 0.6250 & 0.3 \\ 0.4330 & 0.8750 & -0.2165 & 0.3 \\ -0.5000 & 0.4330 & 0.7500 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ T_C^D = \begin{bmatrix} 0 & -0.8660 & 0.5000 & 0.6 \\ 0.5000 & 0.4330 & 0.7500 & 0.4 \\ -0.8660 & 0.2500 & 0.4330 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

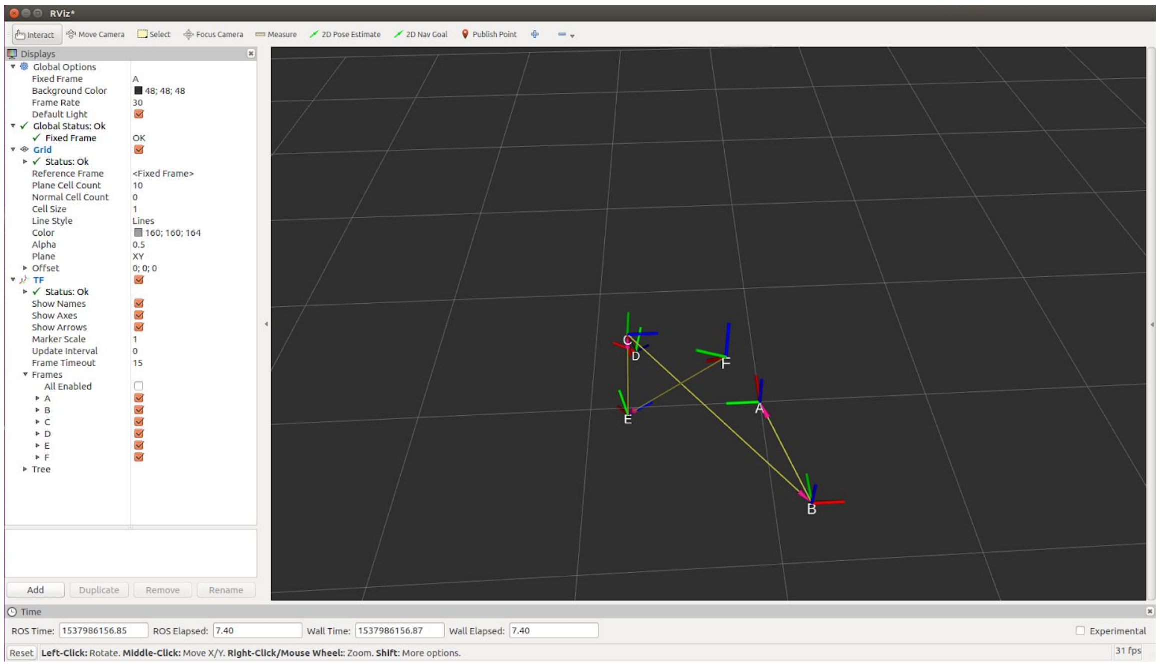
$$T_A^F = \begin{bmatrix} 0.5000 & -0.1464 & 0.8536 & 0.1 \\ 0.5000 & 0.8536 & -0.1464 & 0.2 \\ -0.7071 & 0.5000 & 0.5000 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Edit **~/sis_lab_all_2020/03-ROS_tutorial_1/catkin_ws/src/tutorial/src/assignment2.cpp** such that you have the expected result illustrated below

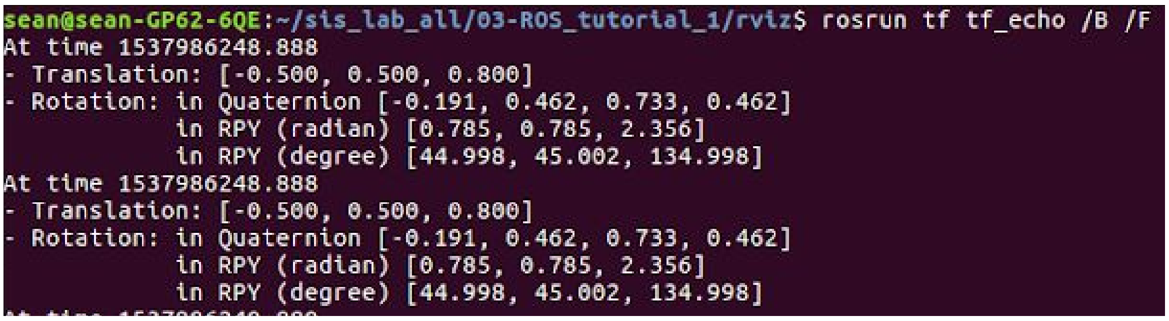
Expected Result:

Fig. 2.1 $ rviz -d tf.rviz



```
Global Options
  Fixed Frame          A
  Background Color     48; 48; 48
  Frame Rate           30
  Default Light        ✓
✓ Global Status: Ok
  ✓ Fixed Frame        OK
Grid                   ✓
  ✓ Status: Ok
  Reference Frame      <Fixed Frame>
  Plane Cell Count     10
  Normal Cell Count    0
  Cell Size            1
  Line Style           Lines
  Color                160; 160; 164
  Alpha                0.5
  Plane                XY
  Offset               0; 0; 0
TF                     ✓
  ✓ Status: Ok
  Show Names           ✓
  Show Axes            ✓
  Show Arrows          ✓
  Marker Scale         1
  Update Interval      0
  Frame Timeout        15
  Frames
    All Enabled        ☐
    A                  ✓
    B                  ✓
    C                  ✓
    D                  ✓
    E                  ✓
    F                  ✓
  Tree

ROS Time: 1537986156.85   ROS Elapsed: 7.40   Wall Time: 1537986156.87   Wall Elapsed: 7.40   ☐ Experimental
Reset  Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Fig. 2.2 $ rosrun tf tf_echo /B /F



```
sean@sean-GP62-6QE:~/sis_lab_all/03-ROS_tutorial_1/rviz$ rosrun tf tf_echo /B /F
At time 1537986248.888
- Translation: [-0.500, 0.500, 0.800]
- Rotation: in Quaternion [-0.191, 0.462, 0.733, 0.462]
            in RPY (radian) [0.785, 0.785, 2.356]
            in RPY (degree) [44.998, 45.002, 134.998]
At time 1537986248.888
- Translation: [-0.500, 0.500, 0.800]
- Rotation: in Quaternion [-0.191, 0.462, 0.733, 0.462]
            in RPY (radian) [0.785, 0.785, 2.356]
            in RPY (degree) [44.998, 45.002, 134.998]
```

[Hint: Many tools available here]

Reference

rospy API document