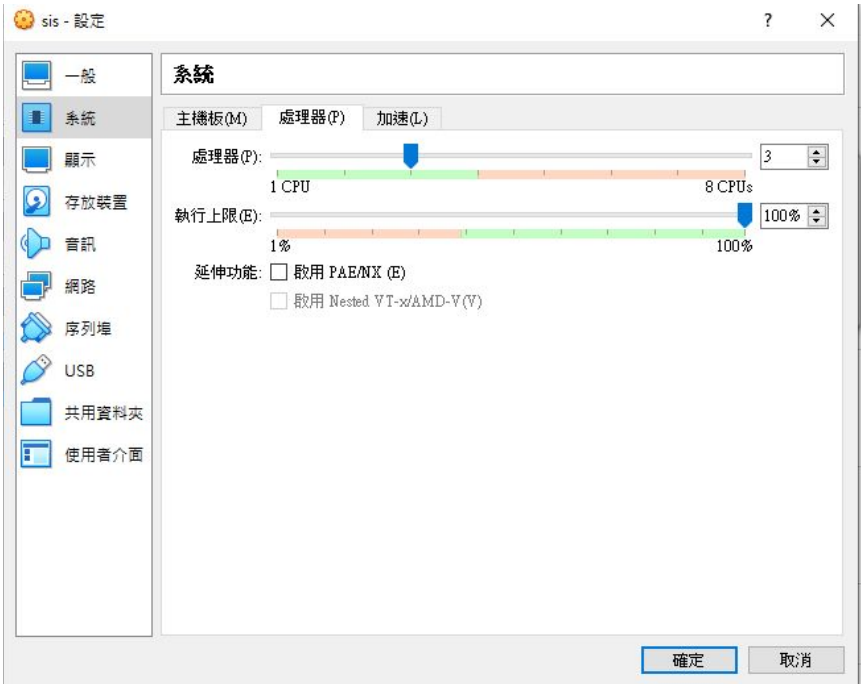
Lab 04: ROS Tutorial 2

By Sean Lu 2018/10, last modified on 03/17, 2020 by Lily Hsu.

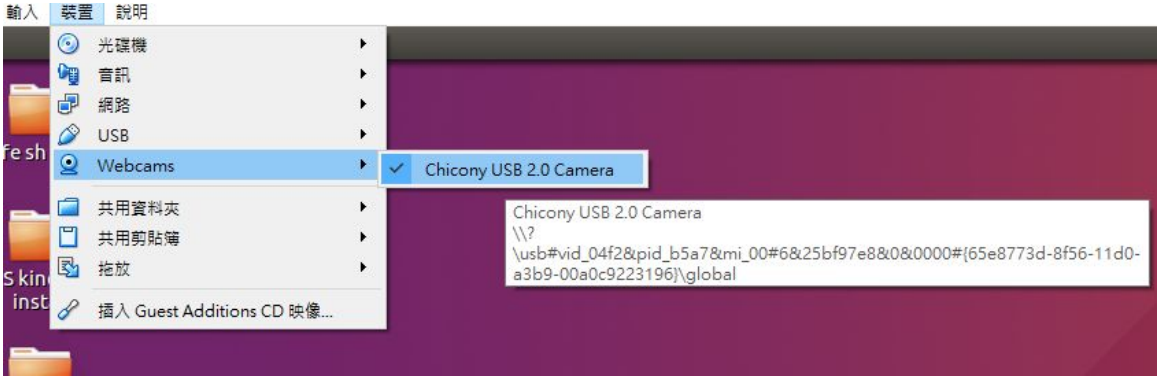
The objective of this tutorial is to introduce ROS service, launch and apriltag_ros package. We will access your laptop USB camera in Topic 1 to do some cool service, and use SR300 in Topic 2 to introduce roslaunch and Apriltag. To turn on your USB camera in ROS, please refer to Hardware and Software Setup part. For Topic 2, please pull the docker image we prepared.

Hardware and Software Setup

You will be given two papers with tags on it, one with two tags and the other with sixteen. Before opening your virtualbox, add the number of CPU used in system.



In VirtualBox, click your Webcam in **Device -> Webcams -> your camera device**
[If not Webcam available, go to here, download and install the extension, after that, enable USB3.0]



Open a terminal (T1),
laptop $ **sudo apt-get install v4l-utils**
laptop $ **cd ~/sis_lab_all_2020 && git stash && git pull**
laptop $ **git stash pop**
laptop $ **cd ~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws && catkin_make**
laptop $ **roscore**
Open another terminal (T2),
laptop $ **cd ~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws && source devel/setup.bash**
laptop $ **rosrun usb_cam usb_cam_node _pixel_format:=mjpeg**
If the above command not work, try:
$ rosrun usb_cam usb_cam_node _pixel_format:=yuyv
If both of them not work,
Open another terminal (T3),
laptop $ **rviz**
In rviz, click ‘ADD’ -> ‘By Topic’ -> ‘/usb_cam/image_raw/Image’ and make sure your image is normal and then turn off rviz window.

Overview

Estimated Time to Finish: 2 hours

After completing this tutorial you should
- understand how to create and use your own service
- be able to use launch file to simplify your project
- develop your application with Apriltag

Topics and Activities

Topic/Activity 1 Service

Problem define: Given a filename, when the client call the service, the server will check if there is a face in the image it receive now, and return the number of face as a string. If there is at least one face, it will save the image with given filename in a specific path.

So, the service we needed will have one request and one response, filename and result, respectively, as a string.

Let's take a look at the service we created, in T3
laptop $ **cd ~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws && source devel/setup.bash**
laptop $ **roscd tutorial/srv**
laptop $ **vim my_service.srv**
You will see the content as following snapshot, where request and response are splitted by three minus symbols.

```
string filename
---
string result
```

Just like creating a new message, here is the demonstration how to create a new service.
Take a look at the description of the service,
laptop $ **rossrv show tutorial/my_service**

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rossrv show tutorial/my_service
string filename
---
string result
```

It is the same as the definition of our .srv file above.

We can separate service into two parts: server and client. Server defines what to do when the service is called, and client calls the service with request, the relation just like the restaurant and we guest.

First, we start a node, which advertises the service 'my_service' and acts as a server, where the source code is available at
~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws/src/tutorial/src/service_server.cpp
laptop $ **rosrun tutorial service_server**
You will see

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rosrun tutorial service_server
[ INFO] [1538301737.164473833]: Successfully loading config file, start to detect face...
```

which indicated that it works normally.
Then open another terminal (T4), run
laptop $ **rosservice list**
This shows all available service in your ROS system, most of them are not related to our usage, what we interested is /my_service.

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rosservice list
/my_service
/rosout/get_loggers
/rosout/set_logger_level
/service_server_node/get_loggers
/service_server_node/set_logger_level
/usb_cam/get_loggers
/usb_cam/image_raw/compressed/set_parameters
/usb_cam/image_raw/compressedDepth/set_parameters
/usb_cam/image_raw/theora/set_parameters
/usb_cam/set_camera_info
/usb_cam/set_logger_level
/usb_cam/start_capture
/usb_cam/stop_capture
```

laptop $ **rosservice info /my_service**

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rosservice info /my_service
Node: /service_server_node
URI: rosrpc://sean-GP62-6QE:44239
Type: tutorial/my_service
Args: filename
```

It shows the node which advertises this service, the URI and type of this service and tell you what arguments are needed to call this service.

We can call the service from command line directly,
laptop $ **rosservice call /my_service "filename: test"**
Then you might see the result below

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rosservice call /m
y_service "filename: test"
result: "1 face in image"
```

and an image named test_YYYYMMDDHHmmss.jpg shown in
~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws/src/tutorial/image/
Other than calling from command line, we can also call the service from our node, the source code is available at
~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws/src/tutorial/src/client_call.cpp
laptop **$ rosrun tutorial call_from_client**
Expected result

```
sean@sean-GP62-6QE:~/sis_lab_all/04-ROS_tutorial_2/catkin_ws$ rosrun tutorial call_from_client
[ INFO] [1538370543.814286007]: 1 face in image
[ INFO] [1538370544.865278447]: 0 face in image
[ INFO] [1538370545.921252815]: 1 face in image
[ INFO] [1538370546.980338775]: 2 faces in image
[ INFO] [1538370548.036284547]: 1 face in image
[ INFO] [1538370549.092118938]: 1 face in image
[ INFO] [1538370550.148322642]: 1 face in image
[ INFO] [1538370551.219723653]: 1 face in image
[ INFO] [1538370552.277040123]: 1 face in image
[ INFO] [1538370553.330485057]: 1 face in image
```

After this node is executed, it will process the incoming images in ten seconds and detect in 1 Hz if there is a face inside. It will save the image with filename prefix test and the current time behind if there is a face.

Check points:

1. **Show TA your result after execute call_from_client.**

2. **Which line declares a service with name my_service at service_server.cpp, and after the service is being called, which callback function will be called?**
3. **At client_call.cpp, which line assigns the client's request? Which line calls the service? How to get the result from the service?**

Topic/Activity 2 Launch and Apriltag

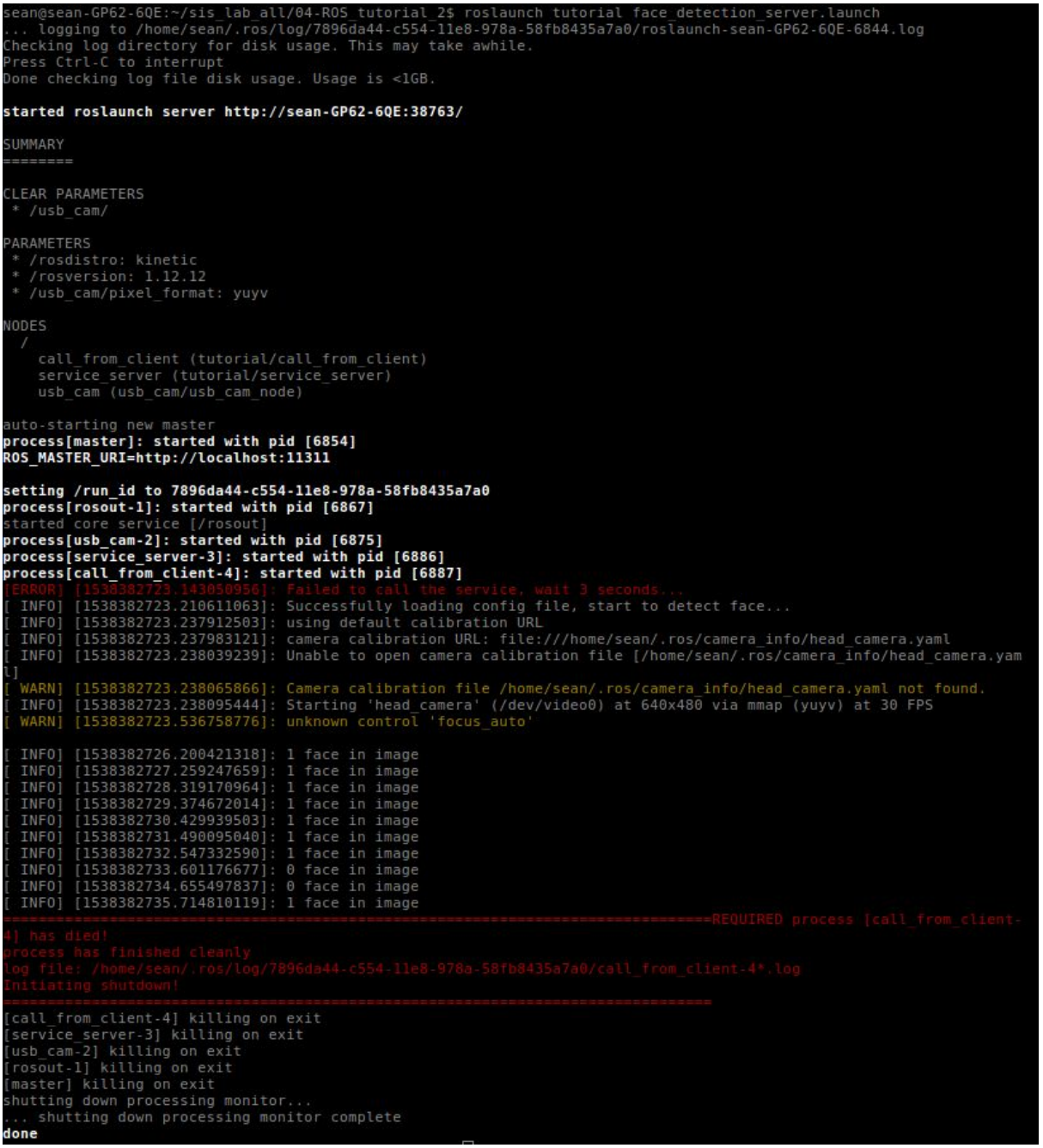
2.1 Is there an easier way to launch multiple nodes?

So far, we use one terminal to start a ROS master, one to turn on your webcam, one to run the service and the last to call the service, what a messy! Is there any easier way to get the same result? Of course YES! But we need the help of roslaunch command and a .launch file.
.launch is a XML format file that help you start up the master, launch multiple nodes locally and remotely via SSH, as well as set parameters on the parameter server.
Though there are many tags you can refer from here, the following is a simple example teaching you how to write a launch file.
laptop **$ roscd tutorial/launch && vim face_detection_server.launch**

1.
```
<launch>
  <!-- Define argument of this launch -->
  <arg name="one_shot" default="true" doc="whether execute multiple times"/>

  <!-- Turn on camera -->
  <node pkg="usb_cam" type="usb_cam_node" name="usb_cam" required="true" clear_params="true">
      <!-- Set parameter -->
      <param name="pixel_format" value="mjpeg"/>
  </node>

  <!-- Face detection service -->
  <node pkg="tutorial" type="service_server" name="service_server" output="screen" required="true"/>

  <!-- Face detection client -->
  <!-- Only execute once -->
  <group if="$(arg one_shot)">
    <node pkg="tutorial" type="call_from_client" name="call_from_client" output="screen" required="true"/>
  </group>
  <!-- After the process end, wait for one second and continue to execute -->
  <group unless="$(arg one_shot)">
    <node pkg="tutorial" type="call_from_client" name="call_from_client" output="screen" respawn="true" respawn_delay="1.0"/>
  </group>
</launch>
```

If your vim is not in highlight mode, **press ':' and type 'set filetype=xml'**

At Line 3, we declare a argument 'one_shot' with default value true, which tells that whether the process will be executed once.
Line 6 through Line 9 specifies a node that will be launched in this launch file, which we turn on the camera.
Line 16 through Line 22 uses a different way, if-unless attributes to specific the client-call node. If the one_shot is true, Line 17 will be process and the remain part will be ignore, and vice versa if one_shot is false. Unless part has respawn and respawn_delay tag which will be restart if the process is died after respawn_delay seconds. Let's try the launch file, Ctrl+C T1 through T4, and at arbitrary terminal,
laptop **$ roslaunch tutorial face_detection_server.launch**



Since call_from_client node is required, and we use default one_shot argument, whole processes will be killed after it finished.

laptop **$ roslaunch tutorial face_detection_server.launch one_shot:=false**

```
SUMMARY
========

CLEAR PARAMETERS
 * /usb_cam/

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12
 * /usb_cam/pixel_format: yuyv

NODES
  /
    call_from_client (tutorial/call_from_client)
    service_server (tutorial/service_server)
    usb_cam (usb_cam/usb_cam_node)

auto-starting new master
process[master]: started with pid [7272]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d3441258-c555-11e8-978a-58fb8435a7a0
process[rosout-1]: started with pid [7285]
started core service [/rosout]
process[usb_cam-2]: started with pid [7294]
process[service_server-3]: started with pid [7304]
process[call_from_client-4]: started with pid [7305]
[ERROR] [1538383304.766464081]: Failed to call the service, wait 3 seconds...
[ INFO] [1538383304.838271582]: Successfully loading config file, start to detect face...
[ INFO] [1538383304.860852569]: using default calibration URL
[ INFO] [1538383304.860915042]: camera calibration URL: file:///home/sean/.ros/camera_info/head_camera.yaml
[ INFO] [1538383304.860960387]: Unable to open camera calibration file [/home/sean/.ros/camera_info/head_camera.yam
l]
[ WARN] [1538383304.860980150]: Camera calibration file /home/sean/.ros/camera_info/head_camera.yaml not found.
[ INFO] [1538383304.861009018]: Starting 'head_camera' (/dev/video0) at 640x480 via mmap (yuyv) at 30 FPS
[ WARN] [1538383305.160148635]: unknown control 'focus_auto'

[ INFO] [1538383307.820670501]: 0 face in image
[ INFO] [1538383308.878871531]: 0 face in image
[ INFO] [1538383309.938835851]: 1 face in image
[ INFO] [1538383310.998394155]: 0 face in image
[ INFO] [1538383312.058424419]: 0 face in image
[ INFO] [1538383313.111356243]: 0 face in image
[ INFO] [1538383314.168975793]: 0 face in image
[ INFO] [1538383315.233473690]: 0 face in image
[ INFO] [1538383316.298378021]: 1 face in image
[ INFO] [1538383317.365042095]: 0 face in image
[call_from_client-4] process has finished cleanly
log file: /home/sean/.ros/log/d3441258-c555-11e8-978a-58fb8435a7a0/call_from_client-4*.log
[call_from_client-4] restarting process
process[call_from_client-4]: started with pid [7447]
[ INFO] [1538383319.740467938]: 1 face in image
[ INFO] [1538383320.795476563]: 0 face in image
[ INFO] [1538383321.851741856]: 0 face in image
[ INFO] [1538383322.912469449]: 0 face in image
[ INFO] [1538383323.969958845]: 0 face in image
^C[call_from_client-4] killing on exit
[service_server-3] killing on exit
[usb_cam-2] killing on exit
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
```

As we expected, after the node finish, it will restart by itself and form a loop.

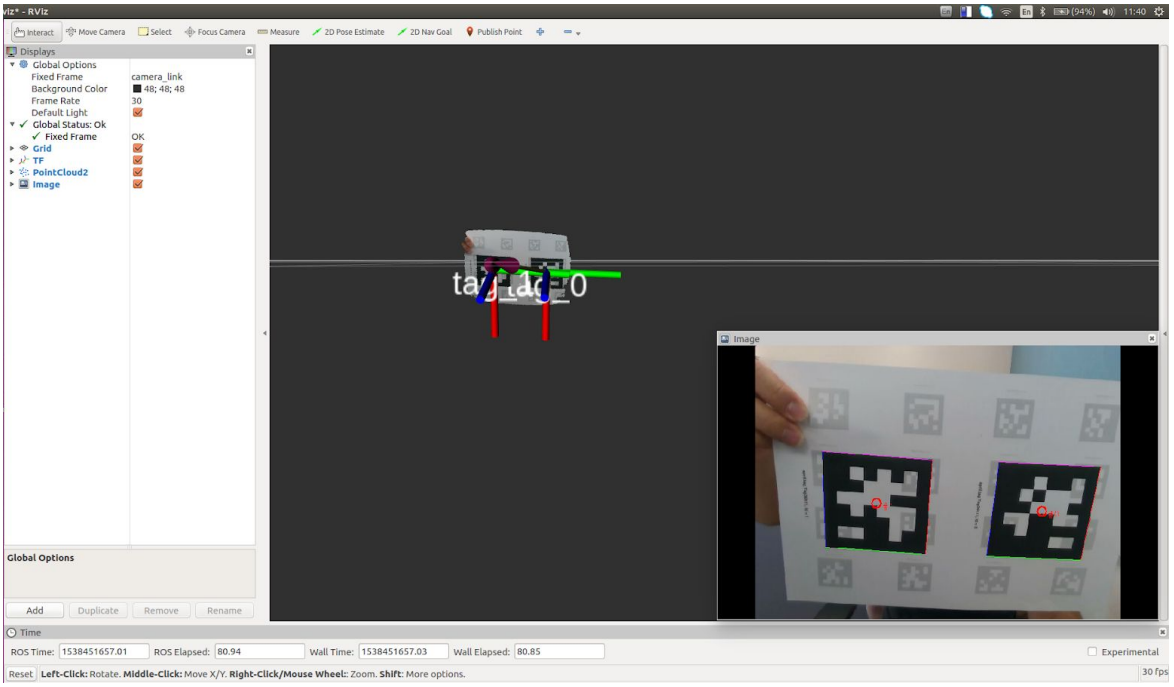
Press Ctrl+C to kill all processes.

2.2 Launch and Apriltag

We can also include a launch file in a bigger launch. Plug SR300 to the USB3.0 port. Check your SR300 in **Device -> USB-> Intel(R) RealSense(TM) Camera SR300**.

laptop $ **roslaunch tutorial topic2.launch**

Click ‘Image’ in ‘Displays’ to enable show the image. If the image includes Apriltag, it will plot a circle in the center of the tag and an TF available in your canvas.



After the camera see two tags, the terminal you launch the file will start to print the information: the distance between the

```
process[camera/ir_rectify_ir-6]: started with pid [14370]
process[camera/depth_rectify_depth-7]: started with pid [14380]
process[camera/depth_metric_rect-8]: started with pid [14389]
process[camera/depth_metric-9]: started with pid [14394]
process[camera/depth_points-10]: started with pid [14410]
process[camera/register_depth_rgb-11]: started with pid [14433]
process[camera/points_xyzrgb_sw_registered-12]: started with pid [14445]
process[camera/depth_registered_sw_metric_rect-13]: started with pid [14455]
process[camera/disparity_depth-14]: started with pid [14459]
process[camera/disparity_registered_sw-15]: started with pid [14469]
process[apriltag_detector-16]: started with pid [14474]
[ INFO] [1538478227.933541550]: Initializing nodelet with 4 worker threads.
process[tag_distance_node-17]: started with pid [14484]
process[rviz-18]: started with pid [14504]
[ INFO] [1538478228.107149763]: Loaded tag config: 0, size: 0.085, frame_name: tag_0
[ INFO] [1538478228.107233485]: Loaded tag config: 1, size: 0.085, frame_name: tag_1
Intel RealSense F200_camera ; 2.60.0.0
Intel RealSense LR200_camera ; 2.0.71.18
Intel RealSense R200_camera ; 1.0.72.06
Intel RealSense SR300_camera ; 3.10.10.0
Intel RealSense ZR300_adapter ; 1.29.0.0
Intel RealSense ZR300_camera ; 2.0.71.28
Intel RealSense ZR300_motion_module ; 1.25.0.0
[ INFO] [1538478228.692060645]: /camera/driver - Detected the following camera:
                                - Serial No: 613204004443, USB Port ID: 2-4-1, Name: Intel RealSense SR300, Camera FW: 3.15.0.0
[ WARN] [1538478228.692098883]: /camera/driver - Detected unvalidated firmware:
                                - 613204004443's current camera firmware is 3.15.0.0, Validated camera firmware is 3.10.10.0
[ INFO] [1538478228.692128884]: /camera/driver - Connecting to camera with Serial No: 613204004443, USB Port ID: 2-4-1
[ INFO] [1538478229.855976746]: /camera/driver - Setting static camera options
[ INFO] [1538478229.860132004]: /camera/driver - Enabling Depth in manual mode
[ INFO] [1538478229.861043638]: /camera/driver - Enabling Color in manual mode
[ INFO] [1538478229.862010884]: /camera/driver - Enabling IR in manual mode
[ INFO] [1538478229.862736436]: /camera/driver - Starting camera
[ INFO] [1538478229.872381071]: /camera/driver - Publishing camera transforms (/tf_static)
[ INFO] [1538478229.872568465]: /camera/driver - Setting dynamic camera options
[INFO] [1538478232.183048]: Distance: 0.138406
[INFO] [1538478232.483916]: Distance: 0.138318
[INFO] [1538478232.834747]: Distance: 0.138305
[INFO] [1538478233.135575]: Distance: 0.138351
[INFO] [1538478233.486282]: Distance: 0.138369
[INFO] [1538478233.787073]: Distance: 0.138334
[INFO] [1538478234.087810]: Distance: 0.138357
[INFO] [1538478234.388533]: Distance: 0.138361
[INFO] [1538478234.689223]: Distance: 0.138388
[INFO] [1538478235.039935]: Distance: 0.138382
[INFO] [1538478235.340689]: Distance: 0.138381
[INFO] [1538478235.641350]: Distance: 0.138381
```

tags, and stop printing if tags not in view.

If rviz has shown the tag, but there isn't any distance information printed in the terminal, edit the "rospy.Duration(3.0)" parameter in "listener.waitForTransform" function from 3 to 5(or more) in "tag_distance.py" file. You may have the same problem during your assignment. Remember to check the listener when you doing your homework.

Check points:

1. **In face_detection_server.launch, what nodes will be run? Where is the different when one_shot is true and false?**
2. **What tags are needed in ‘node’ tag when we want to run the specific node in a launch file?**
3. **What tag is needed when we want to launch a smaller one in a bigger launch file？ How to transport the parameters in it?**

Assignment Tasks

Students should do this by themselves

Task 1 Distance Between Tags Service

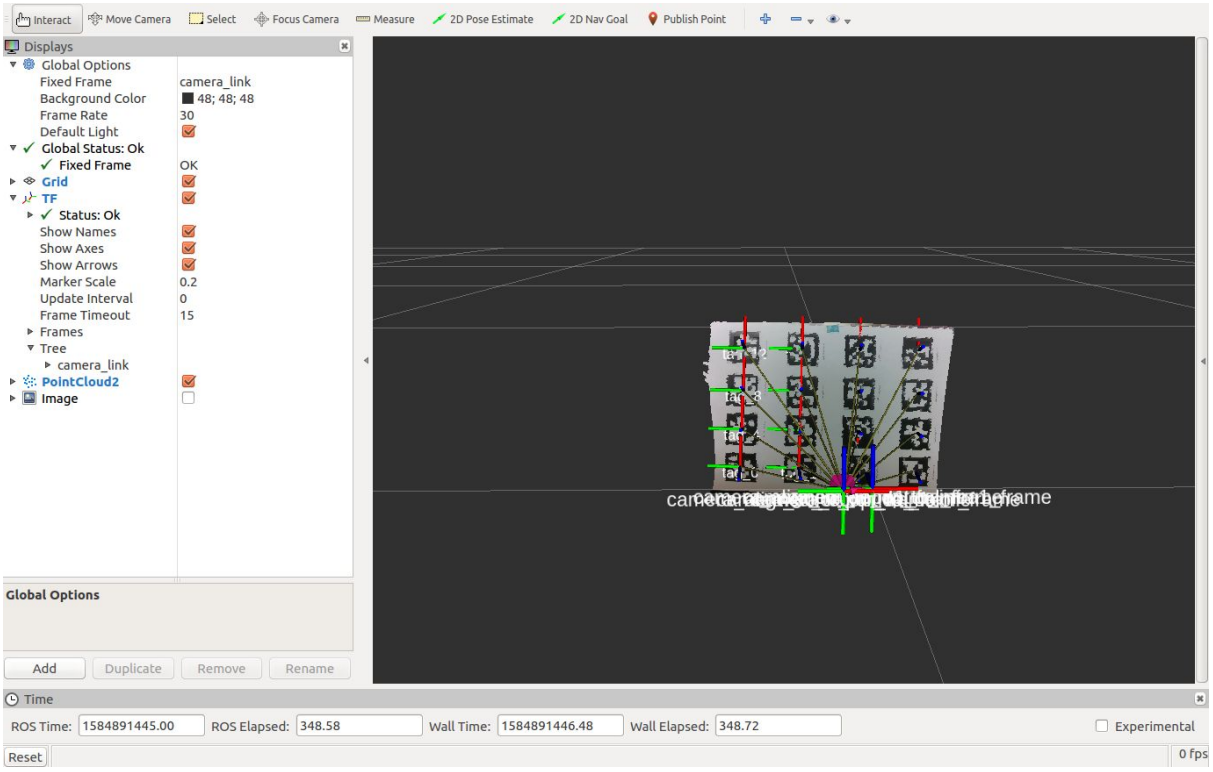
In this assignment, you should edit your codes in VirtualBox. We ask you to implement a service that takes two requests, first tag ID and second tag ID where ID numbers are in range [0, 15], results with a distance from first tag to second one and an information string. The service is pre-defined as **assignment.srv**. Semi finished source code is available at **~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws/src/tutorial/src/assignment.cpp** with three parts unfinished and a semi finished launch file is available at **~/sis_lab_all_2020/04-ROS_tutorial_2/catkin_ws/src/tutorial/launch/assignment.launch** with one part unfinished.

```
bool id_in_range(int id1, int id2){
  bool result;
  /*********************************************************
  //                                                      //
  //                                                      //
  //                Student Implementation                //
  //                                                      //
  //                                                      //
  *********************************************************/
  return result;
}

// Calculate the distance with given transform
double calculateDistance(tf::Transform transform){
  double dist;
  /*********************************************************
  //                                                      //
  //                                                      //
  //                Student Implementation                //
  //                                                      //
  //                                                      //
  *********************************************************/
  return dist;
}

// Service callback
bool serviceCb(tutorial::assignment::Request  &req,
               tutorial::assignment::Response &res){
  if(!id_in_range(req.tag1_id, req.tag2_id)){
    res.result = "Tag ID out of range. Should be 0 through 15.";
    return 1;
  }
  else{
    std::string tag1_str = "tag_", tag2_str = "tag_", parent = "camera_link";
    tag1_str += std::to_string(req.tag1_id);
    tag2_str += std::to_string(req.tag2_id);
    tf::TransformListener listener;
    tf::StampedTransform transform;
    try{
      if(listener.waitForTransform(tag1_str, tag2_str,
                                   ros::Time::now(),
                                   ros::Duration(5.0)))
        {
        /*********************************************************
        //                                                      //
        //                                                      //
        //                Student Implementation                //
        //                                                      //
        //                                                      //
        *********************************************************/}
      else{
        res.result = "Times out.";
        return 1;
      }
    }
    catch (tf::TransformException ex){
      ROS_ERROR("%s", ex.what());
      res.result = "Exception.";
      return 1;
    }
    res.distance = calculateDistance(transform);
```

```
<launch>
  <!-- SR300 -->
  <include file="$(find realsense_camera)/launch/sr300_nodelet_rgbd.launch">
    <arg name="color_fps" value="10"/>
    <arg name="depth_fps" value="10"/>
  </include>
  <!-- Apriltag -->
  <include file="$(find apriltags_ros)/launch/assignment.launch"/>
  <!-- Calculate distance node -->
  <!-- Assignment -->
  <!-- TODO -->
  <!-- rviz -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find tutorial)/config/rviz/topic2.rviz"/>
</launch>
```

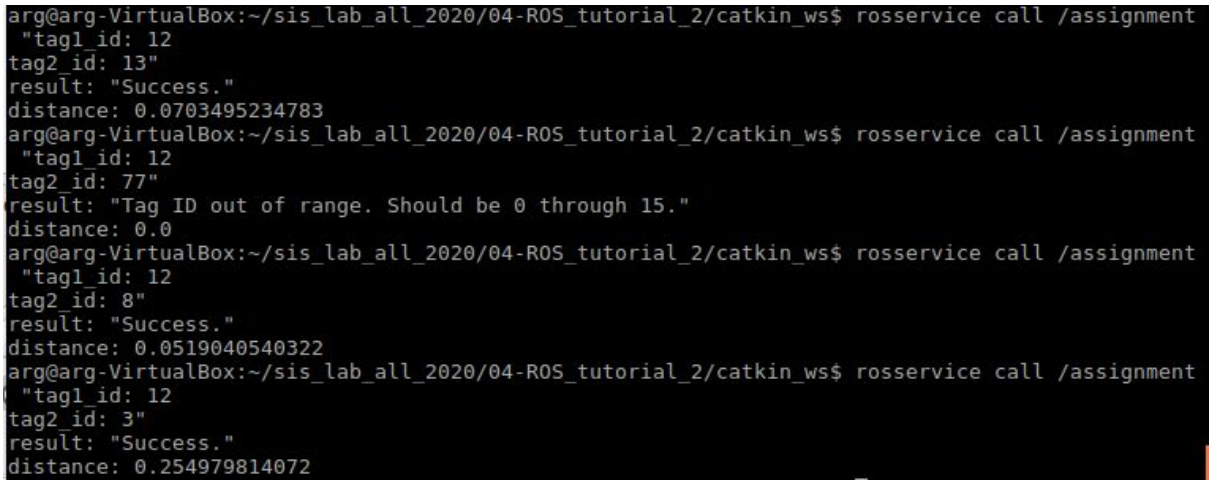
Implement two files above in your VirtualBox.
laptop **$ cd sis_lab_all_2020**
laptop **$ git checkout -b devel-your_student_id**
laptop **$ catkin_make && source devel/setup.bash**

After you set your workspace, run following command to get the expected result
laptop $ **roslaunch tutorial assignment.launch**



In new terminal,
laptop $ **source devel/setup.bash && rosservice call /assignment "tag1_id: 12 tag2_id: 13"**
[Do not directly type the command, you will get the template after press 'TAB' behind /assignment]

[Expected Result]



[Hint: This is a good example from offical]

After you make sure your codes work and have saved the results.
laptop $ cd sis_lab_all_2020
laptop $ git checkout 04-ROS_tutorial_2/catkin_ws/src/tutorial/src/assignment.cpp
laptop $ git checkout 04-ROS_tutorial_2/catkin_ws/src/tutorial/launch/assignment.launch
laptop $ git checkout master

Reference

roslaunch
rosservice