Introduction to Robot Operating System

Udit Singh Parihar
IIIT Hyderabad
udit.singh@research.iiit.ac.in

May 27, 2020

Why ROS?

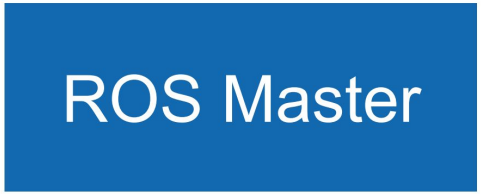
1. Peer to Peer
    a. Each node is a subscriber (receive) and a publisher (publish).
2. Hardware abstraction and low level device control
3. Multi-lingual
    a. ROS nodes can be written in any language (Python and C++).
4. Distributed over across system
5. Open Source and has a large contributing community

ROS Components

1. Nodes
2. Master
3. Messages
4. Topics
5. Parameter Server
6. Services
7. Bags

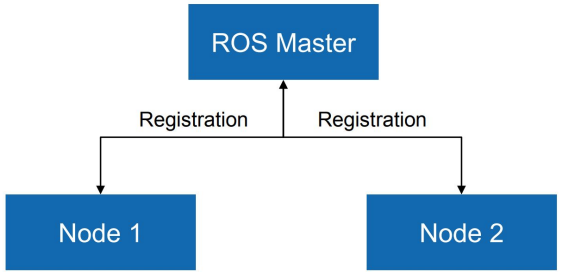
ROS Master

1. Every node registers at startup with the master.
2. It helps in nodes to find each other, exchange messages or invoke services.
3. Starting master node
    a. roscore



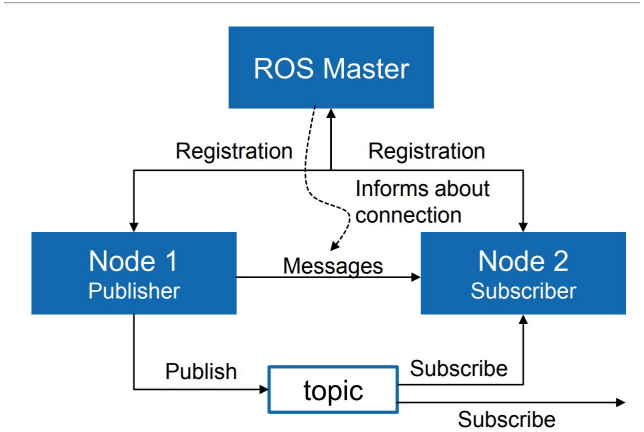
ROS Nodes

1. Nodes are processes that perform computation.
2. Single purpose, executable program.
3. Contain the main logic of the node.
4. Listing active nodes
   a. rosnode list
5. Running a node
   a. rosrun package_name node_name
6. Seeing information about a node
   a. rosnode info node_name



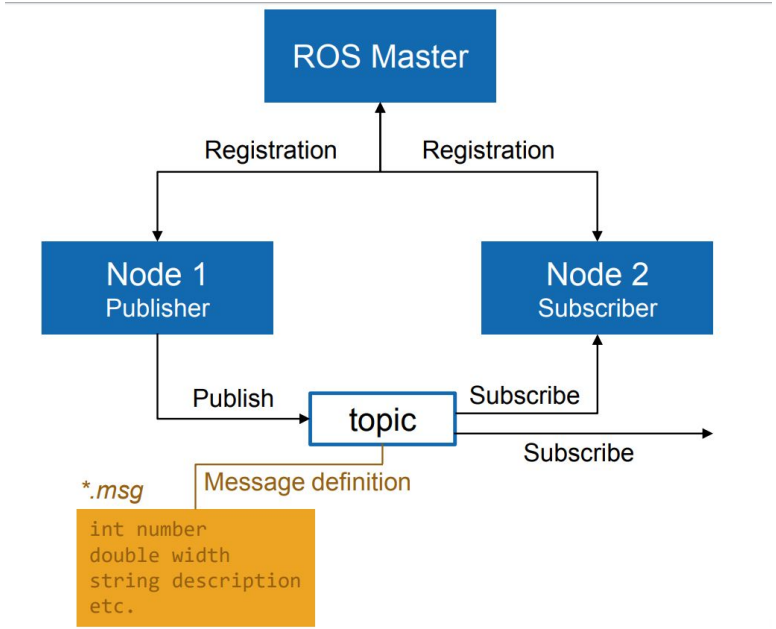
ROS Topics

1. Nodes communicate over topics.
2. Node sends message by publishing to a given topic and receive messages by subscribing to a topic.
3. Multiple concurrent publishers and subscribers.
   a. Typically there is a 1 publisher and n subscribers.
4. List active topics
   a. rostopic list
5. Print data stored in a topic
   a. rostopic echo /topic
6. Publishing data on a topic
   a. rostopic pub /topic /msg_type [args]
7. Viewing publishing rate of a topic
   a. rostopic hz /topic



ROS Messages

1. Nodes communicate with each other by passing messages.
2. A message is simply a data structure, comprising typed fields.
3. Standard primitive types, e.g. integer, floating point and arrays are supported.
4. Stored in *.msg files.



ROS Messages Example

1. geometry_msgs/Point.msg is used to define position of a robot.
2. sensor_msgs/Image.msg is used to define image with its properties captured by the sensor.
3. geometry_msgs/PoseStamped.msg is used to define position, orientation and time of receiving of the message.

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

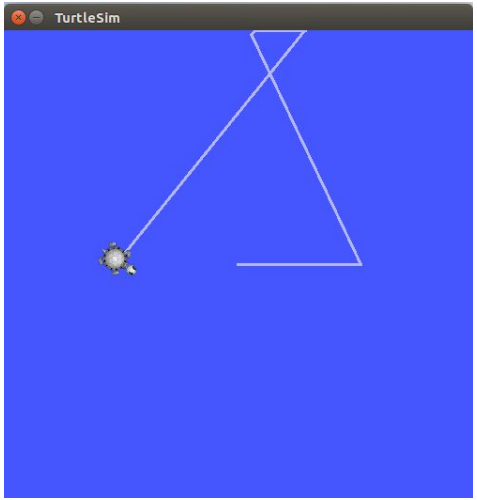
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

ROS TurtleSim Example

1. Toy example demonstrating 2 functional nodes.
2. turtlesim and teleop_turtle are communicating using /turtle1/cmd_vel

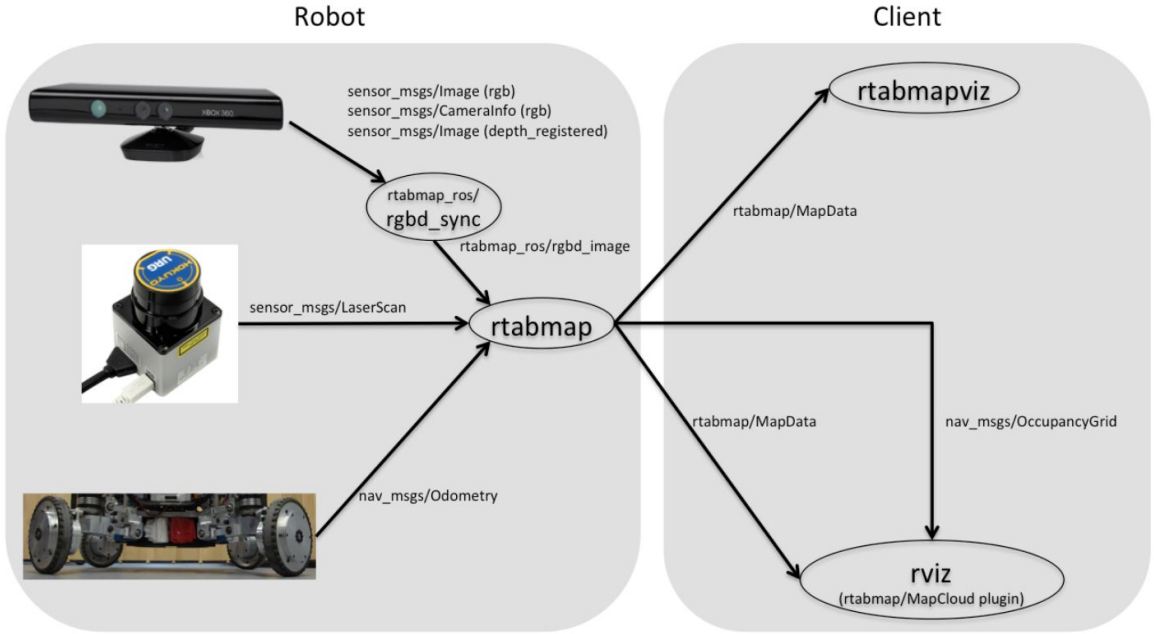



```
cair@ThinkPad:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
cair@ThinkPad:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
cair@ThinkPad:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://ThinkPad:35535/)

Subscribers:
 * /turtlesim (http://ThinkPad:41829/)
```

```
cair@ThinkPad:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

ROS Components on a Mapping Robot

1. Robot is equipped wheel odometry, lidar and kinect sensor.
2. Messages are shown on the arrow and nodes are inside ellipse.
3. Topics from kinect are processed by rtabmap_ros/rgbd_sync node to generate a 3D information.
4. Messages are used by rtabmap node to generate map of the environment and is published on rtabmap/MapData.
5. Visualization of 3D generated is done by rviz node.



ROSBag files

1. Bags are a format for saving and playing back ROS message data.
2. Binary format with file extension *.bag
3. Suited for logging and recording datasets for later visualization and analysis
4. Recording topics in a bag file
   a. rosbag record <topic-names>
5. Displaying a human readable summary of a bag file
   a. rosbag info bag_name.bag
6. Reading and playing the contents of a bag file in a time-synchronized fashion.
   a. rosbag play bag_name.bag

```
cair@ThinkPad:~/backup/floor_loop$ rosbag info lab_corridor5.bag
path:        lab_corridor5.bag
version:     2.0
duration:    14:07s (847s)
start:       Feb 14 2020 16:11:46.08 (1581676906.08)
end:         Feb 14 2020 16:25:53.47 (1581677753.47)
size:        36.3 GB
messages:    92966
compression: none [25350/25350 chunks]
types:       nav_msgs/Odometry      [cd5e73d190d741a2f92e81eda573aca7]
             sensor_msgs/CameraInfo [c9a58c1b0b154e0e6da7578cb991d214]
             sensor_msgs/Image      [060021388200f6f0f447d0fcd9c64743]
             tf2_msgs/TFMessage     [94810edda583a504dfda3829e70d7eec]
topics:      /RosAria/pose                                 8470 msgs    : nav_msgs/Odometry
             /camera/aligned_depth_to_color/image_raw     25339 msgs    : sensor_msgs/Image
             /camera/color/camera_info                    25343 msgs    : sensor_msgs/CameraInfo
             /camera/color/image_raw                      25343 msgs    : sensor_msgs/Image
             /tf                                           8470 msgs    : tf2_msgs/TFMessage
             /tf_static                                       1 msg     : tf2_msgs/TFMessage
```

Creating ROS Workspace

1. ROS codes should be maintained inside catkin workspace.
2. Creating and building catkin workspace
   a. mkdir -p ss_ws/src
   b. cd ss_ws
   c. catkin_make
3. All the ROS nodes would reside in ss_ws/src as packages.
4. General structure of a ROS workspace looks like:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Creating catkin package for ROS Node

1. cd ss_ws/src
2. General command for creating a package
    a. catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
3. In particularly for our case:
    a. catkin_create_pkg tut1 std_msgs rospy roscpp
4. Directory structure looks like:

    1. build
    2. devel
    3. src
        a. CMakeLists.txt
        b. tut1
            i. CMakeLists.txt
            ii. include
            iii. package.xml
            iv. src
                1. listener.py
                2. talker.py

Writing Code for Basic Publisher and Subscriber

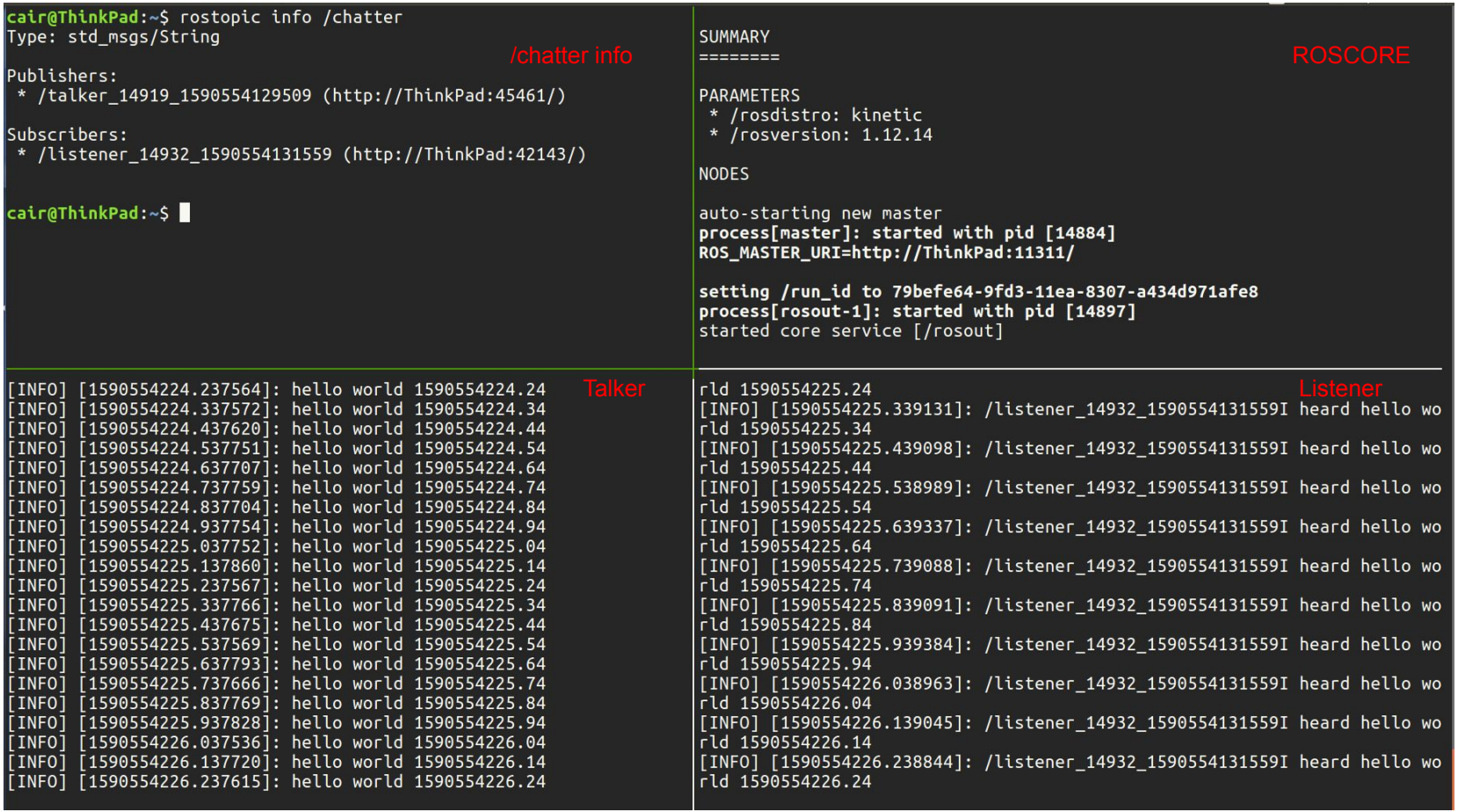
talker.py

```python
#!/usr/bin/env python


import rospy
from std_msgs.msg import String     # Standad ROS string messages


def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)     # Creating a publisher `pub` on `chatter` topic with maximum queue size of 10
    rospy.init_node('talker', anonymous=True)       # Initializing the current node as `talker`
    rate = rospy.Rate(10)   # Publishing rate at 10 hz

    while not rospy.is_shutdown():      # Check for Ctrl+C
        hello_str = "hello world %s" % rospy.get_time()
        print(hello_str)
        pub.publish(hello_str)
        rate.sleep()


if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

listener.py

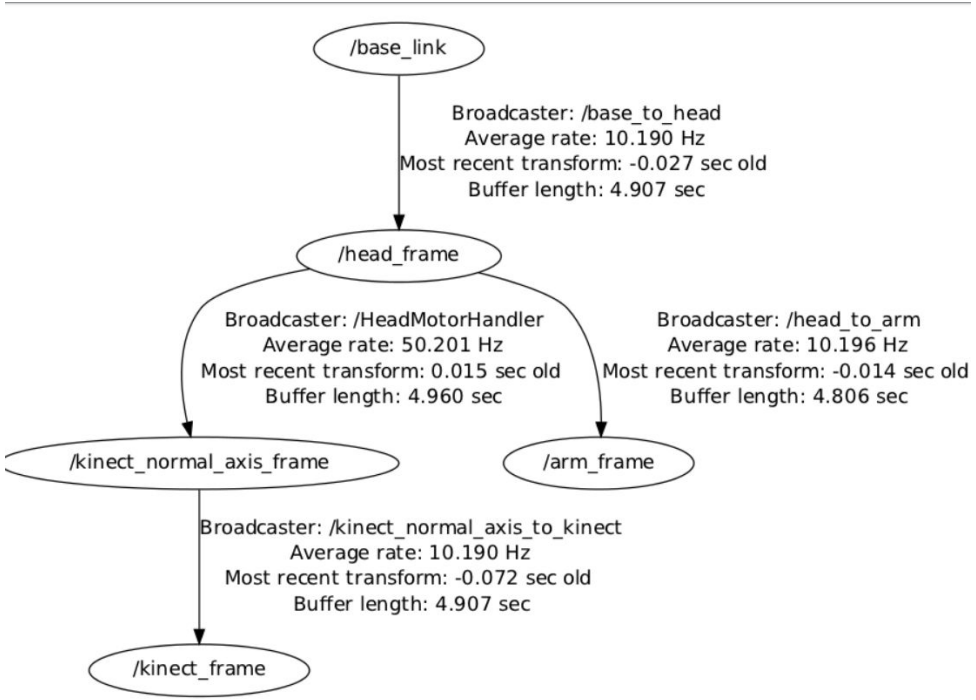
```python
#!/usr/bin/env python


import rospy
from std_msgs.msg import String


def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)      # ROS function to print on terminal with timestamps


def listener():
    rospy.init_node('listener', anonymous=True)     # anonymous=True would ensure unique name for this listener, in case of multiple listeners

    rospy.Subscriber("chatter", String, callback)

    rospy.spin()        # spin() simply keeps python from exiting until this node is stopped


if __name__ == '__main__':
    listener()

```

Terminal Output of Talker and Listener



TF Transformation Package

1. TF is a package that lets the user keep track of multiple coordinate frames
2. Maintains the relationship between coordinate frames in a tree structure buffered in time
3. Lets the user transform points, vectors, etc. between coordinate frames
4. A tf tree showing parent-child relationship between various components of humanoid robot



TF Listener and Broadcaster

1. Turtle Video

Link: https://drive.google.com/open?id=1InMmWjA9B6txlgjqN6KRUjGngh2VGt68

TF Listener and Broadcaster Code

```python
#!/usr/bin/env python
import roslib
import rospy

import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")

if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

broadcaster.py

```python
#!/usr/bin/env python
import roslib
import rospy
import math
import tf
import geometry_msgs.msg
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('turtle_tf_listener')

    listener = tf.TransformListener()

    rospy.wait_for_service('spawn')
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    spawner(4, 2, 0, 'turtle2')

    turtle_vel = rospy.Publisher('turtle2/cmd_vel', geometry_msgs.msg.Twist,queue_size=1)

    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
            continue

        angular = math.atan2(trans[1], trans[0])
        linear = math.sqrt(trans[0] ** 2 + trans[1] ** 2)
        cmd = geometry_msgs.msg.Twist()
        cmd.linear.x = linear
        cmd.angular.z = angular
        turtle_vel.publish(cmd)

        rate.sleep()
```

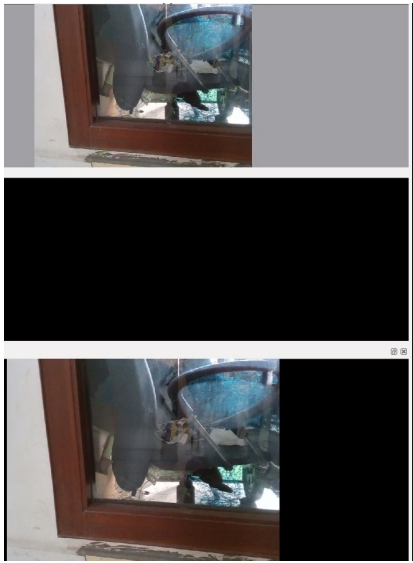
listener.py

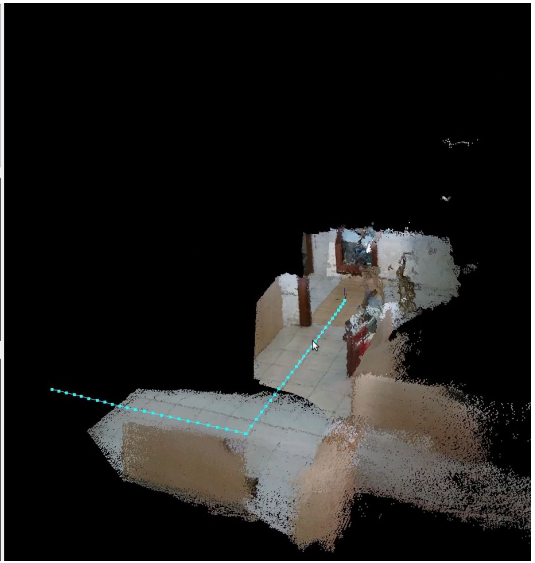
```xml
<launch>
    <!-- Turtlesim Node-->
    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

    <node name="turtle1_tf_broadcaster" pkg="learning_tf" type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle1" />
    </node>
    <node name="turtle2_tf_broadcaster" pkg="learning_tf" type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle2" />
    </node>

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

start.launch

Mini Mapping Framework

1. RTABMAP demo video
2. Simultaneously subscribe to rgb, depth and odometry information from a bag file.
3. Stores data in a directory.
4. Generate individual point cloud using rgb and depth information.
5. Stitch adjacent point clouds using odometry information.
6. May be needed to downsample assembled pointcloud for computational purpose.
7. You can use Open3d for pointcloud visualization and downsampling.





Link: https://drive.google.com/open?id=1Ydn86ETCbKKdYTMfbwSc-vif_oV91gsP