ILLINOIS

LAB 2: ROS

ABE 424/ ECE 498

Principles of Mobile Robotics

Girish Chowdhary

Andres E Baquero Velasquez

Topics to be covered

- ROS definition
- ROS Installation
- Workspace
- ROS Package
- ROS Node
- Subscriber and Publisher

ROS - Definition

ROS (Robotic Operating System) is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

http://wiki.ros.org/ROS/Introduction

https://www.ros.org/about-ros/

ROS - Installation

1. Setup your sources.list:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

2. Set up your keys:

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

3. Installation

3.1. Updating Debian package index:

sudo apt update

3.2. Desktop-Full Install:

sudo apt install ros-melodic-desktop-full

4. Environment Setup:

echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc

http://wiki.ros.org/melodic/Installation/Ubuntu

ROS - Workspace

1. Create the workspace folder:

mkdir -p ~/catkin_ws/src

2. Go to the workspace folder:

cd ~/catkin_ws/

3. Compile your workspace using catkin_make

catkin_make

4. Sourcing your workspace

source devel/setup.bash

5. Verify if your workspace is properly overlayed by the setup script

echo $ROS_PACKAGE_PATH

If your workspace is ok then you should see this message in your screen: /home/youruser/catkin_ws/src:/opt/ros/kinetic/share

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

ROS - Package

ROS packages are the individual units of ROS software which contains all source code, data files, build files and dependencies. The structure of a ROS package is:

1. Launch Folder: Contains the launch files

2. Include Folder: Contains the additional libraries or .h files that the cpp files need

3. Src Folder: Contains all source files (cpp files and python scripts)

4. CmakeLists.txt: List of cmakerules for compilation

5. package.xml: Package information and dependencies

http://wiki.ros.org/ROS/Concepts

ROS - Package

1. Create the ROS package

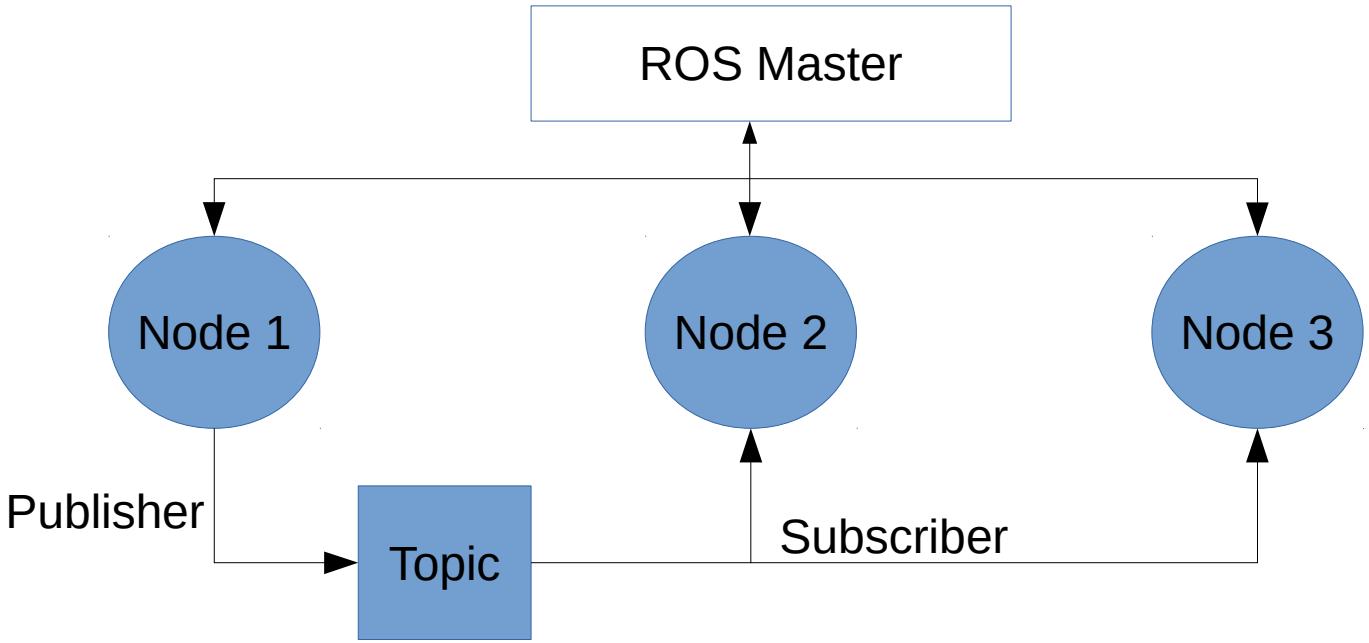
cd ~/catkin_ws/src
catkin_create_pkg “name of the package” std_msgs rospy roscpp

2. Compile the ROS package

cd ~/catkin_ws
catkin_make

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

ROS - Node



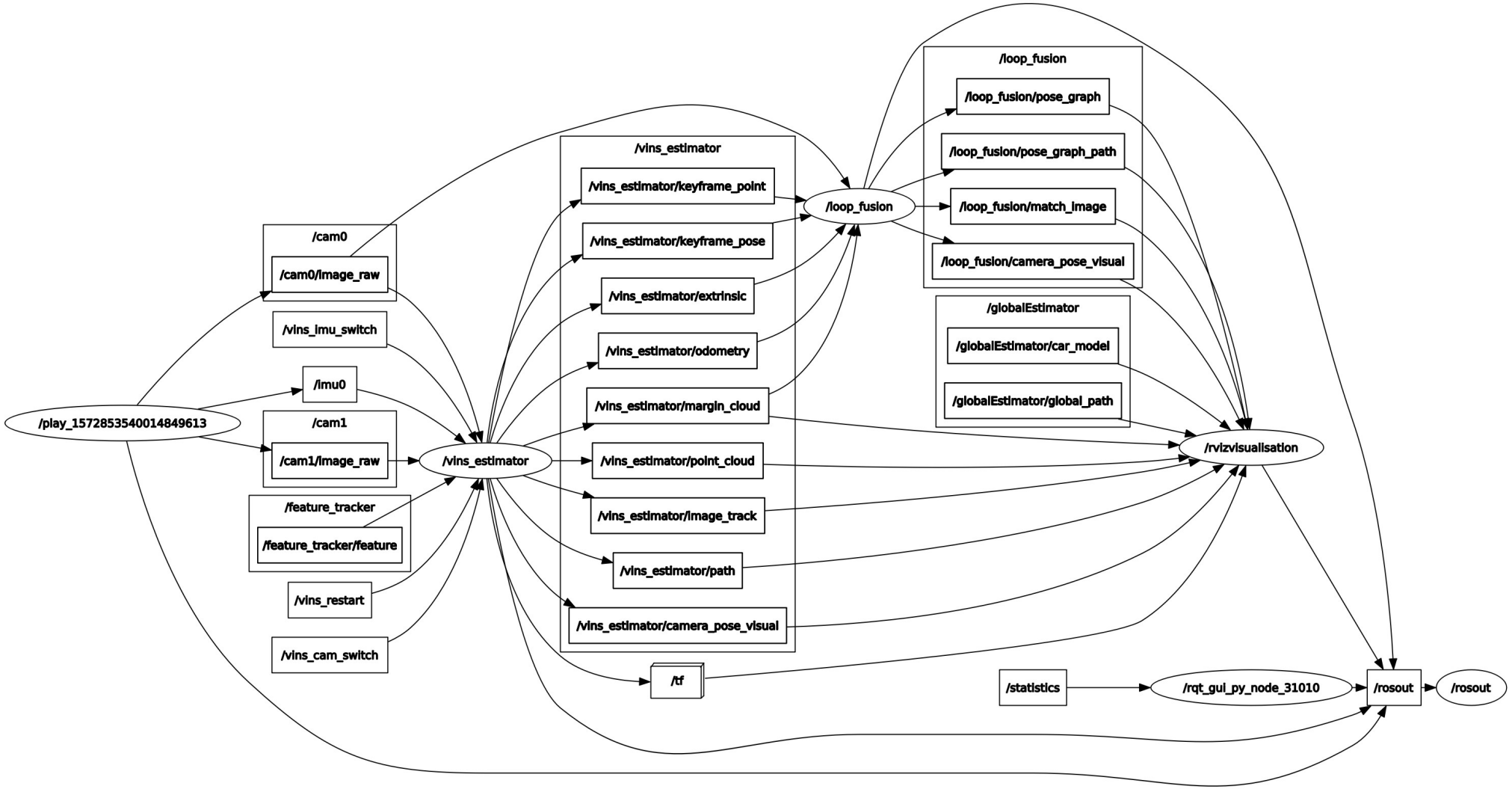
http://wiki.ros.org/ROS/Concepts

Master: An intermediate program that connects ROS nodes. To run the master node you need to use “Roscore”.

Nodes: They are processes that perform computation. They are written with the use of a ROS client library, such as roscpp or rospy.

Topics: Named buses in which ROS nodes can send a message. A node can publish or subscribe any number of topics.

ROS - Node


/loop_fusion
/loop_fusion/pose_graph
/loop_fusion/pose_graph_path
/loop_fusion/match_image
/loop_fusion/camera_pose_visual
/vins_estimator
/vins_estimator/keyframe_point
/vins_estimator/keyframe_pose
/vins_estimator/extrinsic
/vins_estimator/odometry
/vins_estimator/margin_cloud
/vins_estimator/point_cloud
/vins_estimator/image_track
/vins_estimator/path
/vins_estimator/camera_pose_visual
/globalEstimator
/globalEstimator/car_model
/globalEstimator/global_path
/cam0
/cam0/image_raw
/cam1
/cam1/image_raw
/feature_tracker
/feature_tracker/feature
/loop_fusion
/vins_estimator
/rvizvisualisation
/play_1572853540014849613
/vins_imu_switch
/imu0
/vins_restart
/vins_cam_switch
/tf
/statistics
/rqt_gui_py_node_31010
/rosout
/rosout

ROS - Node

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 22 10:22:34 2020

@author: andres
"""

import rospy

rospy.init_node("printer_node")
print("Hi Robotic Class")
```

To run the rospy node, you can use the follow command line:

python2.7 “path of the folder where you have your script”

rosrun “package_name” “python_scrypt_name”.py

http://wiki.ros.org/ROS/Concepts

ROS - Publisher

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 22 10:41:09 2020

@author: andres
"""

import rospy
from std_msgs.msg import Int32

class PublisherNode():

    def __init__(self):
        pub = rospy.Publisher('publisher_node_example',Int32,queue_size=1)
        rate = 10.0
        A = 0
        while not rospy.is_shutdown():
            pub.publish(A)
            A+=1#this is equal to A = A+1
            if rate:
                rospy.sleep(1/rate)
            else:
                rospy.sleep(1.0)

if __name__ == '__main__':
    #Initialize the node and name it
    rospy.init_node('rospy_publishe', anonymous = True)
    #go to the init function
    try:
        ne = PublisherNode()
    except rospy.ROSInterruptException: pass

```

To run the rospy node, you can use the follow command line:

Python2.7 “path of the folder where you have your script”

rosrun “package_name” “python_scrypt_name”.py

ROS - Subscriber

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 22 11:18:12 2020

@author: andres
"""

import rospy
from std_msgs.msg import Int32

class listenerNode():


    number = 0.0
    def __init__(self):

        rate = 10.0
        rospy.Subscriber("publisher_node_example", Int32, self.callback)

        while not rospy.is_shutdown():
            print("The number is: ",listenerNode.number)
            #rospy.loginfo("yaw_rate: %s", yawrate)

            if rate:
                rospy.sleep(1/rate)
            else:
                rospy.sleep(1.0)


    def callback(self,msg):
        listenerNode.number = msg.data
        rospy.loginfo("number %s", self.number)



# Main function.
if __name__ == '__main__':
    # Initialize the node and name it.
    rospy.init_node('rospy_listener_example', anonymous = True)
    # Go to the main loop.
    #try:
    ne = listenerNode()
```

To run the rospy node, you can use the follow command line:

Python2.7 “path of the folder where you have your script”

rosrun “package_name” “python_scrypt_name”.py

ROS - Subscriber+Publisher

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 22 11:18:12 2020

@author: andres
"""

import rospy
from std_msgs.msg import Int32

class listenerNode():


    number = 0.0
    A=0.0
    def __init__(self):

        rate = 10.0
        rospy.Subscriber("publisher_node_example", Int32, self.callback)
        pub = rospy.Publisher('publisher_subscriber',Int32,queue_size=1)
        while not rospy.is_shutdown():
            A = listenerNode.number * 2
            print("The number is: ",listenerNode.number)
            pub.publish(A)
            #rospy.loginfo("yaw_rate: %s", yawrate)

            if rate:
                rospy.sleep(1/rate)
            else:
                rospy.sleep(1.0)


    def callback(self,msg):
        listenerNode.number = msg.data
        rospy.loginfo("number %s", self.number)



# Main function.
if __name__ == '__main__':
    # Initialize the node and name it.
    rospy.init_node('rospy_listener_example', anonymous = True)
    # Go to the main loop.
    #try:
    ne = listenerNode()

```

To run the rospy node, you can use the follow command line:

Python2.7 “path of the folder where you have your script”

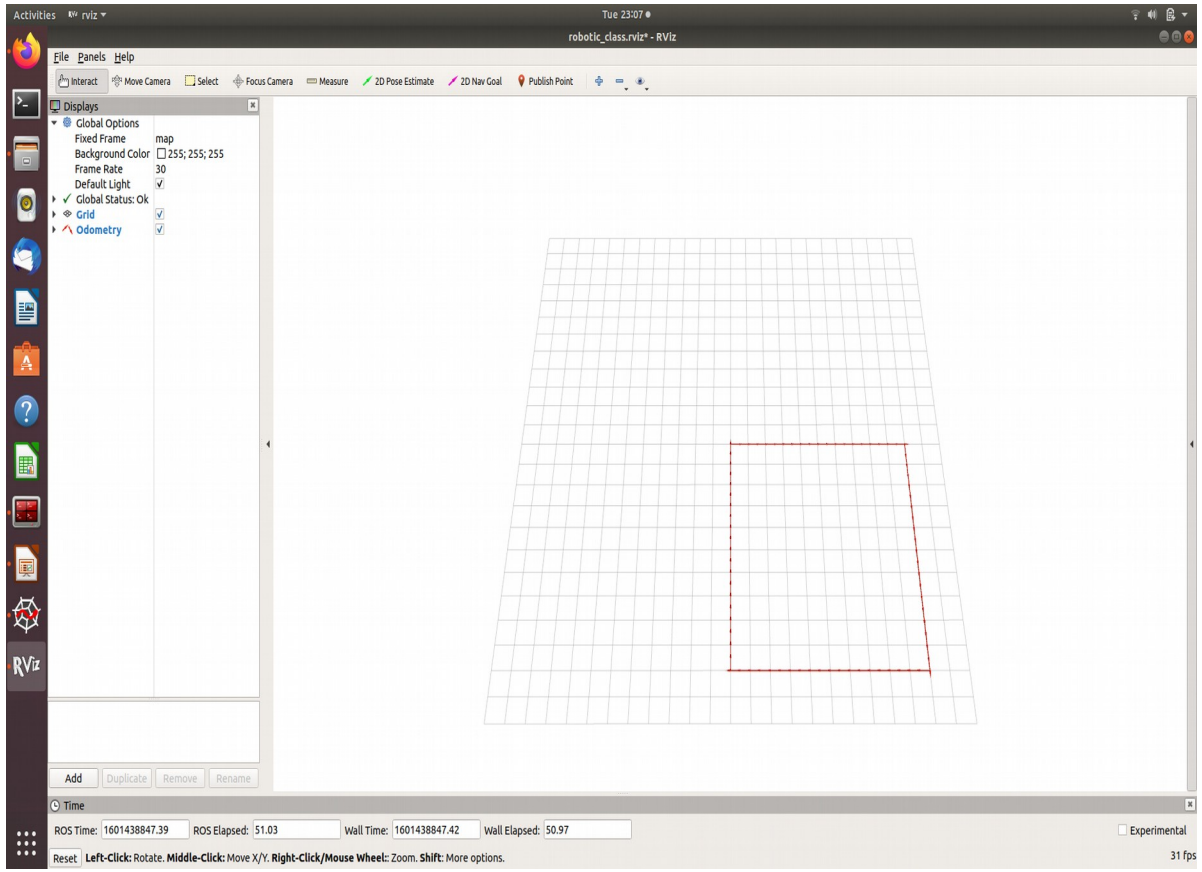
rosrun “package_name” “python_scrypt_name”.py

ROS – Odometry Example

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 28 00:13:45 2020

@author: andres
"""

import rospy
from nav_msgs.msg import Odometry
import math
import tf
from tf.transformations import quaternion_from_euler



pub_odom = rospy.Publisher('odom', Odometry, queue_size=1)#the Publisher is initialized in this line


class listenerNode():


    def __init__(self):
        self.loop_hertz = 5.0#loop frequency
        self.x=0.0
        self.y = 0.0
        self.theta = 0.0
        self.vx = 0.6
        self.yaw_rate = 0.2

    def run(self):
        self.rate = rospy.Rate(self.loop_hertz)#this line is used to declare the time loop
        self.br = tf.TransformBroadcaster()#Initialize the object to be used in the frame transformation
        while not rospy.is_shutdown():

            self.linearvelocity()#call the function used to create the Odometry ROS message
            self.br.sendTransform((self.x, self.y, 0.0), self.q, rospy.Time.now(),"/base_link" , "/map")#this line is used to
            print(self.x)
            pub_odom.publish(self.odom)
```

```python
##########Replace this part with the x and y positions determined us
            if self.x < 10.0 and self.y == 0.0:
                self.x+=0.5
                self.y == 0.0
                if self.x == 10.0:
                    self.theta = -math.pi/2

            elif self.x == 10.0 and self.y == 0.0:
                self.y -= 0.5
                self.theta = -math.pi/2
            elif self.x == 10.0 and self.y < 0 and self.y > -10.0:
                self.y -= 0.5
                self.x = 10.0
                if self.y == -10.0:
                    self.theta -=math.pi/2
            elif self.x > 0.0 and self.y == -10.0:
                self.x-=0.5
                self.y = -10.0
                if self.x == 0.0:
                    self.theta -=math.pi/2
            elif self.x == 0.0 and self.y < 0.0:
                self.y += 0.5
                self.x = 0.0
                if self.y == 0.0:
                    self.theta -=math.pi/2

##########################################################################
            self.rate.sleep()


    def linearvelocity(self):
        self.q = quaternion_from_euler(0.0, 0.0, self.theta)#function
        self.odom = Odometry()
        self.odom.pose.pose.position.x = self.x
        self.odom.pose.pose.position.y = self.y
        self.odom.pose.pose.position.z = 0.0
        self.odom.pose.pose.orientation.x = self.q[0]
        self.odom.pose.pose.orientation.y = self.q[1]
        self.odom.pose.pose.orientation.z = self.q[2]
        self.odom.pose.pose.orientation.w = self.q[3]
        self.odom.twist.twist.linear.x = self.vx
        self.odom.twist.twist.angular.z = self.yaw_rate
        self.odom.header.stamp = rospy.Time.now()
        self.odom.header.frame_id = "/map"
        self.odom.child_frame_id = "/base_link"


# Main function.
if __name__ == '__main__':
    # Initialize the node and name it.
    rospy.init_node('odom_example_node', anonymous = True)
    # Go to the main loop.
    ne = listenerNode()
    ne.run()
```

ROS – Odometry Example



- You can install RVIZ using:

  sudo apt-get install rviz

- To install and use tf library, you need to use:

  sudo apt-get install ros-melodic-tf

  sudo apt-get install ros-melodic-tf-conversions

References

http://wiki.ros.org/melodic/Installation/Ubuntu

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/ECE5463_ROSTutorialLecture1.pdf

Joseph, L. Robot Operating System (ROS) for Absolute Beginners, 2018

https://www.youtube.com/watch?v=N6K2LWG2kRI&t=83s

http://wiki.ros.org/rviz

http://wiki.ros.org/tf