Homework 2
AA275: Navigation for Autonomous Systems

Alexandros Tzikas
alextzik@stanford.edu

November 5, 2021

1 Problem 1

In this homework, the goal is to build ROS publisher and subscriber nodes in order to localize a moving vehicle using stereo camera images from the ORB-SLAM2 package. In addition, a Kalman filter will be implemented in order to provide a better estimate of the system's state at each time-step.

1.1

ROS was installed using the "minimum hassle" method 1. VirtualBox was downloaded at first. Then, the ROS Melodic Morenia VirtualBox image was downloaded and opened with VirtualBox. The output of the command printenv | grep ROS can be seen in Figure 1.

1.2

The installation of the necessary libraries can be seen in Figure 2. The creation and building of a catkin workspace can be seen in Figure 3. The various spaces created, along with their directories are shown. We can observe a source space, a build space and a devel space among others. The source space is where catkin will look for packages when building. The build space is where cmake is invoked. The devel space is where catkin generates binaries and runtime libraries that are executable before installation. Whether the space exists, is missing or unused is also shown. The workspace shown is where all files are located in. We also get a message that there are no packages in the source space to build. Finally, a summary of the build of packages is shown with the runtime and the successful package builds.

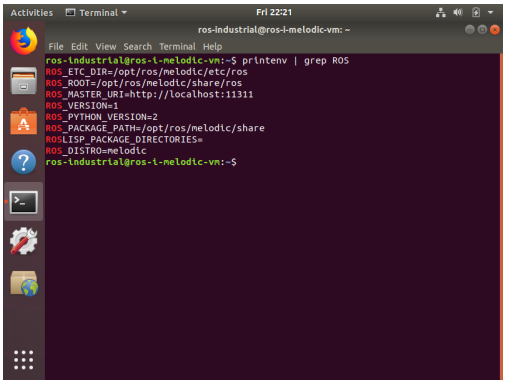
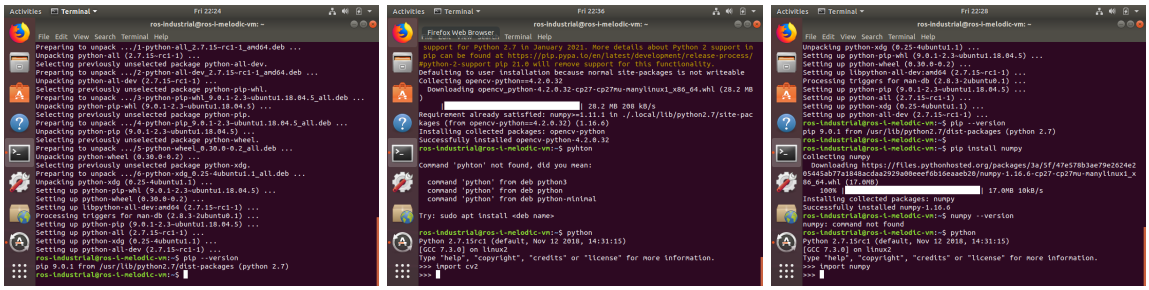


Figure 1: Output of printenv | grep ROS.



(a) pip install. (b) opencv install. (c) numpy install.

Figure 2: Installation of necessary libraries.

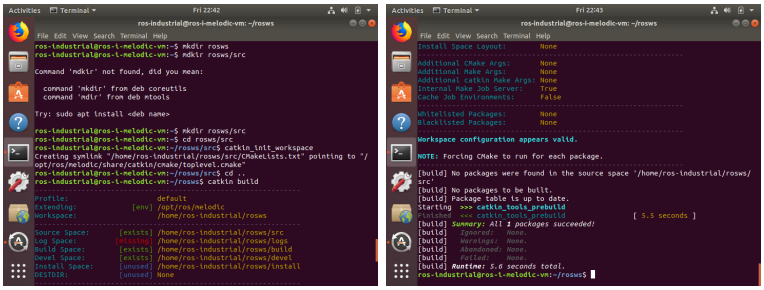


Figure 3: Terminal after catkin build command.

1.3

After running roscore, the contents of the master.log are shown in Figure 4. In this file, we can observe messages of severity level INFO, as well as the time they were written. We see the initialization of the master node and the start of the XML-RPC server. One publisher (PUB) and one subscriber (SUB) node are also seen running using topics /rosout_agg and /rosout respectively.

1.4

The 2011-09-026-drive-0002 dataset was downloaded. All images have the same dimensions and left images are in folder image-02, while right images are in folder image-03. The code for img_pub.py is included below. The output of the rqt is shown in Figure 5.

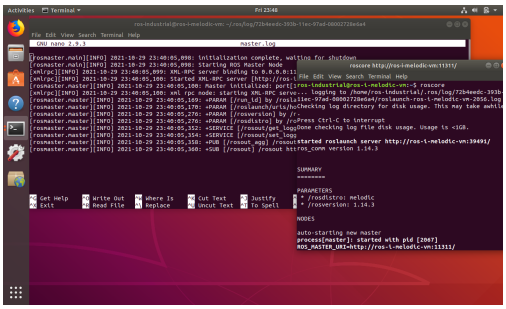


Figure 4: Output of master.log.

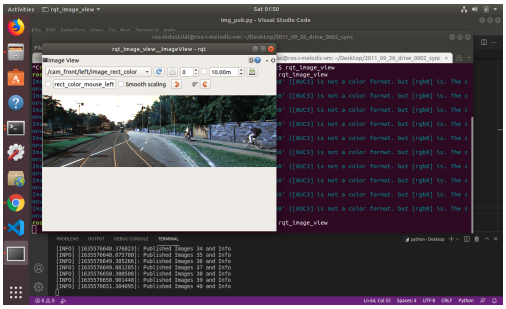


Figure 5: Output of rqt.

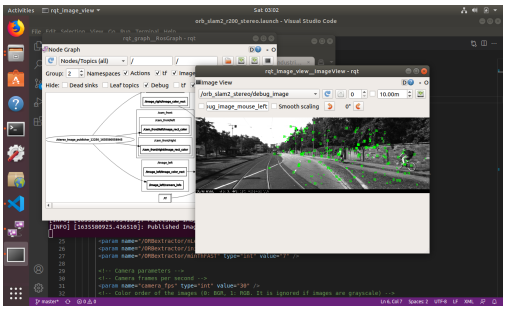


Figure 6: Output of rqt with ORB key points.

1.5

The ORB-SLAM2 ROS node was downloaded and built. The file orb_slam2_r200_stereo.launch file was modified accordingly and the rqt output with ORB key points is shown in Figure 6.

1.6

The following parameters were modified:

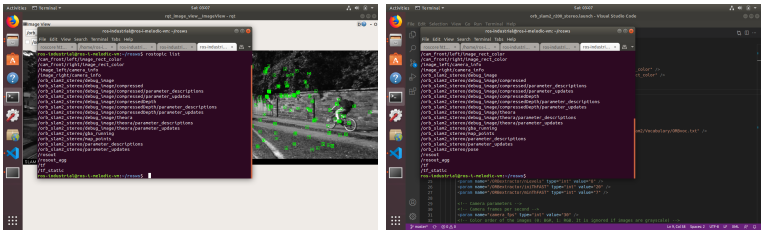
- "publish-pose" was replaced to false: No pose published in rostopic list. The change can be seen in Figure 7.
- nFeatures from 1200 to 500: Less features appear on output, as shown in Figure 8.
- ThDepth from 40 to 5: more features appear on the left side and the light post is recognized, as seen in Figure 9.
- nLevels from 8 to 1: The features appear to be more spread out, more features appear towards the front, less features towards the back, as shown in Figure 10.
- "publish-pointcloud" to FALSE: map-points disappears from rostopic list, as shown in Figure 11.

1.7

The completed code for pose_sub.py is included below.

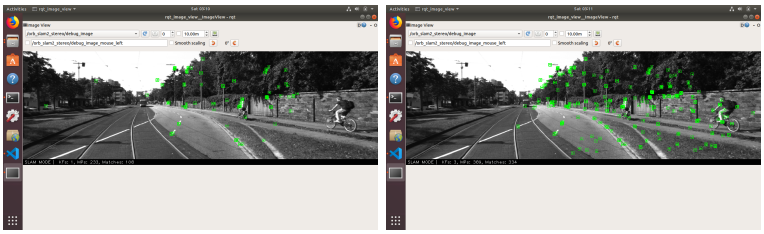
To observe the performance of the Kalman filter, 4 different scenarios were examined:

- No noise has been added to the ORB-SLAM2 output measurements. The results are shown in Figure 12a. It is evident that the code works well, since the path drawn resembles the vehicle trajectory as witnessed in the KITTI website. Since there is no noise, the state is completely observed. The ORB-SLAM2 output completely coincides with the Kalman filter output.



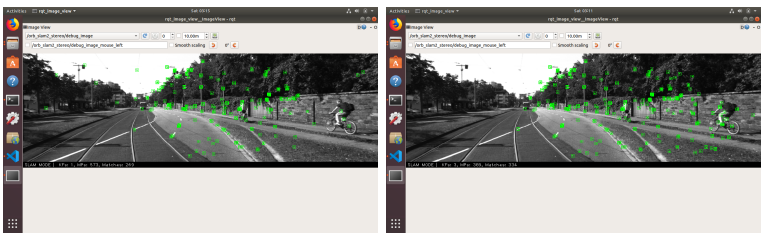
(a) Pose publish to FALSE. (b) Pose publish to TRUE.

Figure 7: Pose publish change.



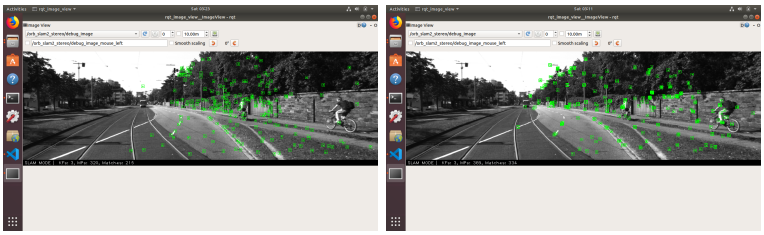
(a) nFeatures to 500. (b) nFeatures to 1200.

Figure 8: nFeatures change.



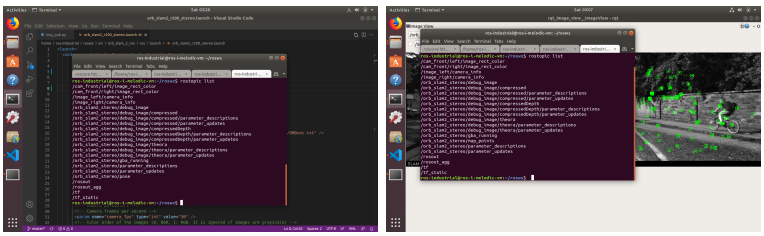
(a) ThDepth = 5. (b) ThDepth = 40.

Figure 9: ThDepth change.



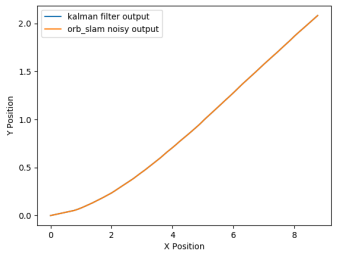
(a) nLevels to 1. (b) nLevels to 8.

Figure 10: nLevels change.

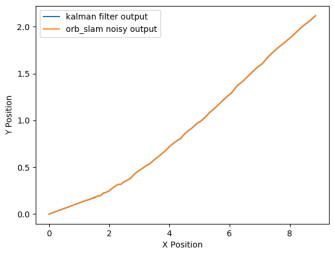


(a) Point Cloud to FALSE. (b) Point Cloud to TRUE.

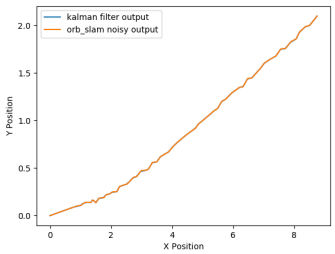
Figure 11: Point Cloud change.



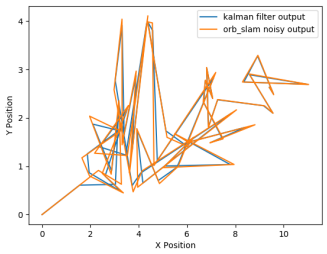
(a) No noise.



(b) Uniform noise in $[0, 0.02]$.



(c) Gaussian noise scaled by 0.02.



(d) Exponential noise with $\lambda = 1$.

Figure 12: Pose publish change.

- Uniform noise between $[0, 0.02]$ is added, as shown in Figure 12b. This choice was made, due to the small range of position values for the vehicle. We observe that again the Kalman filter output is stable and should be able to track the position quite well.
- Gaussian noise with mean 0 and covariance 1, but scaled by a factor of 0.02, is added, as shown in Figure 12c. Again, the Kalman filter output is stable and, since the trajectory noise is small, we observe that it is able to track the true trajectory quite well.
- Exponential distribution noise with $\lambda = 1$ was chosen, as shown in Figure 12d. In this case, the noise is far too great for the Kalman filter to follow the true trajectory and it instead follows the noisy measurement.

Overall, in order to further improve the estimation accuracy we would need a more accurate initial motion and sensing covariance, rather than the identity matrix. Also, a more thorough dynamics model could be of use. In addition, perhaps the use of relevant position might hinder the filter's performance.

2 Problem 2

This week, I dedicated time to play soccer. I really enjoyed playing in one of Stanford's great fields. I also spent time with my friends, drinking coffee and going to San Francisco.

3 Code Implementation

This code can also be found at `https://github.com/alextzik/navigation_autonomous_systems/tree/main/ORB-SLAM2-KalmanFilter-ROS`.

3.1 img_pub.py

```
#!/usr/bin/env python
from __future__ import print_function

import sys, os
import rospy
import cv2
from std_msgs.msg import String, Header
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge, CvBridgeError
import numpy as np

############################################################
# CONFIG AND GLOBAL VARIABLES
############################################################

# TODO: Populate the config dictionary with any
# configuration parameters that you need
config = {
    'topic_name_img_left':
    "/cam_front/left/image_rect_color",
    'topic_name_info_left': "/image_left/camera_info",
    'topic_name_img_right':
    "/cam_front/right/image_rect_color",
    'topic_name_info_right': "image_right/camera_info",
    'node_name': "stereo_image_publisher",
    'pub_rate': 2,   # Hz
    'data_dir': "/home/ros-industrial/Desktop/2011_09_26_dri
    ve_0002_sync",
    'img_dims': (512, 1392)
}

############################################################
# DATASET
############################################################

class ImgDataset:
    def __init__(self, config):
        self.data_dir = config['data_dir']
        self.files =
        sorted(os.listdir(self.data_dir+'/image_02/da
        ta/'))
        self.count = 0
```

```
        self.bridge = CvBridge()
        self.img_dims = config['img_dims']

    def build_camera_images(self):
        if self.count >= len(self.files):
            return None
        else:
            # TODO: Read stereo camera images from
            the dataset and
            # convert to ROS messages. Messages for
            both left and right
            # camera images must be returned.
            lm = cv2.imread(self.data_dir +
            "/image_02/data/" +
            self.files[self.count])
            rm = cv2.imread(self.data_dir +
            "/image_03/data/" +
            self.files[self.count])
            # print(self.data_dir + "/image02/data/"
            +
            self.files[0])


            img = Image()
            lmsg = Image()
            lmsg.height = config["img_dims"][0]
            lmsg.width = config["img_dims"][1]
            lmsg.encoding = "rgb8"
            lmsg = self.bridge.cv2_to_imgmsg(lm,
            "rgb8")

            rmsg = Image()
            rmsg.height = config["img_dims"][0]
            rmsg.width = config["img_dims"][1]
            rmsg.encoding = "rgb8"
            rmsg = self.bridge.cv2_to_imgmsg(rm,
            "rgb8")


            self.count += 1
            return lmsg, rmsg

    def build_camera_info(self):
        # TODO: Build and return camera calibration
        messages
        # for both left and right cameras
        lcamera_info = CameraInfo()
        lcamera_info.header = Header()
        lcamera_info.header.stamp = rospy.Time.now()
        lcamera_info.width = config["img_dims"][1]
        lcamera_info.height = config["img_dims"][0]
        lcamera_info.K = [9.597910e+02, 0.000000e+00,
        6.960217e+02, 0.000000e+00, 9.569251e+02,
        2.241806e+02, 0.000000e+00, 0.000000e+00,
```

```
        1.000000e+00]
        lcamera_info.R = [9.998817e-01, 1.511453e-02,
        -2.841595e-03, -1.511724e-02, 9.998853e-01,
        -9.338510e-04, 2.827154e-03, 9.766976e-04,
        9.999955e-01]
        lcamera_info.P = [7.215377e+02, 0.000000e+00,
        6.095593e+02, 4.485728e+01, 0.000000e+00,
        7.215377e+02, 1.728540e+02, 2.163791e-01,
        0.000000e+00, 0.000000e+00, 1.000000e+00,
        2.745884e-03]
        lcamera_info.D = [-3.691481e-01, 1.968681e-01,
        1.353473e-03, 5.677587e-04, -6.770705e-02]
        lcamera_info.distortion_model = "plumb_bob"

        rcamera_info = CameraInfo()
        rcamera_info.header.stamp = rospy.Time.now()
        rcamera_info.header = Header()
        rcamera_info.width = config["img_dims"][1]
        rcamera_info.height = config["img_dims"][0]
        rcamera_info.K = [9.597910e+02, 0.000000e+00,
        6.960217e+02, 0.000000e+00, 9.569251e+02,
        2.241806e+02, 0.000000e+00, 0.000000e+00,
        1.000000e+00]
        rcamera_info.R = [9.998817e-01, 1.511453e-02,
        -2.841595e-03, -1.511724e-02, 9.998853e-01,
        -9.338510e-04, 2.827154e-03, 9.766976e-04,
        9.999955e-01]
        rcamera_info.P = [7.215377e+02, 0.000000e+00,
        6.095593e+02, 4.485728e+01, 0.000000e+00,
        7.215377e+02, 1.728540e+02, 2.163791e-01,
        0.000000e+00, 0.000000e+00, 1.000000e+00,
        2.745884e-03]
        rcamera_info.D = [-3.691481e-01,
        1.968681e-01,
        1.353473e-03, 5.677587e-04, -6.770705e-02]
        rcamera_info.distortion_model = "plumb_bob"

        return lcamera_info, rcamera_info


############################################################
# ROS PUBLISHER
############################################################

def publish(config):
    lpub = rospy.Publisher(config['topic_name_img_lef
    t'],
    Image, queue_size=10)
    lpub_info =
    rospy.Publisher(config['topic_name_info_left
    '], CameraInfo, queue_size=10)
    rpub = rospy.Publisher(config['topic_name_img_rig
    ht'],
    Image, queue_size=10)
```

```
    rpub_info =
    rospy.Publisher(config['topic_name_info_righ
    t'], CameraInfo, queue_size=10)

    rospy.init_node(config['node_name'],
    anonymous=True)
    rate = rospy.Rate(config['pub_rate'])

    dataset = ImgDataset(config)

    while not rospy.is_shutdown():
        try:
            lmsg, rmsg =
            dataset.build_camera_images()
            lmsg_info, rmsg_info =
            dataset.build_camera_info()
            lpub.publish(lmsg)
            rpub.publish(rmsg)
            lpub_info.publish(lmsg_info)
            rpub_info.publish(rmsg_info)
            rospy.loginfo("Published Images {id} and
            Info".format(id=dataset.count))
        except CvBridgeError as e:
            print(e)

        rate.sleep()

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

3.2 pose_sub.py

```
#!/usr/bin/env python
from __future__ import print_function

import sys, os
import rospy
from std_msgs.msg import String, Header
from geometry_msgs.msg import PoseStamped
from matplotlib import pyplot as plt
import numpy as np
import math
import tf

######################################################
####
# CONFIG AND GLOBAL VARIABLES
######################################################
####

# TODO: Populate the config dictionary with any
```

```
# configuration parameters that you need
config = {
    'topic_name': "/orb_slam2_stereo/pose",
    'node_name': "pose_subscriber",
    'sub_freq': 5,      # measurements to skip
    'pub_rate': 5,      # Hz
}

idx = 0
measurement = None
prev_meas_x = 0
dt = 1/10 # based on Hz from dataset

########################################################
####
# HELPER FUNCIONS
########################################################
####

# Converts rotation matrix to euler angles
def mat2euler(M):
    r11, r12, r13, r21, r22, r23, r31, r32, r33 =
    M.flat
    cy = math.sqrt(r33*r33 + r23*r23)
    if cy > 1e-4: # cos(y) not close to zero,
    standard form
        z = math.atan2(-r12,  r11) #
        atan2(cos(y)*sin(z),
        cos(y)*cos(z))
        y = math.atan2(r13,  cy) # atan2(sin(y), cy)
        x = math.atan2(-r23, r33) #
        atan2(cos(y)*sin(x),
        cos(x)*cos(y))
    else: # cos(y) (close to) zero, so x -> 0.0 (see
    above)
        # so r21 -> sin(z), r22 -> cos(z) and
        z = math.atan2(r21,  r22)
        y = math.atan2(r13,  cy) # atan2(sin(y), cy)
        x = 0.0
    return z, y, x

# Converts quaternions to rotation matrix
def quat2mat(q):
    assert len(q) == 4, "Not a valid quaternion"
    if np.linalg.norm(q) != 1.:
        q = q / np.linalg.norm(q)
    mat = np.zeros((3,3))
    mat[0, 0] = 1 - 2*q[2]**2 - 2*q[3]**2
    mat[0, 1] = 2*q[1]*q[2] - 2*q[3]*q[0]
    mat[0, 2] = 2*q[1]*q[3] + 2*q[2]*q[0]
    mat[1, 0] = 2*q[1]*q[2] + 2*q[3]*q[0]
    mat[1, 1] = 1 - 2*q[1]**2 - 2*q[3]**2
    mat[1, 2] = 2*q[2]*q[3] - 2*q[1]*q[0]
    mat[2, 0] = 2*q[1]*q[3] - 2*q[2]*q[0]
```

```
    mat[2, 1] = 2*q[2]*q[3] + 2*q[1]*q[0]
    mat[2, 2] = 1 - 2*q[1]**2 - 2*q[2]**2
    return mat

def quat2euler(q):
    return mat2euler(quat2mat(q))

def euler2vec(state):
    theta = state[3]
    phi = state[4]
    psi = state[5]

    # theta = roll (rotation around x)
    # phi = pitch (rotation around y)
    # psi = yaw (rotation around z)

    # assume x-body is forward and we are referencing
    to the original pose
    x = np.cos(psi)*np.cos(phi)
    y = np.sin(psi)*np.cos(phi)
    z = np.sin(phi)

    res = np.zeros((3,1))
    res[0,0] = x
    res[1,0] = y
    res[2,0] = z
    return res



######################################################################
# KALMAN FILTER
######################################################################

class KalmanFilter(object):

    def __init__(self, dim_x, dim_y):
        # TODO: modify depending on the tracked state
        # You may add additional methods to this
        class
        # for building ROS messages

        self.dim_x = dim_x          # state dims
        self.dim_y = dim_y          # measurement dims

        self.x = np.zeros((dim_x,1))
        # self.x = pose       # state
        self.P = np.eye(dim_x)              #
        covariance
        self.Q = np.eye(dim_x)              #
        propagation
        noise

        self.F = np.eye(dim_x)              # state
```

```
        transition matrix

        self.H = np.zeros((dim_y, dim_x))   #
        measurement
        matrix
        for i in range(dim_x-1):
            self.H[i,i] = 1

        self.R = np.eye(dim_y)              #
        measurement uncertainty

    def predict(self):
        # TODO: add predict step code
        direction = euler2vec(self.x)*dt
        self.F[0, 6] = direction[0,0]
        self.F[1, 6] = direction[1,0]
        self.F[2, 6] = direction[2,0]
        self.x = np.matmul(self.F,self.x)
        self.P = np.matmul(np.matmul(self.F, self.P),
        np.transpose(self.F)) + self.Q
        pass

    def update(self, y):
        # TODO: add update step code. y is a vector
        # containing all the measurements
        t = self.R + np.matmul(np.matmul(self.H,
        self.P),
        np.transpose(self.H))
        K = np.matmul(np.matmul(self.P,
        np.transpose(self.H)), np.linalg.inv(t))
        self.x = self.x + np.matmul(K,
        y-np.matmul(self.H,
        self.x))
        self.P =
        np.matmul((np.eye(self.dim_x)-np.matmul(K,sel
        f.H)),
        self.P)
        pass

kf = KalmanFilter(1, 1)

##########################################################################
# ROS SUBSCRIBER
##########################################################################

def callback(data):
    global idx, config, measurement
    rospy.loginfo(rospy.get_caller_id()+"
    "+str(idx))
    idx += 1
    # TODO: add code to read position and orientation
    from Pose
    # message, add noise and pass to measurement
    global variable
```

```
    x = data.pose.position.x
    y = data.pose.position.y
    z = data.pose.position.z
    q0 = data.pose.orientation.w
    q1 = data.pose.orientation.x
    q2 = data.pose.orientation.y
    q3 = data.pose.orientation.z

    q = np.array([q0, q1, q2, q3])
    mat = quat2euler(q)
    measurement = np.array([x, y, z, mat[2], mat[1],
    mat[0]])

    # Uniform noise between 0, 1
    # measurement = measurement +
    0.02*np.random.rand(6)

    # Gaussian with mean 0 and covariance 1
    #measurement = measurement +
    0.02*np.random.randn(6)

    # Exponential distribution
    measurement = measurement +
    np.random.exponential(1.0, 6)

    # reshape measurement for the update of the
    Kalman Filter
    measurement = measurement.reshape(-1,1)

    pass

kf_states = np.zeros((3,1))
orb_slam_states = np.zeros((3,1))


def subscribe(config):
    global idx, measurement, kf_states,
    orb_slam_states
    # TODO: add code for creating a publisher for
    output
    rospy.Subscriber(config['topic_name'],
    PoseStamped, callback)

    # measurement[6] = (measurement[0] -
    prev_meas_x)/dt
    # prev_meas_x = measurement[0]

    # TODO: replace with publisher loop which calls
    predict/update
    # in Kalman filter and publishes the estimated
    pose
    kalman = KalmanFilter(7,6)
```

```
pub = rospy.Publisher('/kalman_filter/pose',
PoseStamped, queue_size=10)
rospy.init_node(config['node_name'],
anonymous=True)
rate = rospy.Rate(config['pub_rate'])

while not rospy.is_shutdown():

    if measurement is None:
            pass
    else:
        # print(measurement)
        kalman.predict()
        kalman.update(measurement)
        message = PoseStamped()
        message.header = Header()
        message.pose.position.x = kalman.x[0,0]
        message.pose.position.y = kalman.x[1,0]
        message.pose.position.z = kalman.x[2,0]
        message.pose.orientation.w =
        kalman.x[3,0]
        message.pose.orientation.x =
        kalman.x[4,0]
        message.pose.orientation.y =
        kalman.x[5,0]
        message.pose.orientation.z =
        kalman.x[6,0]

        pub.publish(message)
        print(kalman.x[0])
        print(kalman.x[1])
        print(kalman.x[2])
        kf_states = np.append(kf_states,
        kalman.x[0:3], axis=1)
        orb_slam_states =
        np.append(orb_slam_states,
        measurement[0:3], axis=1)

    rate.sleep()

print(1)
print(kf_states)
ax = plt.axes()
ax.set_ylabel("Y Position")
ax.set_xlabel("X Position")
ax.plot(kf_states[0,:], kf_states[1,:],
label='kalman filter output')
ax.plot(orb_slam_states[0,:],
orb_slam_states[1,:], label='orb_slam noisy
output')
plt.legend()
plt.show()
```

```
if __name__ == '__main__':
    try:
        subscribe(config)

    except rospy.ROSInterruptException:
        pass
```