**IE 410 – INTRODUCTION TO ROBOTICS**

**Lab-6 report**

**Controlling Turtlesim Node using Python code, implementing P controller and implementing PID controller.**

**Team M410:**

201901011 – HIMANSHU DUDHATRA

201901024 – DHAVALSINH RAJ

201901100 – SHUBHAM PATEL

201901145 – GARGEY PATEL

* **Creating Catkin package**

First go to the ‘src’ folder of catkin workspace that we have created.

$ cd catkin\_ws/src

Now create a catkin package named linuxsquad\_pkgforlab4 by using the command given below.

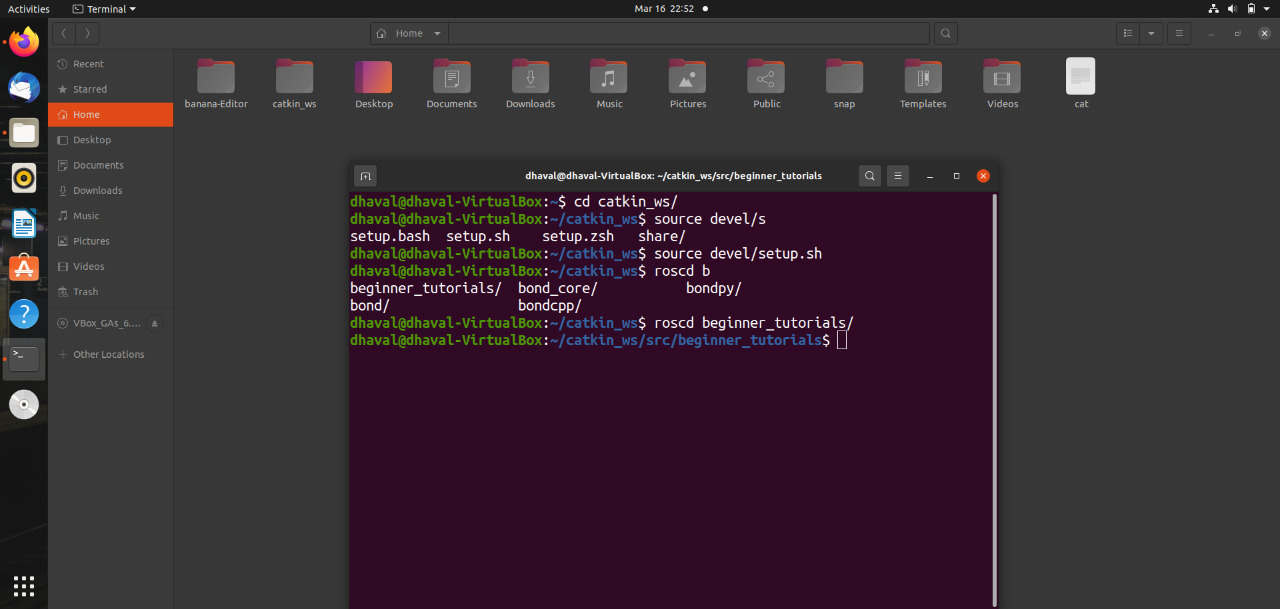
$ catkin\_create\_pkg beginner\_tuts std\_msgs rospy roscpp

Now we will execute catkin\_make.

$ catkin\_make

Now we need to source the generated setup file.

$ source devel/setup.bash



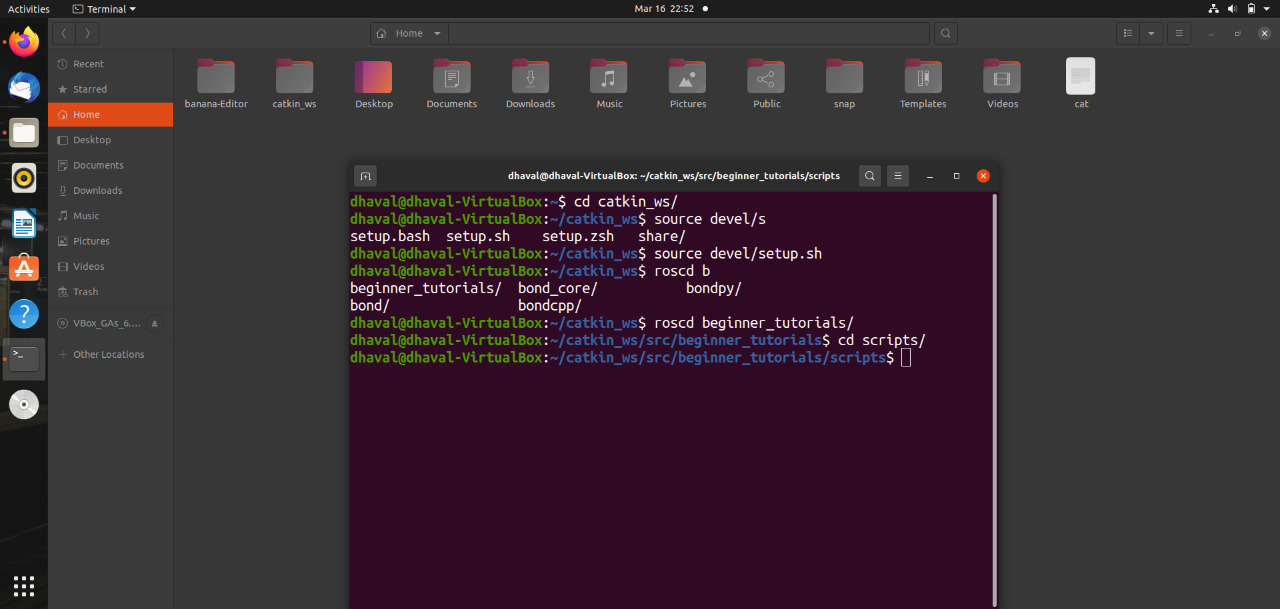
* **Turtlesim\_move.py and Turtlesim\_cleaner.py**

Now, we will change the directory to linuxsquad\_pkglab6 and then will create a scriptsdirectory in the linuxsquad\_pkglab6 directory. Follow these commands.

$ roscd linuxsquad\_pkgforlab6

$ mkdir scripts

$ cd scripts



* Turtlesim\_cleaner.py

#!/usr/bin/env python

import rospy

from geometry\_msgs.msg import Twist

from turtlesim.msg import Pose

import math

import time

from std\_srvs.srv import Empty

x=0

y=0

yaw=0

def poseCallback(pose\_message):

global x

global y, yaw

x= pose\_message.x

y= pose\_message.y

yaw = pose\_message.theta

#print "pose callback"

#print ('x = {}'.format(pose\_message.x)) #new in python 3

#print ('y = %f' %pose\_message.y) #used in python 2

#print ('yaw = {}'.format(pose\_message.theta)) #new in python 3

def move(speed, distance, is\_forward):

#declare a Twist message to send velocity commands

velocity\_message = Twist()

#get current location

global x, y

x0=x

y0=y

if (is\_forward):

velocity\_message.linear.x =abs(speed)

else:

velocity\_message.linear.x =-abs(speed)

distance\_moved = 0.0

loop\_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)

cmd\_vel\_topic='/turtle1/cmd\_vel'

velocity\_publisher = rospy.Publisher(cmd\_vel\_topic, Twist, queue\_size=10)

while True :

rospy.loginfo("Turtlesim moves forwards")

velocity\_publisher.publish(velocity\_message)

loop\_rate.sleep()

#rospy.Duration(1.0)

distance\_moved = abs(0.5 \* math.sqrt(((x-x0) \*\* 2) + ((y-y0) \*\* 2)))

print (distance\_moved);

if not (distance\_moved<distance):

rospy.loginfo("reached")

break

#finally, stop the robot when the distance is moved

velocity\_message.linear.x =0

velocity\_publisher.publish(velocity\_message)

def rotate (angular\_speed\_degree, relative\_angle\_degree, clockwise):

global yaw

velocity\_message = Twist()

velocity\_message.linear.x=0

velocity\_message.linear.y=0

velocity\_message.linear.z=0

velocity\_message.angular.x=0

velocity\_message.angular.y=0

velocity\_message.angular.z=0

#get current location

theta0=yaw

angular\_speed=math.radians(abs(angular\_speed\_degree))

if (clockwise):

velocity\_message.angular.z =-abs(angular\_speed)

else:

velocity\_message.angular.z =abs(angular\_speed)

angle\_moved = 0.0

loop\_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)

cmd\_vel\_topic='/turtle1/cmd\_vel'

velocity\_publisher = rospy.Publisher(cmd\_vel\_topic, Twist, queue\_size=10)

t0 = rospy.Time.now().to\_sec()

while True :

rospy.loginfo("Turtlesim rotates")

velocity\_publisher.publish(velocity\_message)

t1 = rospy.Time.now().to\_sec()

current\_angle\_degree = (t1-t0)\*angular\_speed\_degree

loop\_rate.sleep()

if (current\_angle\_degree>relative\_angle\_degree):

rospy.loginfo("reached")

break

#finally, stop the robot when the distance is moved

velocity\_message.angular.z =0

velocity\_publisher.publish(velocity\_message)

def go\_to\_goal(x\_goal, y\_goal):

global x

global y, yaw

velocity\_message = Twist()

cmd\_vel\_topic='/turtle1/cmd\_vel'

while (True):

K\_linear = 0.5

distance = abs(math.sqrt(((x\_goal-x) \*\* 2) + ((y\_goal-y) \*\* 2)))

linear\_speed = distance \* K\_linear

K\_angular = 4.0

desired\_angle\_goal = math.atan2(y\_goal-y, x\_goal-x)

angular\_speed = (desired\_angle\_goal-yaw)\*K\_angular

velocity\_message.linear.x = linear\_speed

velocity\_message.angular.z = angular\_speed

velocity\_publisher.publish(velocity\_message)

#print velocity\_message.linear.x

#print velocity\_message.angular.z

print ('x=', x, 'y=',y)

if (distance <0.01):

break

def setDesiredOrientation(desired\_angle\_radians):

relative\_angle\_radians = desired\_angle\_radians - yaw

if relative\_angle\_radians < 0:

clockwise = 1

else:

clockwise = 0

print (relative\_angle\_radians)

print( desired\_angle\_radians)

rotate(30 ,math.degrees(abs(relative\_angle\_radians)), clockwise)

def gridClean():

desired\_pose = Pose()

desired\_pose.x = 1

desired\_pose.y = 1

desired\_pose.theta = 0

moveGoal(desired\_pose, 0.01)

setDesiredOrientation(degrees2radians(desired\_pose.theta))

move(2.0, 9.0, True)

rotate(degrees2radians(20), degrees2radians(90), False)

move(2.0, 9.0, True)

rotate(degrees2radians(20), degrees2radians(90), False)

move(2.0, 1.0, True)

rotate(degrees2radians(20), degrees2radians(90), False)

move(2.0, 9.0, True)

rotate(degrees2radians(30), degrees2radians(90), True)

move(2.0, 1.0, True)

rotate(degrees2radians(30), degrees2radians(90), True)

move(2.0, 9.0, True)

pass

def spiralClean():

vel\_msg = Twist()

loop\_rate = rospy.Rate(1)

wk = 4

rk = 0

while((currentTurtlesimPose.x<10.5) and (currentTurtlesimPose.y<10.5)):

rk=rk+1

vel\_msg.linear.x =rk

vel\_msg.linear.y =0

vel\_msg.linear.z =0

vel\_msg.angular.x = 0

vel\_msg.angular.y = 0

vel\_msg.angular.z =wk

velocity\_publisher.publish(vel\_msg)

loop\_rate.sleep()

vel\_msg.linear.x = 0

vel\_msg.angular.z = 0

velocity\_publisher.publish(vel\_msg)

if \_\_name\_\_ == '\_\_main\_\_':

try:

rospy.init\_node('turtlesim\_motion\_pose', anonymous=True)

#declare velocity publisher

cmd\_vel\_topic='/turtle1/cmd\_vel'

velocity\_publisher = rospy.Publisher(cmd\_vel\_topic, Twist, queue\_size=10)

position\_topic = "/turtle1/pose"

pose\_subscriber = rospy.Subscriber(position\_topic, Pose, poseCallback)

time.sleep(2)

#move(1.0, 2.0, False)

#rotate(30, 90, True)

go\_to\_goal(1.0, 1.0)

#setDesiredOrientation(math.radians(90))

except rospy.ROSInterruptException:

rospy.loginfo("node terminated.")

* Turtlesim\_move.py

#!/usr/bin/env python

import rospy

from geometry\_msgs.msg import Twist

import math

import time

from std\_srvs.srv import Empty

def move(speed, distance, is\_forward):

#declare a Twist message to send velocity commands

velocity\_message = Twist()

#get current location

if (speed > 0.4):

print('speed must be lower than 0.4')

return

if (is\_forward):

velocity\_message.linear.x =abs(speed)

else:

velocity\_message.linear.x =-abs(speed)

distance\_moved = 0.0

loop\_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)

cmd\_vel\_topic='/cmd\_vel\_mux/input/teleop'

velocity\_publisher = rospy.Publisher(cmd\_vel\_topic, Twist, queue\_size=10)

t0 = rospy.Time.now().to\_sec()

while True :

rospy.loginfo("Turtlesim moves forwards")

velocity\_publisher.publish(velocity\_message)

loop\_rate.sleep()

t1 = rospy.Time.now().to\_sec()

#rospy.Duration(1.0)

distance\_moved = (t1-t0) \* speed

print(distance\_moved)

if not (distance\_moved<distance):

rospy.loginfo("reached")

break

#finally, stop the robot when the distance is moved

velocity\_message.linear.x =0

velocity\_publisher.publish(velocity\_message)

def rotate (angular\_speed\_degree, relative\_angle\_degree, clockwise):

velocity\_message = Twist()

velocity\_message.linear.x=0

velocity\_message.linear.y=0

velocity\_message.linear.z=0

velocity\_message.angular.x=0

velocity\_message.angular.y=0

velocity\_message.angular.z=0

angular\_speed=math.radians(abs(angular\_speed\_degree))

if (clockwise):

velocity\_message.angular.z =-abs(angular\_speed)

else:

velocity\_message.angular.z =abs(angular\_speed)

angle\_moved = 0.0

loop\_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)

cmd\_vel\_topic='/cmd\_vel\_mux/input/teleop'

velocity\_publisher = rospy.Publisher(cmd\_vel\_topic, Twist, queue\_size=10)

t0 = rospy.Time.now().to\_sec()

while True :

rospy.loginfo("Turtlesim rotates")

velocity\_publisher.publish(velocity\_message)

t1 = rospy.Time.now().to\_sec()

current\_angle\_degree = (t1-t0)\*angular\_speed\_degree

loop\_rate.sleep()

print('current\_angle\_degree: ',current\_angle\_degree)

if (current\_angle\_degree>relative\_angle\_degree):

rospy.loginfo("reached")

break

#finally, stop the robot when the distance is moved

velocity\_message.angular.z =0

velocity\_publisher.publish(velocity\_message)

def go\_to\_goal(x\_goal, y\_goal):

global x

global y, z, yaw

velocity\_message = Twist()

cmd\_vel\_topic='/turtle1/cmd\_vel'

while (True):

K\_linear = 0.5

distance = abs(math.sqrt(((x\_goal-x) \*\* 2) + ((y\_goal-y) \*\* 2)))

linear\_speed = distance \* K\_linear

K\_angular = 4.0

desired\_angle\_goal = math.atan2(y\_goal-y, x\_goal-x)

angular\_speed = (desired\_angle\_goal-yaw)\*K\_angular

velocity\_message.linear.x = linear\_speed

velocity\_message.angular.z = angular\_speed

velocity\_publisher.publish(velocity\_message)

print('x=', x, 'y=',y)

if (distance <0.01):

break

if \_\_name\_\_ == '\_\_main\_\_':

try:

rospy.init\_node('turtlesim\_motion\_pose', anonymous=True)

#move (0.3, 0.5 , False)

time.sleep(1.0)

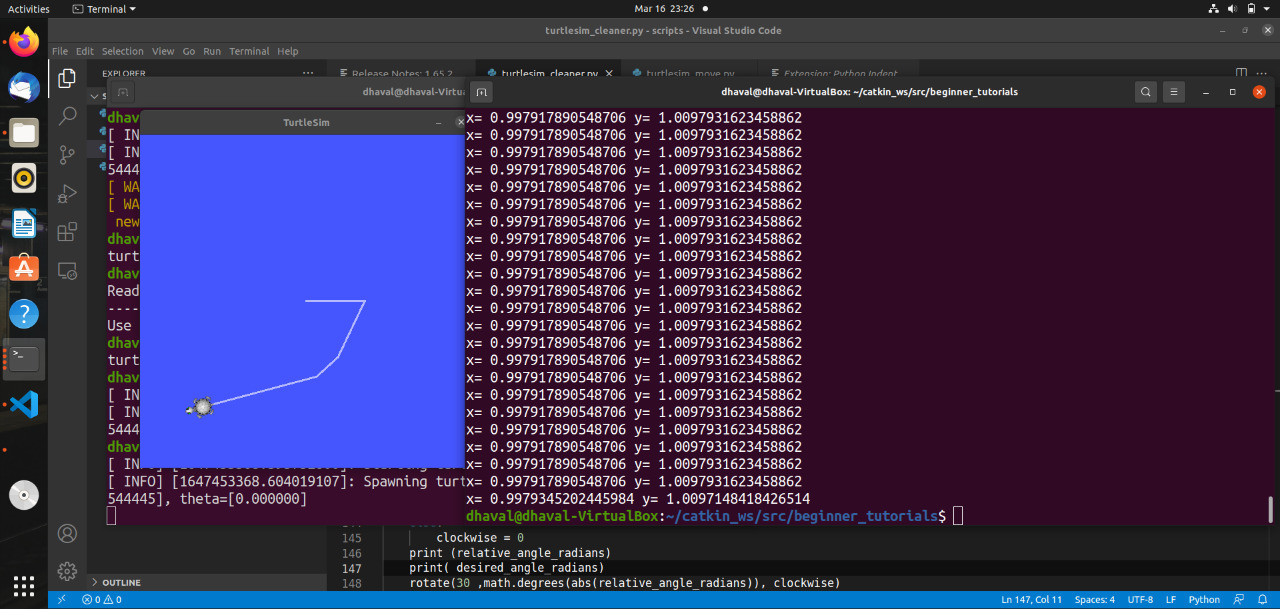
rotate (90, 90 , True)

except rospy.ROSInterruptException:

rospy.loginfo("node terminated.")

* Testing

Turtlesim\_cleaner.py



Turtlesim\_move.py

