```
#!/usr/bin/env python3

#Junhyung Park and Ryan Buckton
#turtlebot_controller.py
#This lab will integrate the on-board IMU with the Turtlebot3 controller to turn the robot 90 degrees left or right.
#last modified: 20 Feb 2023
#Finished all TODO and have the robot turn 90 degrees left or right
# 16 Feb 2023
# Working on callback_controller()

import rospy
from lab1.msg import MouseController
from geometry_msgs.msg import Twist
#TODO Import the squaternion library and Imu message used in ICE6.
from squaternion import Quaternion
from sensor_msgs.msg import Imu


class Controller:
    """Class that controls subsystems on Turtlebot3"""
    #TODO Add necessary class variables above init
    K_HDG = 0.1 # rotation controller constant
    HDG_TOL = 10 # heading tolerance +/- degrees
    MIN_ANG_Z = 0.5 # limit rad/s values sent to Turtlebot3
    MAX_ANG_Z = 1.5 # limit rad/s values sent to Turtlebot3

    def __init__(self):
        #TODO Add instance variables
        self.curr_yaw = 0
        self.goal_yaw = 0
        self.turning = False

        self.cmd = Twist()

        self.cmd.linear.x = 0.0
        self.cmd.linear.y = 0.0
        self.cmd.linear.z = 0.0
        self.cmd.angular.x = 0.0
        self.cmd.angular.y = 0.0
        self.cmd.angular.z = 0.0

        # TODO: create a timer that will call the callback_publish() function every .1 seconds (10 Hz)
        rospy.Timer(rospy.Duration(.1), self.callback_controller)

        #TODO A subscriber to the IMU topic of interest with a callback to the callback_imu() function
        rospy.Subscriber('imu', Imu, self.callback_imu)

        # self.rate = rospy.Rate(10)    # 10 Hz

        self.pub = rospy.Publisher('cmd_vel', Twist, queue_size = 1)
        rospy.Timer(rospy.Duration(.1), self.callback_controller)
```

```python
        rospy.Subscriber('mouse_info', MouseController, self.callback_mouseControl)

        self.ctrl_c = False
        rospy.on_shutdown(self.shutdownhook)

    #TODO Add convert_yaw from ICE6
    # The IMU provides yaw from -180 to 180. This function
    # converts the yaw (in degrees) to 0 to 360
    def convert_yaw (self, yaw):
        return 360 + yaw if yaw < 0 else yaw

    #TODO Add the callback_imu() function from ICE6, removing print statements and setting the instance variable, self.curr_yaw
    def callback_imu(self, imu):
        if not self.ctrl_c:
            # create a quaternion using the x, y, z, and w values
            # from the correct imu message

            # w, x, y, and z is whithin orientation of imu
            imu_q = Quaternion(imu.orientation.w, imu.orientation.x, imu.orientation.y, imu.orientation.z)

            # convert the quaternion to euler in degrees
            imu_e = imu_q.to_euler(degrees=True)

            # get the yaw component of the euler
            yaw = imu_e[2]

            # convert yaw from -180 to 180 to 0 to 360
            self.curr_yaw = self.convert_yaw(yaw)

    #TODO turns the robot 90 degrees in the direction inputed by the user (left or right)
    def callback_controller(self, event):
        # local variables do not need the self
        yaw_err = 0
        ang_z = 0

        # not turning, so get user input
        if not self.turning:
            #read from user and set value to instance variable, self.goal_yaw
            keyboard_cmd = input("Input l or r to turn 90 deg, input ll or rr to turn 180 deg: ")

            #TODO check input and determine goal yaw
            if keyboard_cmd == 'l':
                #set goal yaw to curr yaw plus/minus 90
                self.goal_yaw = self.curr_yaw + 90
                #turning equals True
                self.turning = True
            elif keyboard_cmd == 'r':
                #set goal yaw to curr yaw plus/minus 90
                self.goal_yaw = self.curr_yaw - 90
                #turning equals True
                self.turning = True
            elif keyboard_cmd == 'll':
                #set goal yaw to curr yaw plus/minus 90
                self.goal_yaw = self.curr_yaw + 180
                #turning equals True
                self.turning = True
            elif keyboard_cmd == 'rr':
                #set goal yaw to curr yaw plus/minus 90
                self.goal_yaw = self.curr_yaw - 180
                #turning equals True
                self.turning = True
            else:
                # print error and tell user valid inputs
                print("Error: Please input valid inputs.")

            # check bounds
            #TODO if goal_yaw is less than 0 then add 360 else if goal_yaw is greater
than 360 then subtract 360
            if self.goal_yaw < 0:
                self.goal_yaw += 360
            elif self.goal_yaw > 360:
                self.goal_yaw -= 360

        # turn until goal is reached
        elif self.turning:
            yaw_err = self.goal_yaw - self.curr_yaw

            # determine if robot should turn clockwise or counterclockwise
            if yaw_err > 180:
                yaw_err = yaw_err - 360
            elif yaw_err < -180:
                yaw_err = yaw_err + 360

            # proportional controller that turns the robot until goal
            # yaw is reached
            ang_z = self.K_HDG * yaw_err

            #TODO Add negative test
            if abs(ang_z) < self.MIN_ANG_Z and ang_z > 0:
                ang_z = self.MIN_ANG_Z
            elif abs(ang_z) < self.MIN_ANG_Z and ang_z < 0:
                ang_z = -self.MIN_ANG_Z
            elif ang_z > self.MAX_ANG_Z:
                ang_z = self.MAX_ANG_Z
            elif abs(ang_z) > self.MAX_ANG_Z:
                ang_z = -self.MAX_ANG_Z


            # check goal orientation
            if abs(yaw_err) < self.HDG_TOL:
                self.turning = False
                ang_z = 0

        # set twist message and publish
        self.cmd.linear.x = 0
        self.cmd.angular.z = ang_z
        # TODO Publish cmd
```

```python
        self.pub.publish(self.cmd)

    def callback_mouseControl(self, mouseInfo):
        #TODO Scale xPos from -1 to 1 to -.5 to .5
        scaled_xPos = -(mouseInfo.xPos)/2

        #TODO Set angular z in Twist message to the scaled value in the appropriate direction
        self.cmd.angular.z = scaled_xPos

        #TODO Scale yPos from -1 to 1 to -.5 to .5
        scaled_yPos = -(mouseInfo.yPos)/2

        #TODO Set linear x in Twist message to the scaled value in the appropriate direction
        self.cmd.linear.x = scaled_yPos

        #TODO 8 publish the Twist message
        self.pub.publish(self.cmd)


    def shutdownhook(self):
        print("Controller exiting. Halting robot.")
        self.ctrl_c = True
        #TODO 9 force the linear x and angular z commands to 0 before halting
        self.cmd.linear.x = 0
        self.cmd.angular.z = 0

        self.pub.publish(self.cmd)




if __name__ == '__main__':
    rospy.init_node('controller')
    c = Controller()
    rospy.spin()
```