EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 4

SOLUTION

Task 1: Robomaster Setup [20 MARKS]

Just follow the steps in the manual.

Task 2: Understanding Robomaster [20 MARKS]

For tasks 2.1, 2.2 , 2.3, follow steps in the manual.

For task 2.4, The python script **teleop.py**:

```
# -*-coding:utf-8-*-
# Copyright (c) 2020 DJI.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License in the file LICENSE.txt or at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.


from robomaster import robot
import keyboard
import time

def sub_position_handler(position_info):
    x, y, _ = position_info
    print("chassis position: x:{0}, y:{1}".format(x, y))

def sub_attitude_info_handler(attitude_info):
    yaw, _, _ = attitude_info
    print("chassis attitude: yaw:{0} ".format(yaw))



if __name__ == '__main__':
    ep_robot = robot.Robot()
    ep_robot.initialize(conn_type="ap")

    ep_chassis = ep_robot.chassis

    x_val = 0.5
    y_val = 0.5
    z_val = 30

    ep_chassis.sub_position(freq=10, callback=sub_position_handler)
    ep_chassis.sub_attitude(freq=10, callback=sub_attitude_info_handler)

    while True:
      if keyboard.read_key() == 'w':
         ep_chassis.drive_speed(x=x_val, y=0, z=0, timeout=5)
      #time.sleep(1)
      elif keyboard.read_key() == 'a':
         ep_chassis.drive_speed(x=0, y=0, z=z_val, timeout=5)
      #time.sleep(1)
      elif keyboard.read_key() == 's':
         ep_chassis.drive_speed(x=-x_val, y=0, z=0, timeout=5)
      #time.sleep(1)
      elif keyboard.read_key() == 'd':
         ep_chassis.drive_speed(x=0, y=0, z=z_val, timeout=5)
      #time.sleep(1)
      else:
         ep_chassis.move(x=0, y=0, z=0, xy_speed=0).wait_for_completed()



    ep_robot.close()
```

Task 3: Go-to-Goal-with-PID [15 MARKS]

The python script **pid.py**:

```
# -*-coding:utf-8-*-
# Copyright (c) 2020 DJI.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License in the file LICENSE.txt or at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.


from robomaster import robot
import time
import numpy as np
from math import sin,cos,degrees,radians,atan2,sqrt



def map(ang):
    if ang >= 0 and ang<=180:
        return radians(ang)
    if ang >=-180 and ang <0:
        return radians(360 - abs(ang))



class PIDRobot:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.current_angle = None
        self.ep_robot = robot.Robot()
        self.ep_robot.initialize(conn_type="ap")
        self.ep_chassis = self.ep_robot.chassis
        self.ep_chassis.sub_position(freq=10, callback=self.sub_position_handler)
        self.ep_chassis.sub_attitude(freq=10,
callback=self.sub_attitude_info_handler)

    def sub_position_handler(self,position_info):
        self.x, self.y,_ = position_info
        # print("chassis position: x:{0}, y:{1} theta:{2}".format(self.x,
self.y,self.current_angle))

    def sub_attitude_info_handler(self,attitude_info):
        self.current_angle, _, _ = attitude_info
        # print("chassis attitude: yaw:{0} ".format(self.current_angle))

    def gotogoal(self):

        dx = 0.1    #threshold (min distance)
        goal = [1,0]
        prev_err = 0
        cumulative_err = 0
        dt = 0.01   #best=0.01
        lin_vel = 0.3

          #print("GOAL: goal[0]:{0}, goal[1]:{1} ".format(goal[0],goal[1]))
        while True:

            current_loc = [round(self.x,2),round(self.y,2)]

            print("current location and goal  ",current_loc,goal)
            xdiff = goal[0]-current_loc[0]
            ydiff = goal[1]-current_loc[1]
            dist = round(sqrt((xdiff*xdiff) + (ydiff*ydiff)),2)

            if dist < dx:
                self.ep_chassis.drive_speed(x=0, y=0, z=0, timeout=5)
                time.sleep(0.5)
                break

            desired_angle = atan2(ydiff,xdiff)
            ang_err = desired_angle - radians(self.current_angle)
            ang_vel = self.pid(ang_err,prev_err,cumulative_err,dt)

            print("ANGLES ... desired,current ",desired_angle,
self.current_angle)
            print("ang-vel,ang-err, distance",ang_vel,ang_err,dist)
            print("heading control + distance control")

            self.ep_chassis.drive_speed(x=lin_vel, y=0, z=degrees(ang_vel),
timeout=5)
            time.sleep(dt)

            prev_err = ang_err
            cumulative_err += prev_err

    def run(self):
        time.sleep(1)
        self.gotogoal()
        self.ep_robot.close()

    def pid(self,ang_err,prev_err,cumulative_err,dt):
        kp = 2
        ki = 0.2
        kd = 0

        res =  kp*ang_err + kd*((ang_err-prev_err)/dt) +
ki*dt*(ang_err+cumulative_err)
        return res


if __name__ == '__main__':
    controller = PIDRobot()
    controller.run()
```

Task 4: Odometry and IMU Sensor Data[15 MARKS]

The python script **odom_imu_pub.py**:

```
#!/usr/bin/env python3
import robomaster
from robomaster import robot
import time
import rospy
from math import degrees,radians
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu
from geometry_msgs.msg import Quaternion, TwistWithCovarianceStamped
import numpy as np
def get_quaternion_from_euler(roll, pitch, yaw):
  """
  Convert an Euler angle to a quaternion.

  Input
    :param roll: The roll (rotation around x-axis) angle in radians.
    :param pitch: The pitch (rotation around y-axis) angle in radians.
    :param yaw: The yaw (rotation around z-axis) angle in radians.

  Output
    :return qx, qy, qz, qw: The orientation in quaternion [x,y,z,w] format
  """
  qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) *
np.sin(pitch/2) * np.sin(yaw/2)
  qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) *
np.cos(pitch/2) * np.sin(yaw/2)
  qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) *
np.sin(pitch/2) * np.cos(yaw/2)
  qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) *
np.sin(pitch/2) * np.sin(yaw/2)

  return [qx, qy, qz, qw]



class OdomImuPublisher:
      def __init__(self,ep_chassis):
        rospy.init_node('odometry_imu_publisher', anonymous=True)
        ep_chassis.sub_imu(freq=10,callback=self.sub_imu_info_handler)

ep_chassis.sub_attitude(freq=10,callback=self.sub_attitude_info_handler)

ep_chassis.sub_position(freq=10,callback=self.sub_position_info_handler)
        time.sleep(3)

        self.odomPub = rospy.Publisher('odom', Odometry, queue_size=10)
        self.imuPub = rospy.Publisher('imu', Imu, queue_size=10)
        self.rate = rospy.Rate(10) # 10 Hz
        self.prev_x = 0
        self.prev_y = 0
        self.prev_theta = 0
        self.prev_time = rospy.get_time()

    def publish_odometry(self):
        odom = Odometry()
        odom.header.stamp = rospy.get_rostime()
        odom.header.frame_id = 'odom'
        odom.child_frame_id = 'base_link'

        odom.pose.pose.position.x = self.x
        odom.pose.pose.position.y = self.y
        q = get_quaternion_from_euler(radians(self.roll),
radians(self.pitch), radians(self.theta))
        odom.pose.pose.orientation.x = q[0]
        odom.pose.pose.orientation.y = q[1]
        odom.pose.pose.orientation.z = q[2]
        odom.pose.pose.orientation.w = q[3]

        timediff = (rospy.get_time() - self.prev_time)
        odom.twist.twist.linear.x = (self.x - self.prev_x) / timediff
        odom.twist.twist.linear.y= (self.y - self.prev_y) / timediff
        odom.twist.twist.angular.z = (self.theta - self.prev_theta) /
timediff

        self.prev_time = rospy.get_time()
        self.prev_x = self.x
        self.prev_y = self.y
        self.prev_theta = self.theta
        odom.pose.covariance = [0.1, 0, 0, 0, 0, 0,
        0, 0.1, 0, 0, 0, 0,
        0, 0, 0.1, 0, 0, 0,
        0, 0, 0, 0.1, 0, 0,
        0, 0, 0, 0, 0.1, 0,
        0, 0, 0, 0, 0, 0.1] #example covarience matrix

        self.odomPub.publish(odom)
        self.rate.sleep()

    def publish_imu(self):
        imu = Imu()
        imu.header.frame_id = "imu"
        imu.header.stamp = rospy.get_rostime()

        q = get_quaternion_from_euler(radians(self.roll),
radians(self.pitch), radians(self.theta))
        imu.orientation.x = q[0]
        imu.orientation.y = q[1]
        imu.orientation.z = q[2]
        imu.orientation.w = q[3]
        imu.linear_acceleration.x = self.acc_x
        imu.linear_acceleration.y = self.acc_y
        imu.linear_acceleration.z = self.acc_z
        imu.angular_velocity.x = self.ang_x
        imu.angular_velocity.y = self.ang_y
        imu.angular_velocity.z = self.ang_z

        imu.orientation_covariance = [0.1,0,0,0,0.1,0,0,0,0.1]
        imu.angular_velocity_covariance = [0.1,0,0,0,0.1,0,0,0,0.1]
        imu.linear_acceleration_covariance = [0.1,0,0,0,0.1,0,0,0,0.1]

        self.imuPub.publish(imu)
        self.rate.sleep()

    def sub_position_info_handler(self,position_info):
        self.x,self.y,self.z = position_info
    def sub_attitude_info_handler(self,attitude_info):
        self.theta,self.pitch,self.roll = attitude_info

    def sub_imu_info_handler(self,imu_info):
        self.acc_x,self.acc_y,self.acc_z,self.ang_x,self.ang_y,self.ang_z = imu_info

    def spin(self):
        while not rospy.is_shutdown():
            self.publish_odometry()
            self.publish_imu()

if __name__ == '__main__':
    try:
        ep_robot = robot.Robot()
        ep_robot.initialize(conn_type="ap")
        ep_chassis = ep_robot.chassis

        odometry_imu_pub = OdomImuPublisher(ep_chassis)
        odometry_imu_pub.spin()
        ep_chassis.unsub_imu()
        ep_chassis.unsub_attitude()
        ep_chassis.unsub_position()
        ep_robot.close()
    except rospy.ROSInterruptException:
        pass
```