Code for part 2

Jarl Benjamin Andersen

13.11.2023

1 Python Code for ROS Odometry and Goal Subscriber

Below is an example of a Python script used in a ROS (Robot Operating System) environment for subscribing to odometry and goal point data:

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

import rospy
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Point

# Global variables to store the position and orientation
position_x = 0.0
position_y = 0.0
position_z = 0.0
orientation_w = 0.0

# Global variables to store the goal point
goal_x = 0.0
goal_y = 0.0
goal_z = 0.0

def odom_callback(data):
    # Callback function to handle received odometry messages
    global position_x, position_y, position_z, orientation_w

    # Store the position and orientation
    position_x = data.pose.pose.position.x
    position_y = data.pose.pose.position.y
    position_z = data.pose.pose.position.z
    orientation_w = data.pose.pose.orientation.w

    rospy.loginfo("Received Odometry Data: "
                  "Position (x, y, z): (%f, %f, %f), Orientation (w): %f",
                  position_x, position_y, position_z, orientation_w)

def goal_callback(data):
    # Callback function to handle received goal point messages
    global goal_x, goal_y, goal_z

    # Store the goal point
    goal_x = data.x
    goal_y = data.y
    goal_z = data.z

    rospy.loginfo("Received Goal Point: (x, y, z): (%f, %f, %f)",
                  goal_x, goal_y, goal_z)

def main():
    rospy.init_node('odometry_goal_subscriber', anonymous=True)

    # Set the rate to 5 Hz
    rate = rospy.Rate(5)

    # Subscribe to the '/ground_truth/state' odometry topic
    rospy.Subscriber('/ground_truth/state', Odometry, odom_callback)

    # Subscribe to the '/goal' topic with Point message type
    rospy.Subscriber('/goal', Point, goal_callback)

    while not rospy.is_shutdown():
        # Perform other processing here if needed

        rate.sleep()

if __name__ == '__main__':
    main()
```

article

Python Code for PID Controller in ROS

2 PID Controller Code

Below is the Python script implementing a PID controller in ROS. The script subscribes to odometry and goal topics to receive current and goal positions, respectively, and publishes control commands based on PID control algorithm:

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

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Point

# Global variables for current position, goal position, and control command
current_position = Point()
goal_position = Point()
control_command = Twist()

# PID controller gains
kp = 0.5  # Proportional gain
ki = 0.0  # Integral gain
kd = 0.0  # Derivative gain

# Previous error for derivative control
previous_error = Point()  # Initialize previous_error here

def odom_callback(data):
    # Callback function to update the current position from the odometry
    global current_position
    current_position = data.pose.pose.position

def goal_callback(data):
    # Callback function to update the goal position
    global goal_position
    goal_position = data

def pid_controller():
    global previous_error  # Declare previous_error as global

    # Initialize the ROS node
    rospy.init_node('pid_controller', anonymous=True)

    # Create subscribers for odometry and goal topics
    rospy.Subscriber('/ground_truth/state', Odometry, odom_callback)
    rospy.Subscriber('/goal', Point, goal_callback)
    # Create a publisher for control commands (/cmd_vel)
    control_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

    # Create a rate object to control the PID loop frequency (adjust as needed)
    rate = rospy.Rate(5)  # 5 Hz

    while not rospy.is_shutdown():
        # Calculate the error between current and goal position
        error = Point()
        error.x = goal_position.x - current_position.x
        error.y = goal_position.y - current_position.y
        error.z = goal_position.z - current_position.z

        # PID control algorithm
        control_command.linear.x = kp * error.x + ki * error.x + kd * (error.x - previou
        control_command.linear.y = kp * error.y + ki * error.y + kd * (error.y - previou
        control_command.linear.z = kp * error.z + ki * error.z + kd * (error.z - previou

        # Publish the control command
        control_publisher.publish(control_command)

        # Store the current error for the next iteration
        previous_error = error

        rate.sleep()

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