初赛

远程连接

设备号： 4-1 NC077 4-2 YN275

使用vscode中的remote-ssh拓展，通过ssh连接机器狗进行开发，机器狗上已安装code server

在实验室cyberdog wifi环境下，各狗的ip地址和ssh登陆凭证为

> 4-1:mi@10.0.0.189 4-2:mi@10.0.0.245
>
> 连接密码均为123

简单命令行控制

坐/立 使用service （CyberDog_4_1 或者 CyberDog_4_2）

```
ros2 service call /CyberDog_4_1/motion_result_cmd protocol/srv/MotionResultCmd
"motion_id: 111"
```

走 使用topic

```
ros2 topic pub /CyberDog_4_1/motion_servo_cmd protocol/msg/MotionServoCmd "
{motion_id303, vel_des: [0.0, 0.0, 0.0], step_height: [0.05, 0.05]}"
```

监听超声信号 使用topic

```
ros2 topic echo /CyberDog_4_1/ultrasonic_payload
```

各个参数的含义：见开发者文档

ros2 pkg 控制

开发文件目录结构为

```
~
  - preliminary
    - src
      - avoidance
        - avoidance
          - 若干具体实现的py脚本
        - setup.py
        - ...
    - log
    - ...
```

运行方法

cd ~/preliminary

colcon build （这时候会报一个warning，pkg也没有编译成功）

source ~/preliminary/install/setup.bash （这时候会报一个 not found，不用管）

colcon build （这次build之后就出现pkg了）

ros2 run avoidance stand （stand/stand_sit/walk 是.py文件的名称）

如何加新的功能

建立新的xx.py，在main中实现功能

在setup.py.'console_scripts' 中添加一行'xx = avoidance.xx:main' （猜测也可以不在main中实现，然后把:main换成其他函数名称）

初赛避障实现

初赛的避障脚本为~/preliminary/src/avoidance/avoidance/avoid_jez.py，相关代码已上传

超声感知

避障方案通过超声实现，代码中定义了名为 `ultrasonic_sensor`的类，它用于处理超声波传感器的数据。这个类继承自 `Node`基类，`Node`是ROS2中用于创建节点的类。下面是对这个类的详细解释：

```
class ultrasonic_sensor(Node):
    def __init__(self, name) -> None:
        super().__init__(name)
        self.sub = self.create_subscription(Range,
'/CyberDog_4_1/ultrasonic_payload', self.sub_callback, 10)
        self.status = SAFE

    def sub_callback(self, msg: Range):
        self.dist = msg.range
        if self.dist <= 0.6:
            self.status = TURN
            self.get_logger().info(f"Need to turn")
        elif self.dist <= 0.5:
            self.status = STOP
            self.get_logger().info(f"Need to stop")
        else:
            self.status = SAFE

    def get_status(self):
        return self.status
```

- `__init__(self, name) -> None`：类的构造函数，它接受一个参数 `name`，这是节点的名称。在构造函数中，它调用基类的构造函数 `super().__init__(name)`，然后创建一个订阅者 `self.sub`，订阅名为 `/CyberDog_4_1/ultrasonic_payload`的 `Range`类型的主题，并指定回调函数

self.sub_callback和队列大小为10。self.status初始化为 SAFE，这是表示安全状态的常量。

- sub_callback(self, msg: Range)：这是订阅者的回调函数，每当有新的消息发布到 /CyberDog_4_1/ultrasonic_payload主题时，这个函数就会被调用。msg参数是收到的消息，它包含超声波传感器测量的距离。根据距离 self.dist的值，节点会更新 self.status的状态为 TURN、STOP或 SAFE，并通过日志记录相应的信息。
- get_status(self)：这个方法返回当前的状态 self.status。根据状态的不同SAFE,STOP,TURN，分别进行行走、后退和转向。

行走控制

移动转向通过定义的 move_cmd类实现，它用于发布运动控制命令到名为 /CyberDog_4_1/motion_servo_cmd的主题，这是开发者已经编写好的控制运动的方法。下面是对这个类的详细解释：

```
class move_cmd(Node):
    def __init__(self, name):
        super().__init__(name)
        self.publisher_ = self.create_publisher(MotionServoCmd,
'/CyberDog_4_1/motion_servo_cmd', 10)
        self.count = 0

    def pub_motion(self, motion_id, vel_des, step_height=[0.05, 0.05]):
        msg = MotionServoCmd()
        msg.motion_id = motion_id
        msg.vel_des = vel_des
        msg.step_height = step_height
        self.publisher_.publish(msg)
        # self.get_logger().info(f'Publishing: "{msg.motion_id}" "{msg.vel_des}" "
{msg.step_height}" {self.count}')
        self.count += 1
```

- __init__(self, name)：类的构造函数，它接受一个参数 name，这是节点的名称。在构造函数中，它调用基类的构造函数 super().__init__(name)，然后创建一个发布者 self.publisher_，发布到名为 /CyberDog_4_1/motion_servo_cmd的 MotionServoCmd类型的主题，并指定队列大小为10。self.count初始化为0，可能是用来记录发布消息的次数。
- pub_motion(self, motion_id, vel_des, step_height=[0.05, 0.05])：这个方法创建一个 MotionServoCmd类型的消息 msg，并填充它的字段 motion_id、vel_des和 step_height。然后它使用 self.publisher_.publish(msg)发布这

个消息。motion_id是用于指定运动类型的标识符，可以参看开发者文档，vel_des可能是速度，step_height是步态高度。

避障实施

最后的避障行走实现结合超声和行走，通过主程序中的一个循环实现，它将持续运行一分钟（60秒），期间会不断检查超声波传感器的状态并相应地控制机器人的运动。这里使用了之前定义的 ultrasonic_sensor和 move_cmd类。下面是对这段代码的详细解释：

```
import time
import rclpy
# 假设 ultra 和 move 是之前创建的 ultrasonic_sensor 和 move_cmd 实例
# ultra = ultrasonic_sensor('ultra')
# move = move_cmd('move')
now = time.time()  # 获取当前时间
while time() - now < 60:  # 循环将持续60秒
    rclpy.spin_once(ultra)  # 处理一次ultrasonic_sensor节点的回调
    if ultra.get_status() == SAFE:
        turning = False
        move.pub_motion(303, [0.5, 0., 0.])  # 前进
    elif ultra.get_status() == TURN:
        if not turning:
            # dir = random.choice([1,-1])  # 随机选择转向方向
            dir = 1  # 确定转向方向
            turntime = time()
            turning = True
        if time() - turntime < 3.3:
            move.pub_motion(303, [-0.02, 0., 0.5 * dir])  # 转向
        else:
            move.pub_motion(303, [-0.01, 0., -0.5 * dir])  # 微调
    elif ultra.get_status() == STOP:
        move.pub_motion(303, [-0.3, 0., 0.])  # 停止
```

- while time() - now < 60:：这个循环将运行直到当前时间与 now之间的差值达到60秒。
- rclpy.spin_once(ultra)：这个函数调用是为了处理 ultrasonic_sensor节点的一次回调。它接收和处理来自超声波传感器的数据。
- if ultra.get_status() == SAFE:：如果传感器报告安全状态，机器人将前进。
- elif ultra.get_status() == TURN:：如果传感器报告需要转向，机器人将执行转向动作。这里的转向逻辑使用了 turning变量来跟踪是否已经在转向，以及turntime来记录开始转向的时间，当转向了90度仍然不可以安全前进时，机器人向反方向转向。

- `elif ultra.get_status() == STOP:`：如果传感器报告需要停止，机器人将减速停止。