ROS2_5일차_최범수

19기 예비단원 최범수

흐름도

설정된 목표 거리 (desired_distance) 과 LiDAR 데이터를 구독 /scan 토픽을 비교해서 거리 차이 계산을 통해 현재 거리와 목표거리의 차이를 구한다. 그 후 각속도 및 선속도를 계산해서 일정한 가중치를 곱해 적절하게 회전하도록 하였다. 그 후 /cmd_vel 토픽에 명령을 보내서 로봇 방향 및 속도를 조정한다.

```
this->declare_parameter<double>("desired_distance", 1.4);
```

이 코드를 통해 목표 거리를 정한다. 그 후 파라미터에 적용하고 /scan 토픽을 서브스크라이브 한다.

```
WallFollowerNode::WallFollowerNode() : Node("wall_follower")
{
    this->declare_parameter<double>("desired_distance", 1.4);
    this->get_parameter("desired_distance", desired_distance_

    scan_subscriber_ = this->create_subscription<sensor_msgs:
        "/scan", 10, std::bind(&WallFollowerNode::scanCallbac

    cmd_vel_publisher_ = this->create_publisher<geometry_msgs
}
```

10이라는 숫자는 메시지 큐 크기를 제한한것이다. 콜백함수로서 scanCallback를 지정한다. 현재
클래스 멤버 함수로 있다. cmd_vel_publisher는 속도 명령을 전달하는 토픽과 10이라는 메시지 큐 크기를 통해 Twist라는 메시지를 생성한다.

```
void WallFollowerNode::scanCallback(const sensor_msgs::msg::La
{
    double left_distance = scan_msg->ranges[270];
    double right_distance = scan_msg->ranges[90];
    double front_distance = scan_msg->ranges[0];

    geometry_msgs::msg::Twist cmd_vel_msg;
    cmd_vel_msg.linear.x = 0.15; // 전진 속도 설정

    if (front_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = -0.7; // 오른쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Front wall");
    }

    // 오른쪽 벽이 가까울 때 왼쪽으로 회전
    else if (right_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = 0.7; // 왼쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Turning left.");
    }
    // 왼쪽 벽이 가까울 때 오른쪽으로 회전
    else if (left_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = -0.7; // 오른쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Turning right.");
    }
    // 양쪽 벽이 적절한 거리일 때는 직진
    else
    {
        cmd_vel_msg.angular.z = 0.0;
        RCLCPP_INFO(this->get_logger(), "Both sides clear. Mo
    }

    RCLCPP_INFO(this->get_logger(), "Left distance: %f, Right
    RCLCPP_INFO(this->get_logger(), "cmd_vel: linear.x=%f, an
                cmd_vel_msg.linear.x, cmd_vel_msg.angular.z);
```

```
    cmd_vel_publisher_->publish(cmd_vel_msg);
}
```

callback 함수를 만들어 레이저 스캔에 따른 작동 메커니즘을 구성하였고 그 후 cmd_vel에 퍼블리시 하는 과정이다.

```
double left_distance = scan_msg->ranges[270];
double right_distance = scan_msg->ranges[90];
double front_distance = scan_msg->ranges[0];
```

값을 읽어올 방향을 지정해준다.

```
if (front_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = -0.7; // 오른쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Front wall");
    }

    // 오른쪽 벽이 가까울 때 왼쪽으로 회전
    else if (right_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = 0.7; // 왼쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Turning left.");
    }
    // 왼쪽 벽이 가까울 때 오른쪽으로 회전
    else if (left_distance < desired_distance_)
    {
        cmd_vel_msg.angular.z = -0.7; // 오른쪽으로 회전
        RCLCPP_INFO(this->get_logger(), "Turning right.");
    }
    // 양쪽 벽이 적절한 거리일 때는 직진
    else
    {
        cmd_vel_msg.angular.z = 0.0;
        RCLCPP_INFO(this->get_logger(), "Both sides clear. Mo
    }
```

정면을 인식하면 오른쪽으로 회전해 충돌을 피했고 오른쪽 벽과 왼쪽 벽의 거리를 계산해서 반대 방향으로 틀며 벽이 인식되지 않으면 전진하도록 예외처리를 하였다.

```
cmd_vel_publisher_->publish(cmd_vel_msg);
```

마지막으로 실제 퍼블리시를 해서 작동하도록 하였다.