ROS DAY4
과제 2번 보고서

2023741024
로봇학부 박건후

목차

(1) day4_hw2_pkg 설명

joy_node로부터 조이스틱 값을 받아오게 됩니다. topic echo로 /joy값을 찍어본 결과 axes로 -1~1사이의 값을 갖고 있다는 것을 알게 되었습니다.

```
geonhu@PGH:~$ ros2 topic echo /joy
header:
  stamp:
    sec: 1758163697
    nanosec: 972467088
  frame_id: joy
axes:
- -0.0
- -0.0
- 1.0
- -0.0
- -0.0
- 1.0
- 0.0
- 0.0
buttons:
```

이런 joy_node의 메시지를 받기 위해서는 메시지 타입을 알고 해당 토픽이름을 통해 Subscription으로 값을 받아와야 합니다. 이 메시지의 타입은 sensor_msgs::msg::Joy였습니다. 그래서 이 타입을 통해 아래와 같이 Subscription을 구성하였습니다.

```
rclcpp::Subscription<sensor_msgs::msg::Joy>::SharedPtr sub_ptr;
```

메시지를 받았다면 이 값을 다시 패키지 2로 퍼블리시 해야 합니다. 이 때 이 메시지에 있는 선속도 값, 각속도 값을 퍼블리시 해야 합니다. 이 값을 한 번에 보낼 수 있는 타입이 있어야 하는데 우선 Joy의 타입을 사용하고 싶었지만 그렇게 되면 프로그램을 실행할 때 조이스틱에서 보내는 값과 1번 패키지에서 퍼블리시하는 것과 섞이거나 꼬일 수 있었습니다. 그래서 터틀심에서 선속도, 각속도를 저장해 한 번에 보냈던 것을 활용하여 터틀심 cmd_vel 토픽의 타입인 geometry_msgs::msg::Twist 타입을 Publisher의 템플릿 타입으로 정하게 되었습니다. 아래의 사진이 해당 Publisher를 선언한 것입니다.

```
rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr pub_ptr;
```

```
joy_subscribe::joy_subscribe():Node("sub_and_pub"){ //노드명 잡기
    pub_ptr = create_publisher<geometry_msgs::msg::Twist>("/cmd_vel", 10); //cmd_v
    sub_ptr = create_subscription<sensor_msgs::msg::Joy>("/joy", 10, std::bind(&jo
}
```

이를 구현한 부분을 보겠습니다. 위 사진은 각각을 토픽명과 해당 타입으로 Publisher와 Subscription을 각각 생성한 것입니다. 그리고 Subscription을 생성할 때 메시지를 받았을

때 호출할 콜백 메서드를 바인딩 해줬습니다.

```
geometry_msgs::msg::Twist value;

value.linear.x = msg->axes[1] * 1.0; /
value.angular.z = msg->axes[3] * 1.0;
```

이 부분은 메시지를 받았을 때 호출되는 콜백 메서드 구현 부분입니다. 퍼블리시할 메시지 객체를 생성하고 그 객체에 값을 저장하고 퍼블리시하는 것입니다. msg의 멤버로는 topic echo에서 보셨듯 axes와 button이 있었습니다. 제가 직접 topic echo를 통해 조작해본 결과 axes[0]이 왼쪽 조이스틱의 가로 조작, axes[1]이 왼쪽 조이스틱의 세로 조작, axes[3]이 오른쪽 조이스틱의 가로 조작, axes[4]가 오른쪽 조이스틱의 세로 방향 조작이었습니다. 저는 이 중 왼쪽 세로 조작을 선속도로, 오른쪽 가로 조작을 각속도로 쓸 것으로 정하여 이를 Twist 객체의 멤버에 대입하였습니다. 이 때 1.0을 곱한 이유는 선속도, 각속도로 변환을 해야 하는데 조이스틱으로부터 받아온 값에서는 단위라는 개념이 없기에 각각 1m/s, 1rad/s를 곱하는 방식으로 변환하는 것을 명시하기 위해 작성한 것입니다.

(2) day4_hw2_2_pkg 설명

```
rclcpp::Subscription<geometry_msgs::msg::Twist>::SharedPtr sub_ptr;
std::shared_ptr<tf2_ros::TransformBroadcaster> tf_broadcaster; //tf_
```

패키지 1에서는 geometry_msgs::msg::Twist로 publish했으므로 패키지 2에서 geometry_msgs::msg::Twist로 받아야 합니다. 그리고 TransformBroadcaster를 사용해야 하며 메시지를 보낼 때에 실제로 이 클래스를 통해 보내게 되므로 이를 shared_ptr 선언하여 이 객체의 sendTransform 메서드를 통해 메시지를 보내고자 했습니다.

```
sub_ptr =this->create_subscription<geometry_msgs::msg::Twist>(
    "cmd_vel", 10, std::bind(&controlling::msg_callback, this, std::placeholders::_1));
```

클래스 생성자 구현 부분입니다. 이처럼 sub_ptr을 생성하고 메시지를 받았을 때 호출되는 콜백 메서드를 바인딩한 것입니다.

```
tf_broadcaster = std::make_shared<tf2_ros::TransformBroadcaster>(this);
```

그리고 shared_ptr을 선언하였으므로 make_shared를 통해 객체를 생성해줬습니다. 이때 this포인터를 넘겨서 TransformBroadcaster 클래스 생성자에 필요한 인자를 넣어줬습니다. TransformBroadcaster는 ros::Node 참조를 필요로 하기에 그렇습니다.

```
timer_ = this->create_wall_timer(0.01s, std::bind(&controlling::timer_callback, this));
```

이전에 많이 했듯이, create_wall_timer로 타이머를 둬서 시간 간격마다 콜백 메서드를 호출하게 했습니다. 0.01초마다 timer_callback메서드를 통해 odom의 위치를 최신화하고자 했습니다. 메시지 콜백 메서드에서는 이를 처리하려고 하면 메시지를 받을 때마다 최신화를 하게 되는데 그렇게 되면 좀 불안정하게 위치 최신화가 될 것 같았기 때문입니다. 그래서 0.01초라는 간격마다 일정하게 위치를 최신화하고 메시지를 보내어 Rivz의 odom위치를 최신화하고자 했습니다.

```
void controlling::msg_callback(const geometry_msgs::msg::Twist::SharedPtr msg){
    linear_v=msg->linear.x;  //msg받으면 callback메서드가 끝나면 내용이 소멸되므로 이를 멤
    angular_v = msg-> angular.z; //timer_callback에서 이를 처리해야 하므로
}
```

메시지를 받으면 호출되는 콜백 메서드입니다. cmd_vel의 메시지를 받아서 이를 linear_v, angular_v라는 멤버변수에 저장하는 부분입니다. 그렇게 한 이유는 timer_callback에서 이 값을 사용하여 거리를 계산해야 하는데 메시지 콜백 메서드가 종료되면 메시지 데이터가 사라지므로 이렇게 멤버변수에 저장하게 되었습니다.

```
double interval = 0.01;                    /
x+= interval* linear_v*cos(angle);
누적
y+= interval* linear_v*sin(angle);
```

timer_callback의 구현부입니다. 0.01초마다 호출되므로 시간 간격을 0.01초로 해서 해당 선속도로 0.01초 동안 이동한 거리를 구하여 이를 x, y라는 멤버변수에 누적시켰습니다. TransformBroadcaster로 메시지를 보내게 되는데 이 메시지의 타입은 geometry_msgs::msg::TransformStamped이었습니다. 그리고 메시지를 보낼 때 설정되어

야 하는 값은 header, child_frame, transform이라는 멤버였습니다. 그 중 transform멤버에는 translation과 rotation이라는 멤버가 있었고 translation 멤버에는 x, y, z라는 멤버가 있었고 rotation에는 x, y, z, w가 있었습니다. translation에는 x, y, z 축에서의 좌표 값을 적어줘야 합니다. 2차원 평면이기에 z값은 없으므로 0이 되고 x, y 위치값을 구해야 했기에 위와 같이 x, y라는 멤버 변수를 생성해서 누적시켰던 것입니다. 갖고 있는 정보는 선속도와 각속도입니다. 이 때 선속도를 x와 y성분으로 나누어 각 성분마다 시간을 곱하면 평균 속도에 대한 시간을 곱한 것으로 거리 값이 되기에 이를 누적시키면 x와 y 위치를 구할 수 있게 됩니다.

```
angle+=interval*angular_v;
```

각속도 또한 시간 간격을 곱하면 그 시간 동안의 이동된 각도값이 되므로 이를 angle이라는 멤버변수에 누적시켰습니다. 이 angle을 구한 이유는 rotation에 이 값을 넣어야 했기 때문입니다.

```
rclcpp::Time now = this->get_clock()->now();
geometry_msgs::msg::TransformStamped t; //메
```

그리고 header와 child_frame을 초기화해줘야 합니다. header에는 stamp와 frame_id라는 멤버가 있습니다. 이 때 stamp는 현재 시간을 값으로 넣어줘야 하기에 현재 시간을 구해서 now라는 변수에 넣어준 것입니다. 그리고 TransformStamped 클래스의 객체를 생성하여 메시지 내용을 담을 객체를 생성해줬습니다.

```
t.header.stamp = now;
t.header.frame_id = "map";
t.child_frame_id = "odom";
```

stamp를 현재 시간으로 초기화시켜주고, frame_id에는 부모 프레임이라는 개념을 넣어줘야 하는 것이었습니다. map->odom->baselink->... 순으로 가게 되는데 현재 과제에서는 부모 프레임이 map이고 자식 프레임이 odom입니다. 그래서 frame_id에 map을 넣어주고 child_frame_id에 odom을 넣어줬습니다.

```
t.transform.translation.x = x; /
t.transform.translation.y = y; /
t.transform.translation.z = 0.0;
```

누적된 x, y값은 넣어줘서 x, y, z에 대한 좌표값을 넣어주는 부분입니다.

```
tf2::Quaternion q;//오일
q.setRPY(0, 0, angle);
```

그리고 회전과 관련된 rotation을 초기화할 때 rall, pitch, yaw에 대한 벡터값을 넣고, w에는 스칼라값인 각도 값을 넣는 것으로 알고 있습니다. 또 이는 오일러 변환과도 관계있습니다. 하지만 오일러 변환의 단점으로 짐벌락이라는 현상이 생겨서 회전하다보니 축이 겹치는 현상을 의미합니다. 물론 z축에 대해서만 회전하기에 그럴 일은 없지만 이 짐벌락에 대한 해결책인 쿼터니언 변환 라이브러리를 사용하고자 했습니다. 그래서 tf2/LinearMath/Quaternion이라는 헤더파일을 인클루드하여 사용했습니다. 메서드로 setRPY를 사용하게 되는데 z축에 대한 회전이기에 angle값을 Yaw에 해당되는 3번째 인수에 넣어줬습니다.

```
t.transform.rotation.x = q.x();
t.transform.rotation.y = q.y();
t.transform.rotation.z = q.z();
t.transform.rotation.w = q.w();
```

그리고 x(), y(), z(), w()메서드로 쿼터니언 변환한 각각의 값을 받아 rotation을 초기화했습니다.

```
tf_broadcaster->sendTransform(t);
```

멤버 초기화를 모두 끝냈으니 TransBroadcaster의 메서드를 통해 이 TransformStamped 메시지를 보내면 됩니다.

아래 링크는 제가 tf에 대해 공부한 자료, TransformStamped 라이브러리에 대한 멤버 정보 쿼터니언 변환에 대한 것을 공부할 때 참고한 사이트입니다.

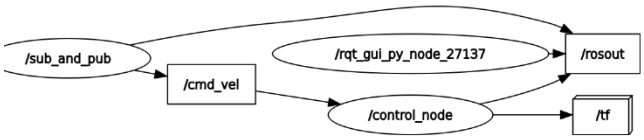
https://itsnyeong.tistory.com/14#google_vignette : 오일러, 쿼터니언 변환

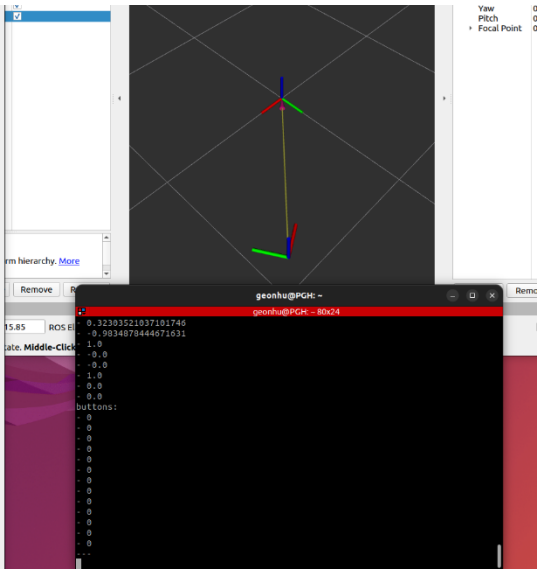
https://docs.ros.org/en/jade/api/tf/html/c++/classtf_1_1TransformBroadcaster.html#ad187b6778814f347014798260b32ddc4 :TransformBroadcaster 생성자, 라이브러리 조사

https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/TransformStamped.html

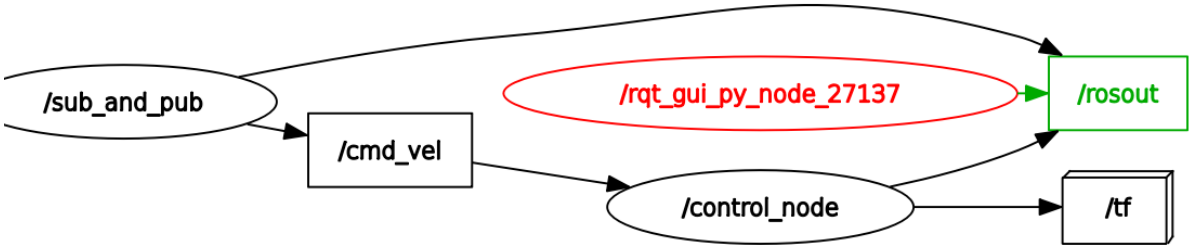
: TransformStamped의 메시지 보낼 때 채워줘야 하는 멤버들

https://eeoon.github.io/robotics/TF-content/ : TF관계 개념 및 사용되는 매개변수, 라이브러리, TransformStamped 멤버 초기화할 때 코드 참고





이 부분은 실행 화면으로 rqt_graph, topic echo, rviz를 모두 띄운 화면 스크린샷을 찍은 것입니다.



패키지 1의 노드명인 sub_and_pub에서 cmd_vel 토픽을 통해서 패키지 2의 노드인 control_node에 보내고 이 노드에서 tf로 메시지를 보내는 부분입니다.

이 부분은 topic echo 부분으로 위부터 차례로 axes 0,1,2,3,4,5...를 나타낸 부부입니다. 현재 axes 2를 제외한 0, 1, 3, 4부분의 값이 조이스틱 값에 따라 최신화되는 것을 확인할 수 있었습니다.