센서모션로봇실습

odometry

학　과 : 스마트시스템소프트웨어학과

이　름 : 20170404 한종수

제출일 : 2021. 05. 11

1. Week10 과제에서 하고자 하는 것 (목표)가 무엇인가?(1점)

    1.1.Odometry는 시간에 따른 위치 변화를 추정하기 위해 모션 센서의 데이터를 사용하는 것이다. 목표는 Odometry 실습으로 바퀴(센서) 정보(속도)를 이용해 시간의 흐름에 따른 로봇의 위치를 계산하는 것이다.

2. calcWheelVelocityGazeboCB() (1점)

    2.1.ptr->twist[2], ptr->twist[4] 가 무엇인가? (0.5점)

        2.1.1.ptr에는 gazebo_msgs::LinkStates에 대한 정보가 들어온다. LinkState에서 pose는 위치 좌표와 회전 각 정보가 들어오고 twist에는 x, y, z 각 축의 속도 성분과 각속도 성분이 들어온다. 이 값들은 배열로 저장되어 로봇의 각 link에 대한 pose, twist 정보를 인덱싱하여 가져올 수 있다. wheel_1, wheel_3은 인덱스가 각각 2, 4로 되어있다. 따라서 ptr->twist[2], ptr->twist[4]은, 각각 wheel_1(왼쪽 바퀴), wheel_3(오른쪽 바퀴)의 twist 정보를 가져온다.

    2.2.wheel_1_velocity, wheel_3_velocity, velocity, theta_dot 계산 방법. (0.5점)

        2.2.1.wheel_1_velocity & wheel_3_velocity: wheel_1의 twist 정보를 가져오면 현재 wheel_1의 x, y, z 축 속도 벡터 정보를 알 수 있다. 각 축 벡터를 모두 더한 것이 wheel_1의 속도 벡터 정보가 된다. 이 벡터의 크기가 wheel_1의 속력 정보가 된다. 이를 구하는 방법은 벡터의 크기를 구하는 방법과 같으므로 각 축 벡터의 크기를 제곱하여 더한 후 제곱근을 취하여 구할 수 있다. wheel_3_velocity도 마찬가지로 구할 수 있다. 이미 theta_dot, L, R, velocity 정보를 알고 있다면, wheel_3_velocity는 (2velocitiy + theta_dot * L) / 2R, wheel_1_velocity는 (2velocitiy - theta_dot * L) / 2R 로 구할 수 있다.

        2.2.2.velocity: 축 중심의 속력이다. 축 중심은 wheel_1, wheel_3의 중심에 있다. 축 중심의 속력은 (wheel_3_velocity + wheel_1_velocity) * R / 2 로 구할 수 있다.

        2.2.3.theta_dot: 각속도이다. 각속도는 축 중심이 회전하는 속도로 (wheel_3_velocity - wheel_1_velocity) * (R / 2)로 구할 수 있다.

3. boardcastTransform (1점)

    3.1.setOrigin() 과 setRotation(), sendTransform() 의 역할을 설명하라. (0.5점)

        3.1.1.setOrigin(): TF되어 만들어질 좌표계의 원점 각 축에 대한 이동 (Translation)

        3.1.2.setRotation(): TF되어 만들어질 좌표계의 각 축에 대한 회전 설정 (Rotation)

        3.1.3.sendTransform(): 앞의 두 함수로 만들어진 TF식, TF 발행 시각, parent frame, child frame. 이 네가지 인자들을 argument로 넘겨주면 parent frame으로 부터 TF된 child frame을 생성하고 publish한다.

    3.2./custom_odom, /custom_base, /base_link 의 관계를 설명하라. (0.5점)

        3.2.1./custom_base & /base_link: 현재 로봇의 회전과 이동은 로봇의 바퀴가 전륜으로 작동하기때문에 로봇의 중앙이 기준이 아닌 로봇의 앞 바퀴 축의 중앙을 기준으로 하고 있다. custom_base는 launchfile에서 staticTF로 로봇의 중심인 base_link 좌표계에서 앞 바퀴 쪽으로 0.1m만큼 축을 이동시킨 좌표계이다. 이 좌표계로 차량이 실제 움직이는 축을 표현할 수 있다.

        3.2.2./custom_odom & /custom_base: yaml파일에서 parameter로 base_link_id로 /custom_base, odom_link_id 로 /custom_odom을 넘겨준다. sendTransform에서 3.1에서 설명으로 만들어진 TF를 /custom_odom을 parent로 TF된 custom_base를 publish한다. custom_base는 부모 좌표계 custom_odom으로부터 사용자 키보드 입력을 기반으로 TF된 좌표계를 의미한다.

4. pubTF() (2점)

    4.1.dt와 theta, x_dot, y_dot, x, y 계산 방법. (1점)

        4.1.1.dt: 이전 발행부터 현재 발행까지의 시간이다. 이전 발행시간은 cur에 저장되었고 현재 시간은 ros::Time::now() 이다. (ros::Time::now() - cur).tosec()으로 ros 시간으로 계산된 시간 차를 초 단위로 바꾸어 구할 수 있다.

        4.1.2.theta: 현재 속도 벡터의 각도이다. 이전 theta에 theta_dot을 시간에 적분한 값인 theta_dot * dt를 더해서 새로 얻는다.

        4.1.3.x_dot: 현재 속도 벡터의 x축 성분으로 x축 속도이다. 속력인 velocity에 cos(theta)를 곱함으로 구할 수 있다.

        4.1.4.y_dot: 현재 속도 벡터의 y축 성분으로 y축 속도이다. 속력인 velocity에 sin(theta)를 곱함으로 구할 수 있다.

        4.1.5.x: 현재 x축 상의 좌표이다. 이전 x좌표에 x_dot을 시간에 적분한 값인 x_dot * dt를 더해서 얻는다.

        4.1.6.y: 현재 y축 상의 좌표이다. 이전 y좌표에 y_dot을 시간에 적분한 값인 y_dot * dt를 더해서 얻는다.

    4.2.본인이 짠 코드를 캡쳐해서 추가할 것. (todo 부분) (1점)

        4.2.1.

```
// todo - odometry 계산. ( theta, x_dot, y_dot, x, y )
double dt = (ros::Time::now() - cur).toSec();
theta += theta_dot * dt;
x_dot = velocity * std::cos(theta);
y_dot = velocity * std::sin(theta);
x += x_dot * dt;
y += y_dot * dt;
```