Encoder와 Imu의 EKF 센서 퓨전을 통한 2휠 차동 모바일 로봇의 Orientation보정

2020077792 공민수

Orientation Calibration of a Two-Wheel Differential Mobile Robot Using EKF Sensor Fusion of Encoder and IMU

Minsu Kong (2020077792)

초 록

로봇의 Localization은 인식 데이터를 이해하고 판단하기 위해 필수적이다. 맵이 주어지지 않은 상황에서 로봇은 Odometry를 통해 초기 위치 및 방향에서 이동할 위치와 방향을 알 수 있다. 즉 Odometry는 상대 위치 개념으로 자신의 상태를 파악하는 데 유용하게 사용된다. 하지만 Odometry 하나만으로는 정확한 상태 추정이 불가능하다. 휠의 슬립, 운동 모델링의 부정확성 등을 통해 주행 누적 오차가 발생한다. 따라서 단기적으로 Odometry의 정확도는 우수하지만 장거리 주행에서의 정확도는 떨어진다. 이에 GPS, IMU, LIDAR와 같은 센서를 추가하여 주기적으로 상태를 보정한다. 본 논문에서는 Extended Kalman Filter를 사용하여 2휠 차동 모바일 로봇의 Orientation을 보정하는 내용을 담았으며 이는 이후 LIDAR를 사용한 SLAM 프로젝트에 기초가 된다.

1. 서 론

확률론적 로보틱스는 로봇이 실제 세계에서 가지는 여러 불확실성 때문에 로봇의 상태를 확률적으로 추론하려는 이론이다. 센서 오차, 운동 모델링의 오차, 동적 환경과 같은 여러 요인들이 불확실성의 원인이 된다. 확률적 추론에 기본이 되는 이론인 베이즈 정리가 있으며 이를 기반한 베이즈 필터가 존재한다. 또한 베이즈 필터에서 실용적인 사용을 위해 파생된 칼만필터, 파티클 필터 등과 같은 상태 추정 알고리즘이 존재한다. 본 논문은 EKF(Extended Kalman Filter)를 활용한 2휠 차동 모바일 로봇의 방향 상태 추정 내용을 담았다. 작성 순서는 다음과 같다. 2장에서 베이즈 정리, 베이즈 필터를 소개하고 칼만필터, 확장 칼만 필터를 정리한다. 그 다음 3장에서 2휠 차동 모바일 로봇에 대한 상태 벡터, 제어 입력, 측정 데이터 표현을 정의하고 상태 전이 모델, 관측 모델을 정의한다. 이후 프로세스 오차, 센서 오차를 테스트를 통해 공분산을 구한다. 그리고 4장에서 최종 알고리즘을 구현한다. 5장에서는 로봇을 일정 각속도로 회전시키면서 RViz를 통해 Yaw의 예측값, 측정값, 필터 계산값을 시각화 한다. 마지막으로 6장에서 결론 및 향후계획을 밝힌다.

2. 배경 이론

2.1 베이즈 정리

베이즈 정리는 베이즈 확률론의 핵심 개념으로, 사전확률을 새로운 증거를 바탕으로 사후 확률로 갱신하는 이론이다. 이것이 확률론적 로보틱스에서도 매우 중요한 역할을 하며, 예를 들어 로봇의 위치 x, 센서 데이터 z를 확률변수로 표현 가능하다.

$$p(x|z) = \frac{p(x,z)}{p(z)} \text{ ------ Eq (1)}$$

$$p(z|x) = \frac{p(x,z)}{p(x)} \text{ ------ Eq (2)}$$

Eq(1)과 Eq(2)를 연립하여 베이즈 정리의 일반 형태를 얻을 수 있다.

$$p(x|z) = \frac{p(z|x)*p(x)}{p(z)} \text{ -- Eq (3)}$$

p(x)는 사전확률로 로봇이 위치 x에 있을 것으로 기대되는 초기 확률이다. P(z|x)는 likelihood로 로봇이 위치 x에 있을 때 센서 데이터 z가 관측될 확률이다. P(x|z)는 사후확률로 센서 데이터가 관측되었을 때 로봇이 위치 x에 있을 확률이다. Eq(3)은 다음과 같이 해석 가능하다.

‘사전확률 *p(x)*를 바탕으로, 증거(우도) *p(z|x)*를 이용하여 사후 확률 *p(x|z)*를 계산한다.’ 좀 더 직관적으로 보면 센서 데이터 z가 관측됐을 때 로봇이 x에 있을 확률은 로봇이 x에 있을 때 센서 데이터 z가 인식될 확률을 기반으로 계산된다고 볼 수 있다.

.

2.2 베이즈 필터

베이즈 필터 알고리즘을 다루기전에 확률론적 로보틱스에 필요한 개념을 먼저 살펴본다.

(1) State– – – – – – – – – – (2.2.1)
(2)환경 인터렉션 – – – – – –(2,2.2)
(3)확률론적 생성 법칙 – – – (2.2.3)
(4) Belif – – – – – – – – – – (2.2.4)
(5) Bayes filter algorithm – – – (2.2.5)

2.2.1 State

State는 로봇 시스템과 환경의 모든 관련된 정보를 모아놓은 것으로 볼 수 있다. 크게 로봇 외부 환경 상태와 로봇 내부 상태로 나눌 수 있다. 외부 환경 상태는 정적 동적 환경으로 정의 가능하며 static state, dynamic state라고 한다. 예를 들어 건물, 고정 사물은 static state, 사람, 자동차는 dynamic state이다. 로봇 내부 상태에서는 kinematic state와 dynamic state로 할 수 있으며, 예를 들어 Pose 데이터는 kinematic state, Twist 데이터는 dynamic state로 볼 수 있다. 이 외에도 센서의 정상 작동 여부, 배터리 잔량과 같은 시스템 변수들 또한 state로 정의할 수 있다. 확률론적 로보틱스에서는 확률 모델을 단순화하기 위해 상태의 완전성(Markov)을 가정한다. 다시 말해 미래 상태를 예측하기 위해 과거 상태는 더 이상 유의미한 정보를 제공하지 않음을 말한다. 이는 과거 상태는 조건부로 독립하다는 의미이다. 이러한 조건을 충족시키는 시간 프로세스를 Markov Chain이라고 부른다. 하지만 실제로는 로봇의 상태를 완벽하게 측정하거나 정의할 수 없다. 따라서 로봇은 불완전한 상태 정보를 기반으로 상태 추론을 수행하게 된다.

2.2.2 환경 인터랙션

로봇과 환경 간의 인터렉션은 센서 측정과 제어 액션을 통해 이루어진다. 로봇은 센서를 통해 환경의 일시적 상태에 대한 정보를 얻게 된다. 그리고 로봇은 제어 액션을 통해 상태를 변화시킨다. 이러한 센서 측정 및 제어 액션은 시간에 따라 누적되며, 로봇은 이를 모두 데이터로 저장한다고 가정한다. 따라서 센서 측정 기록과 제어 액션 기록은 각각 측정 데이터, 제어데이터라고 불린다. 측정 데이터는 카메라 이미지, 라이다 스캔 등으로 얻어질 수 있다. 제어 데이터는 로봇의 상태 변화에 대한 정보를 제공한다. 예를 들어 Odometer를 이용하여 t초 후 로봇의 변화된 상태를 알 수 있다. Odometer도 센서 측정이지만 이는 제어 액션의 효과를 측정한다는 점이 측정 데이터와 다르다.
이러한 측정과 제어 액션은 확률론적 로보틱스에서 각각 상태 전이 확률과 측정확률로 표현된다. 이를 통해 환경과의 상호작용을 수학적으로 모델링할 수 있다.

2.2.3 확률론적 생성 법칙

시간에 대한 상태를 x_t라고 할 때 이는 이전 시점의 상태 x_{t-1}로부터 확률적으로 생성된다. 이러한 시간에 따른 상태 변화를 evolution이라고 말하며 이를 확률론적 로보틱스에서는 확률 분포 형태로 모델링한다. 이 때 u 와 z의 시간 순서는 은닉 마르코프 모델을 따른다. 즉 Figure 1에서 볼 수 있듯, 제어 액션이 먼저 선행된 후 측정이 일어난다고 가정한다.

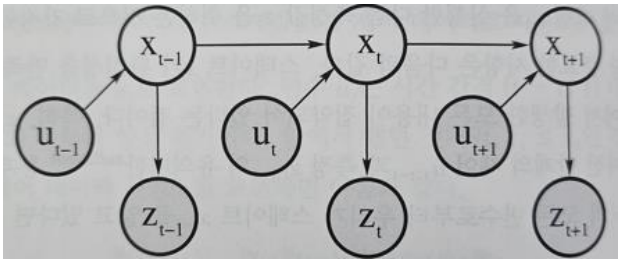

Figure1 Hidden Markov Model

State가 완전성을 만족할 경우 상태 모델을 다음과 같이 표현할 수 있다.

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t)$$

이를 상태 전이 확률이라고 한다. 완전성을 가정했을 때 이전 상태와 제어 데이터만 있으면 현재 상태를 확률적으로 알 수 있다. 그리고 측정 모델은 다음과 같이 표현할 수 있다.

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t)$$

이를 측정 확률이라고 하며 이는 측정데이터의 노이즈와 같은 불확실성을 반영한다.

2.2.4 Belief

belief란 로봇이 가진 내부 지식 상태를 말한다. 즉 로봇이 스스로 어떤 상태에 있다고 믿는 확률 분포를 의미하며 이는 진짜 스테이트와 구분할 필요가 있다. 왜냐하면 state는 바로 측정될 수 없기 때문이다. 예를 들어 센서를 통해 Pose를 직접 측정할 수 없고 데이터를 통해 추론해야 한다. 확률론적 로보틱스는 belief를 조건부 확률 분포로 표현하며 다음과 같이 정의된다.

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$$

이 표현은 시간 t에서의 state x_t에 대한 확률 분포로, 모든 이전 측정값 $z_{1:t}$와 이전 제어 입력 $u_{1:t}$에 따라 결정된다. 앞서 2.1.1.3절에서 HMM구조를 기반으로 다음과 같이 belief를 정의할 수 있다.

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t})$$

$\overline{bel}(x_t)$를 확률론적 필터링 관점에서 예측이라고 한다. 즉 과거 측정값과 현재까지의 제어명령만으로 모델링 된 확률 분포이다. 따라서 현재 측정값을 바탕으로 $bel(x_t)$ 을 계산하는 과정을 보정 또는 측정 업데이트라고 한다. 이러한 예측과 업데이트를 반복하며 belief를 갱신하는 구조는 바로 베이즈 필터이다.

2.2.5 Bayes filter

이전 2.1.1.4 장에서 베이즈 필터는 예측과 업데이트를 반복하며 belief를 갱신하는 알고리즘이라고 밝힌 바 있다. 이는 Table 1의 의사코드를 통해 구체적으로 확인할 수 있다.

Table 1 pseudo code of base filter algorithm

1:	**Algorithm Bayes_filter**($bel(x_{t-1}), u_t, z_t$)**:**
2:	*for all* x_t *do*
3:	$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx_{t-1}$
4:	$bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5:	*endfor*
6:	*return* $bel(x_t)$

Table 1에 따르면 x_t에 대한 belief를 구하기 위해 $bel(x_{t-1}), z_t, u_t$이 필요함을 알 수 있다.

- 3번 라인은 예측 단계며, $bel(x_{t-1})$에 상태 전이 확률을 곱해 $\overline{bel}(x_t)$을 계산한다.
- 4번 라인은 업데이트 단계이며, $\overline{bel}(x_t)$ 에 측정확률을 곱해 최종 확률 $bel(x_t)$을 계산한다. 특히 업데이트 단계의 계산과정은 베이즈 정리 형태를 띄는 것을 알 수 있다.

베이즈 필터 알고리즘은 불확실성을 다루는 확률론적 로보틱스에서 belief를 계산하기 위한 주요 알고리즘이다. 그러나 베이즈 필터는 상태 공간 전체에 대해 확률 분포를 유지하고 적분을 수행해야 하기 때문에, 계산량이 크고 연속 상태 공간에 대해선 수치적으로 매우 비효율적이다. 특히 유한한 적분값을 얻기 위해 상태 공간에 제약을 가해야 하며 이는 디지털 컴퓨터 환경, 실제 로보틱스 시스템에 적용되기에는 실용적인 알고리즘이 아니다. 이러한 한계를 해결하기 위해, 실제 로보틱스 응용에서는 belief를 근사하여 계산하는 다양한 파생 알고리즘들이 사용된다. 대표적으로는 Kalman Filter, Particle Filter 등이 있으며, 이들은 베이즈 필터의 구조를 유지하되 계산 가능한 형태로 확률 분포를 근사함으로써 실시간 추정을 가능하게 한다.

2.3 칼만 필터

2.2 장에서 다뤘듯이 베이즈 필터의 belief는 $p(x_t \mid z_{1:t}, u_{1:t})$로 확률 분포 자체를 나타낸다. 이것은 표현이 어렵고 계산 비용이 크기 때문에 실용적이지 않았다. 이에 belief를 가우시안 분포로 근사화 한 것이 칼만 필터이다. 가우시안 분포는 평균과 공분산으로 정의되며 다변량 정규

분포와 같다. 즉 칼만 필터는 모멘트 파라미터화를 통해 belief를 표현하게 된다.
가우시안 분포는 확률 분포상 최고점이 하나인 유니모달 특성을 가지므로 불확실성에 대한 마진이 작은 진짜 상태에 가깝게 사후확률을 얻을 수 있다. 또한 적분 계산이 아닌 행렬 연산으로 처리할 수 있어 계산 효율성이 높다. 칼만 필터를 사용하기 위해서는 연속 상태 공간이어야 하며 마르코프 가정 외에 다음 3가지 조건을 만족해야 한다.

(1) 상태 전이 확률은 가우시안 노이즈 입력 인자를 가지는 선형 함수여야 한다. 따라서 상태 전이 확률은 가우시안 분포를 따른다.

$$\boldsymbol{x}_t = A_t \boldsymbol{x}_{t-1} + B_t \boldsymbol{u}_t + \boldsymbol{\varepsilon}_t$$

$\boldsymbol{\varepsilon}_t$ 는 상태 전이 과정에서 나타날 수 있는 불확실성을 모델링한 가우시안 확률 벡터이다. 평균 0, 공분산 R_t를 갖는다.

(2) 측정 확률은 가우시안 노이즈 입력 인자를 가지는 선형 함수여야 한다.

$$\boldsymbol{z}_t = C_t \boldsymbol{x}_t + \boldsymbol{\delta}_t$$

$\boldsymbol{\delta}_t$ 는 측정 노이즈로 평균 0, 공분산 Q_t 인 가우시안 분포를 따른다.

(3) 초기 belief는 반드시 정규 분포를 따라야 한다.

이 속성의 핵심은 모두 가우시안 분포를 따르며 확률이 선형 함수라는 것이다. 따라서 belief는 가우시안 분포를 유지할 수 있다.

2.3.1 칼만 필터 알고리즘

Table 2에는 칼만 필터 알고리즘이 의사 코드로 정리되어 있다.

Table 2 pseudo code of kalman filter algorithm

1:	$\textbf{Kalman\_filter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**
2:	$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$
3:	$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$
4:	$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$
5:	$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$
6:	$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$
7:	$return\ \mu_t, \Sigma_t$

2,3번 라인이 예측 단계이고 5,6번 라인이 업데이트 단계이다. Table 1과 비교해보았을 때 칼만 필터는 belief 자체를 예측하고 업데이트 하는 것이 아닌 파라미터(평균, 공분산)을 예측하고 업데이트하는 것을 볼 수 있다. 이는 칼만 필터가 모멘트 파라미터화를 기반으로 작동하기 때문이다.
4번 라인에서는 칼만 이득 계산을 수행하며, 이는 측정값과 예측값에 대한 신뢰도를 결정하는 역할을 한다.
특히 칼만 필터의 물리적 의미는 5번 라인을 통해 알 수 있다. C_t 를 상수로 본다면 다음과 같이 표현 가능하다.

$$\boldsymbol{\mu}_t = (1 - K_t) * \bar{\boldsymbol{\mu}}_t + K_t * \boldsymbol{z}_t$$

즉 이득이 클수록 측정값를 더 많이 신뢰하며, 반대로 이득이 작을수록 예측값을 더 많이 신뢰한다. 이러한 칼만 이득은 C_t 와 Q_t 즉 상태 전이 모델의 불확실성과 측정 모델의 불확실성에 의해 결정된다.

2.4 확장 칼만 필터

칼만 필터에서는 선형 시스템에 의해 가우시안 분포가 유지되었다. 하지만 비선형 시스템일 때는 가우시안 분포가 유지되지 못한다. 따라서 테일러 1차 근사를 통해 가우시안 분포를 유지하도록 하는 방식이 확장 칼만 필터이다. 여기서 비선형 시스템이라는 것은 상태 전이 확률과 측정 확률이 비선형 함수라는 것이다. 따라서 상태 전이 확률과 측정 확률은 다음과 같이 표현된다.

$$\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) + \boldsymbol{\varepsilon}_t$$
$$\boldsymbol{z}_t = h(\boldsymbol{x}_t) + \boldsymbol{\delta}_t$$

그리고 선형행렬 A_t, B_t, C_t 가 아닌 자코비안 행렬을 사용한다.

$$F_t = \frac{\partial f}{\partial x}$$
$$H_t = \frac{\partial h}{\partial x}$$

이러한 차이를 칼만 필터 알고리즘에 반영하면 확장 칼만 필터 알고리즘이 된다.

3. 2휠 모바일 로봇 상태 추정

3.1 상태, 제어, 측정 정의

EKF를 코드를 구현하기 전에 로봇의 상태 벡터, 제어 입력 벡터, 측정 벡터를 정의해야 한다. 이 벡터들을 어떻게 정의하는지에 다양한 관점이 제시될 수 있다. 본인은 다음과 같이 정의하였다.

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$
$$\boldsymbol{u} = \begin{bmatrix} \Delta S \\ \Delta\theta \end{bmatrix}$$
$$\boldsymbol{z} = [\theta]$$

로봇의 상태 벡터는 kinematic state x,y,θ 로 정의했다. 제어 입력은 직진 변화량 각도 변화량으로 정의했다. 측정 데이터는 θ로 정의하였다.

3.2 예측 모델 및 측정 모델

3.2.1 예측 모델

2륜 차동형 모바일 로봇의 시스템 모델은 비선형 특성을 갖는다. 따라서 EKF를 사용해야 하는 것이며, 상태 예측 모델은 다음과 같이 표현 가능하다.

$$\boldsymbol{x}_k = \begin{bmatrix} x_{k-1} + \Delta S cos(\theta + \frac{\Delta\theta}{2}) \\ y_{k-1} + \Delta S sin(\theta + \frac{\Delta\theta}{2}) \\ \theta + \Delta\theta \end{bmatrix}$$

이 모델은 로봇이 직진과 회전을 동시에 수행하는 경우를 고려한 것이다. 회전에 의한 영향을 반영하기 위해서 $\Delta\theta$ 의 중간 값을 보간하여 방향각 변화의 평균을 위치 계산에 적용하였다. Figure 2에는 회전이 반영되지 않는 예측 모델과 반영된 예측 모델의 시간에 따른 거동을 MATLAB으로 시각화 하였다. Figure 2를 통해 $\Delta\theta$ 를 포함하지 않으면 모션 모델의 부정확한 것을 확인할 수 있었다. 더 나아가 time step을 더 작게 할수록 모델의 정확성이 향상된다는 것을 고려해 볼 수 있다.

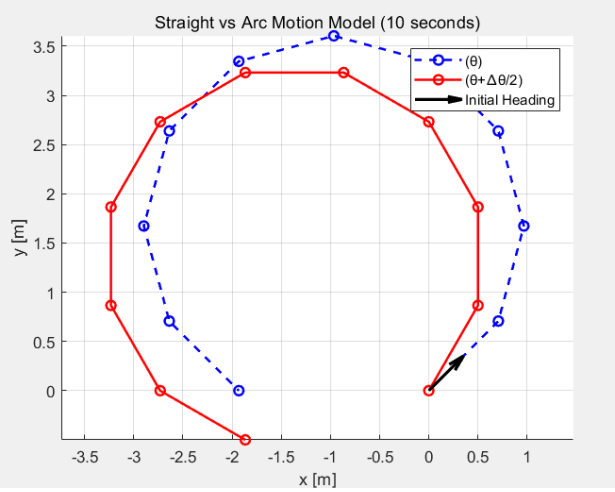

Figure2 Straight vs Arc Motion Model

3.2.2 측정 모델

본 논문에서 측정 모델은 Yaw만 정의된다. 즉 스칼라 형태이고 데이터는 MPU6050의 dmp기능에서 제공된 쿼터니언을 Yaw 값으로 변환하여 사용한다.

$$z_t = h(x_t) + \delta_t$$

3.3 공분산 행렬 구하기

3.3.1 Q

모바일 로봇에 나침반을 부착하고 회전 주행 모습을 카메라로 촬영했다. 이를 통해 일정 time step에서 나침반의 Yaw 변화량과 예측값의 Yaw 변화량 차이로 Q을 계산한다. Odometry는 시스템 모델의 오차, 슬립에 의해 오차가 누적된다. 따라서 누적 영향을 최소화하기 위해 짧은 간격안에서 측정하였다. 측정 조건은 Table 3과 같다. 이러한 누적 오차를 보정하기 위해 필터를

사용하는 것이므로 프로세스 노이즈는 짧은 간격에 대한 데이터로 구해도 충분하다.

Table 3 Mobile Robot Components

Avg time step	0.06s
Number of samples	7개
Avg delta degree of compass	3 deg

Figure 3은 0.42초 동안 7개의 변화량 샘플을 보여준다.

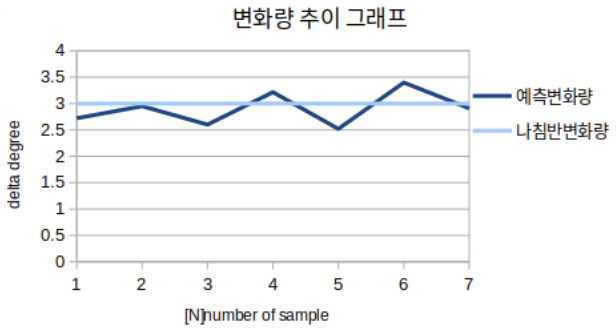

Figure 3 변화량 추이 그래프

이를 통해 분산은 $0.10275\ deg^2$ 으로 계산되었다. 이외의 x, y는 현재 고려대상이 아님으로 1로 가정하였다.

3.3.2 R

IMU도 마찬가지로 drift오차가 있다. 이는 센서 노이즈와 구별되며 bias 상태를 로봇의 상태변수에 포함시키거나 실험적 튜닝을 통해 보안해야 한다. 센서 노이즈 자체만 판별해보기 위해 drift가 뚜렷하게 보이지 않는 구간의 데이터들을 바탕으로 노이즈를 측정하였다.

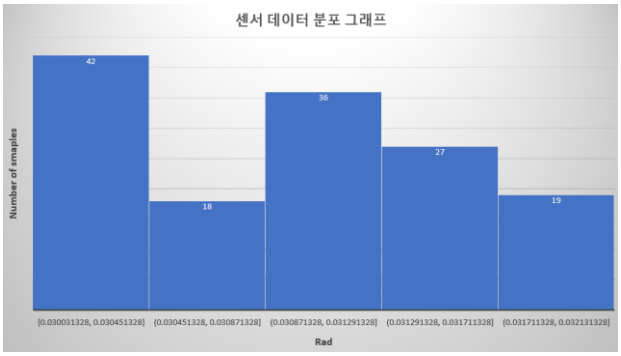

Figure 4 센서 데이터 분포 그래프

Figure 4의 샘플 수는 142개이고 평균은 $-7.349 \times 10^{-8}\ rad$ 이며 분산은 $2.7 \times 10^{-15} rad^2$으로 계산되었다. R이 너무 작게 나와 측정값을 과신할 수도 있어 $2.7 \times 10^{-3} rad^2$을 사용하였다.

4. EKF 알고리즘 코드 작성

4.1 자코비안 행렬

상태 벡터는 다음과 같이 나타낼 수 있다

$$\boldsymbol{x}_k = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} x_{k-1} + \Delta S cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ y_{k-1} + \Delta S sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \theta + \Delta\theta \end{bmatrix}$$

f를 **x**에 대해 편미분한 자코비안 행렬은 다음과 같이 계산된다.

$$F_k = \begin{bmatrix} 1 & 0 & -\Delta S sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 1 & \Delta S cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 1 & 0 \end{bmatrix}$$

4.2 예측 및 업데이트 주요 코드

다음 4.2.1과 4.2.2는 예측과 업데이트에 관한 파이썬 코드를 보여준다. 부록 A에 전체 코드 링크를 첨부하였다.

4.2.1 ekf_prediction Code

```
def ekf_prediction(self):
    if self.latest_odom is None or self.latest_imu is None:
    return

    theta = self.x[2, 0]
    delta_s = self.v_enc * self.dt
    self.delta_theta = self.w_enc * self.dt

    # 상태 전이 자코비안 행렬 F_k
    F_k = np.array([
[1, 0, -delta_s * math.sin(theta + self.delta_theta / 2)],
[0, 1, delta_s * math.cos(theta + self.delta_theta / 2)],
[0, 0, 1]
])

   # 상태 예측: x_k = f(xk-1 , uk)
    self.x[0,0] += delta_s*math.cos(theta + self.delta_theta/2)
  self.x[1,0] += + delta_s*math.sin(theta + self.delta_theta/2)
self.x[2,0]= theta + self.delta_theta

   # 공분산 예측: P_k = F_k * P_k-1 * F_k.T + Q
   self.P = F_k @ self.P @ F_k.T + self.Q
    self.ekf_update()
```

4.2.2 ekf_update Code

```
def ekf_update(self):
    """ EKF 보정 단계 """
    Z = np.array([[self.yaw_imu]])

    # 칼만 이득 K 계산
    S = self.H @ self.P @ self.H.T + self.R
    K = self.P @ self.H.T @ np.linalg.inv(S)

    # 상태 업데이트
    self.x = self.x + K @ (Z - (self.H @ self.x))

    # 공분산 업데이트
    self.P = (np.eye(3) - K @ self.H) @ self.P
```

5. Sensor Fusion Test

5.1 모바일 로봇 구성품

현재 모바일 로봇의 구성요소는 Table 4과 같다.

Table 4 Mobile Robot Components

Encoder DC Motor	JGB32-520
Motor Driver	L298N
IMU	MPU6050
MCU	ESP32 DevkitC V4

5.2 Test Setting

Sensor Fusuion을 통한 Orientation 보정을 진행하기 위해 조건 및 재료들을 Table 5에 기재해 두었다.

Table 5 Experimental Conditions and Materials

Global 좌표	360° 지도
Angular Velocity	1rad/s
Revolutions	4.75바퀴
Visualization Tool	RVIZ

RVIZ에서는 다음과 같은 벡터를 시각화 했다.

(1) 예측 방향 벡터 (Green)

(2) 측정 방향 벡터 (Red)

(3) EKF 후 방향 벡터 (Blue)

5.3 Test Results

Figure5은 초기의 RVIZ와 실제 상태를 보여준다.

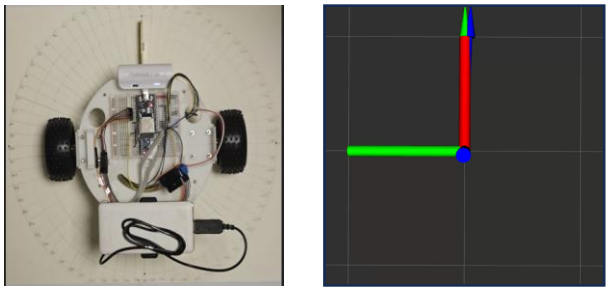

Figure 5 Left: Real World Right: RVIZ

측정 yaw 벡터 기준으로 4바퀴 도는 동안 예측 yaw 벡터와 측정 Yaw 벡터, EKF 후 Yaw 벡터는 Figure3 과 같다.

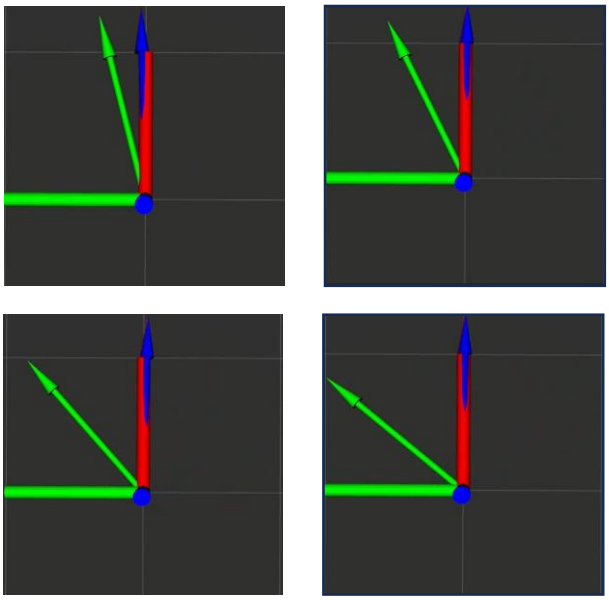

Figure 6vectors for each revolution

Table 6는 각 회전 마다 예측값과 필터 추정값의 차이와 계산된 Kalman gain 값을 보여준다.

Table 6 Comparison of Yaw Angle at each revolution (Left) and Kalman Gain (Right)

11.5 (deg)	0.95114
20.5 (deg)	0.95114
32.9 (deg)	0.95114
45.9 (deg)	0.95114

Figure 7는 실제 IMU을 기준으로 로봇이 4.75바퀴 회전 후 정지했을 때의 모습이다.

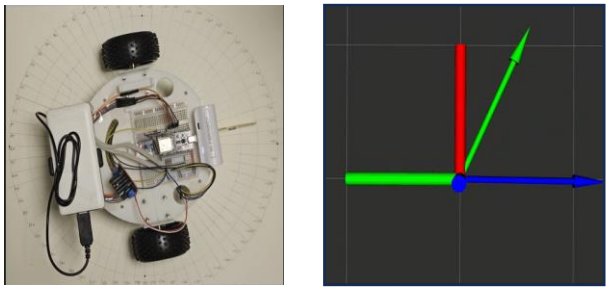

Figure 7 Robot State and Visualization after Completing 4.75 Revolutions

6. 결론 및 향후 계획

6.1 결론

테스트를 통해 두 가지 사실을 확인하였다. 첫째, Odometry의 주행 누적 오차를 확인하였다. Odometry는 슬립, 모션 모델링의 오차 등의 원인으로 주행을 지속할수록 오차가 누적되는 특성이 있다. 이는 Figure 6의 결과를 통해 명확히 드러났다. 따라서 장거리 주행 시 Odometry기반 위치 추정에 대해 반드시 보정이 필요하다. 둘째, 필터의 성능과 한계점을 확인하였다. Figure 7를 통해 실제 로봇의 방향과 필터로 추정된 방향 사이에 약 8°오차를 확인할 수 있었으며 이는 약 55° 오차가 발생한 나는 Odometry보단 우수한 성능이다. 하지만 측정 벡터와 필터 추정벡터가 일치하는 것을 볼 수 있는데 이는 필터가 측정값을 과신하고 있다고 판단할 수 있었다. 그 이유로는 측정 노이즈의 분산 $2.7 * 10^{-3} rad^2$ 이 0.95114 Kalman Gain이 나오는 결과로 이어졌기 때문이다. 따라서 IMU센서의 drift 오차와 노이즈가 크게 반영될 가능성이 높다. 이로 인해 실제 로봇과 필터 사이에 약 8° 의 오차가 발생했음을 짐작할 수 있다.

6.2 향후계획

이번 프로젝트 이후의 향후 계획은 다음과 같다. IMU의 drift를 state에 반영해서 drift 자체를 예측하게 하거나 실험적으로 R값을 증가시키는 방식으로 측정값을 과신하지 않도록 튜닝이 필요해 보인다. 이에 대한 구체적인 방법 조사를 통해 측정 데이터를 과신하지 않도록 해본다. 그리고 IMU를 사용하지 않고 LIDAR 센서를 장착하여 방향뿐만 아니라 위치까지 상태 추정을 위한 Sensor fusion을 진행해 볼 것이다. 이후 FAST SLAM, Particle SLAM 과 같은 SLAM 알고리즘을 구현해보며 두개의 SLAM 알고리즘을 선택하여 성능 비교를 진행해본다.

참고문헌

(1) Sebastian Thrun, Wolfram Burgard, Dieter Fox , 2020, Probabilistic Robotics, pp.27~96
(2) https://taeyoung96.github.io/slam/SLAM_07/
(3) https://soohwan-justin.tistory.com/8

부록

A. Python Code

Full Python implementation is available at:

https://github.com/gongminsu-maker/filter/blob/main/src/mobile_robot_pkg/mobile_robot_pkg/kalman_filter_yaw.py