GPS Only Kalman Filter 완전 해부 - 칼만 필터 기초부터

칼만 필터를 처음 접하시는 것 같으니, 기초부터 차근차근 설명드리겠습니다.

1. 왜 GPS에 칼만 필터가 필요한가?

1.1 GPS의 문제점

```
실제 위치: (37.123456, 127.654321)

GPS 측정값들:
시간 0초: (37.123460, 127.654318) ← 4m 오차
시간 1초: (37.123451, 127.654325) ← 3m 오차
시간 2초: (37.123465, 127.654315) ← 6m 오차
시간 3초: (37.123459, 127.654323) ← 2m 오차
```

GPS는 **항상 노이즈가 섞인 부정확한 측정값**을 줍니다. 그런데 우리는 **정확한 위치와 속도**가 필요

1.2 칼만 필터의 아이디어

> "물리 법칙 + 노이즈가 섞인 측정값 = 더 정확한 추정값"

물리 법칙: "물체는 등속직선운동을 한다"

- 1초 후 위치 = 현재 위치 + 속도 × 1초

측정값: GPS가 주는 노이즈 섞인 위치

결과: 둘을 합쳐서 더 정확한 위치와 속도 추정

---

2. 상태 벡터가 무엇인가?

2.1 상태 벡터의 의미

```
x = [px, py, pz, vx, vy, vz]ᵀ
```

이건 "지금 이 순간 물체의 모든 상태"를 나타내는 벡터입니다.

- **px, py, pz**: 지금 어디에 있는가? (위치)

- **vx, vy, vz**: 지금 어느 방향으로 얼마나 빨리 움직이고 있는가? (속도)

2.2 왜 속도까지 추정하나?

GPS는 위치만 알려주는데 왜 속도까지?

예시로 설명:

```
시간 0초: GPS = (100, 200) m
시간 1초: GPS = (103, 205) m

단순 계산: 속도 = (3, 5) m/s
```

하지만 GPS에 노이즈가 있으면:

```
시간 0초: GPS = (100±5, 200±5) m
시간 1초: GPS = (103±5, 205±5) m

속도 계산이 부정확해집니다!
```

칼만 필터는 속도를 별도로 추정해서 더 정확한 위치 예측을 합니다.

---

3. 칼만 필터의 두 단계

3.1 예측 단계 (Prediction) - "물리 법칙으로 예측"

물리 모델:

```
다음 위치 = 현재 위치 + 속도 × 시간
다음 속도 = 현재 속도 (등속 가정)
```

수식:

```
px(k+1) = px(k) + vx(k) × Δt
py(k+1) = py(k) + vy(k) × Δt
pz(k+1) = pz(k) + vz(k) × Δt
vx(k+1) = vx(k)
```

```
vy(k+1) = vy(k)
vz(k+1) = vz(k)
```

실제 예시:

```
현재 추정 상태 (시간 k):
위치: (100, 200, 50) m
속도: (3, 5, 0) m/s

1초 후 예측 (시간 k+1):
위치: (100+3×1, 200+5×1, 50+0×1) = (103, 205, 50) m
속도: (3, 5, 0) m/s (그대로)
```

3.2 갱신 단계 (Update) - "GPS 측정값으로 보정"

상황:

- **예측값**: (103, 205, 50) m
- **GPS 측정값**: (101, 207, 48) m
- **어느 쪽을 더 믿을까?**

칼만 필터의 해답:

```
최종 추정값 = 가중평균

위치 = α × 예측값 + (1-α) × 측정값

여기서 α는 "어느 쪽을 더 믿는가"의 비율
```

실제 계산:

```
예측값이 더 정확하다면 (α = 0.7):
최종 위치 = 0.7×(103,205,50) + 0.3×(101,207,48)
          = (102.4, 205.6, 49.4) m
```

---

4. 관측 방정식의 의미

4.1 관측 방정식

```
z = H × x + v
```

이게 무슨 뜻이냐면:

z: GPS가 실제로 측정한 값 [gps_x, gps_y, gps_z] **x**: 우리가 추정하는 상태 [px, py, pz, vx, vy, vz] **H**: "상태에서 측정값을 뽑아내는 행렬" **v**: 측정 노이즈

4.2 H 행렬의 의미

```
H = [1  0  0  0  0  0]  ← GPS는 px만 측정
    [0  1  0  0  0  0]  ← GPS는 py만 측정
    [0  0  1  0  0  0]  ← GPS는 pz만 측정
```

즉:

```
[gps_x]   [1 0 0 0 0 0] [px]
[gps_y] = [0 1 0 0 0 0] [py]  + 노이즈
[gps_z]   [0 0 1 0 0 0] [pz]
                        [vx]
                        [vy]
                        [vz]
```

의미: "GPS는 위치만 측정하고, 속도는 직접 측정 안 함"

---

5. 노이즈 모델이 왜 필요한가?

5.1 프로세스 노이즈 Q

"우리 물리 모델이 얼마나 부정확한가?"

등속직선운동 가정의 문제:

- 실제로는 가속, 감속, 방향 전환함
- 바람, 떨림 등의 외부 요인

```
# 예시: 자동차가 갑자기 브레이크
실제: 시속 60km/h → 시속 0km/h
모델 예측: 시속 60km/h → 시속 60km/h (계속 등속)
```

Q 행렬: "모델 예측이 이정도는 틀릴 수 있어"

5.2 관측 노이즈 R

"GPS 측정이 얼마나 부정확한가?"

```
R = [σ²_horizontal    0                0              ]
    [0                σ²_horizontal    0              ]
    [0                0                σ²_vertical    ]
```

의미:

- `σ²_horizontal`: GPS 수평 정확도의 분산 (예: 5m 정확도 → 25)
- `σ²_vertical`: GPS 수직 정확도의 분산 (보통 수평보다 나쁨)

---

6. 칼만 필터 전체 흐름

6.1 한 사이클의 동작

```
# 1단계: 예측 (물리 법칙)
def predict():
    # 다음 위치/속도 예측
    x_pred = F @ x_prev   # F: 상태전이행렬
    P_pred = F @ P_prev @ F.T + Q   # 불확실성도 증가

# 2단계: 갱신 (GPS 측정값)
def update():
    # 칼만 이득 계산
    K = P_pred @ H.T @ inv(H @ P_pred @ H.T + R)

    # 최종 추정값
    x_new = x_pred + K @ (z_gps - H @ x_pred)
    P_new = (I - K @ H) @ P_pred
```

6.2 실제 동작 예시

```
초기 상태:
위치: (0, 0, 0) m, 속도: (0, 0, 0) m/s, 불확실성: 매우 큼

=== 1초 후 ===
1. 예측: 위치 (0, 0, 0), 속도 (0, 0, 0) - 변화 없음
```

```
2. GPS 측정: (102, 198, 49) m
3. 갱신: GPS를 거의 그대로 믿음 (불확실성이 컸으므로)
   → 위치: (102, 198, 49), 속도: (0, 0, 0)

=== 2초 후 ===
1. 예측: 위치 (102, 198, 49), 속도 (0, 0, 0) - 여전히 변화 없음
2. GPS 측정: (105, 203, 48) m
3. 갱신: 위치 변화를 감지하여 속도 추정
   → 위치: (104, 201, 48.5), 속도: (3, 5, -0.5)

=== 3초 후 ===
1. 예측: 위치 (104+3, 201+5, 48.5-0.5) = (107, 206, 48)
2. GPS 측정: (106, 207, 47) m
3. 갱신: 예측과 측정값의 가중평균
   → 위치: (106.5, 206.3, 47.7), 속도: (2.5, 5.3, -0.3)
```

7. 코드에서의 실제 구현

7.1 예측 단계

```kotlin
private fun predict(dt: Double) {
    // F 행렬 구성
    val F = Array(6) { DoubleArray(6) { 0.0 } }
    for (i in 0..5) F[i][i] = 1.0  // 대각선 1
    for (i in 0..2) F[i][i + 3] = dt  // 위치 = 위치 + 속도×시간

    // 상태 예측
    val newState = MatrixUtils.multiply(F, stateVector)
    stateVector = newState

    // 불확실성 증가 (프로세스 노이즈 추가)
    val Q = createProcessNoise(dt)
    val FP = MatrixUtils.multiply(F, errorCovariance)
    val FPFT = MatrixUtils.multiply(FP, MatrixUtils.transpose(F))
    errorCovariance = MatrixUtils.add(FPFT, Q)
}
```

7.2 갱신 단계

```kotlin
private fun updateWithGPS(gpsPosition: RTKPosition) {
    // GPS 위치를 UTM으로 변환
```

```
    val (x, y) = coordinateTransform.latLonToMeters(
        gpsPosition.latitude, gpsPosition.longitude
    )
    val observation = doubleArrayOf(x, y, gpsPosition.altitude)

    // H 행렬: GPS는 위치만 관측
    val H = Array(3) { DoubleArray(6) { 0.0 } }
    H[0][0] = 1.0  // px
    H[1][1] = 1.0  // py
    H[2][2] = 1.0  // pz

    // 관측 노이즈
    val R = Array(3) { DoubleArray(3) { 0.0 } }
    val accuracy = calculateAccuracy(gpsPosition)
    R[0][0] = accuracy * accuracy  // 수평
    R[1][1] = accuracy * accuracy  // 수평
    R[2][2] = accuracy * accuracy * 4  // 수직 (더 부정확)

    // 칼만 이득 계산
    val predicted = MatrixUtils.multiply(H, stateVector)
    val innovation = doubleArrayOf(
        observation[0] - predicted[0],  // x 차이
        observation[1] - predicted[1],  // y 차이
        observation[2] - predicted[2]   // z 차이
    )

    // 최종 갱신
    val K = calculateKalmanGain(H, R)
    val Ky = MatrixUtils.multiply(K, innovation)
    for (i in stateVector.indices) {
        stateVector[i] += Ky[i]  // 보정 적용
    }
}
```

8. 결국 칼만 필터가 하는 일

8.1 한 문장 요약

> **"물리 법칙으로 예측한 값과 센서로 측정한 값을 적절히 섞어서, 둘 다보다 더 정확한 추정값을 만든다"**

8.2 GPS Only 칼만 필터의 효과

Before (Raw GPS):

```
시간 0: (100±5, 200±5) m
시간 1: (103±5, 205±5) m
시간 2: (101±5, 208±5) m ← 뒤로 간 것처럼 보임
시간 3: (107±5, 206±5) m
```

After (Kalman Filter):

```
시간 0: (100, 200) m, 속도 (0, 0) m/s
시간 1: (103, 205) m, 속도 (3, 5) m/s
시간 2: (106, 210) m, 속도 (3, 5) m/s ← 부드럽게 연결
시간 3: (107, 206) m, 속도 (1, -4) m/s ← 방향 전환 감지
```

개선점:

1. **노이즈 제거**: 지글지글한 GPS 궤적이 부드러워짐
2. **속도 추정**: GPS로는 알기 어려운 정확한 속도 계산
3. **예측 가능**: 다음 위치를 미리 예측 가능
4. **일관성**: 물리 법칙에 맞는 합리적인 움직임