SAR 시뮬레이션 RCS 활용

01/21/2026 18:18:24

좋은 요구예요. 이건 **물리적으로 매우 중요한 모델링 단계**이고, 지금까지 만든 SAR 파이프라인과도 정확히 맞아떨어집니다.

아래에서

1. **beam footprint의 물리적 정의**
2. **수학적 판정 조건**
3. **바로 쓸 수 있는 Python 코드 (CPU / SAR 시뮬레이션용)**
4. **지금 SAR 코드에 어떻게 끼워 넣는지**

순서대로 정리합니다.

1️⃣ Beam footprint의 정의 (정확한 물리 모델)

가정 (일반적인 SAR stripmap / spotlight 공통)

- 센서는 위치 `sensor_pos = (xs, ys, zs)`
- 안테나 보어사이트 방향 `boresight_dir` (보통 타깃 중심을 향함)
- **3 dB beamwidth** (half-power beamwidth) `θ_bw` [rad]
- target 위치 `p = (x, y, z)`

Beam footprint 안에 있다는 조건

> *타깃 방향 벡터가 보어사이트와 이루는 각도가 **beamwidth/2** 이내*

즉,

$$\cos^{-1}\left(\frac{(p-s)\cdot\hat{b}}{\|p-s\|}\right) \le \frac{\theta_{\mathrm{bw}}}{2}$$

2️⃣ 판정을 위한 수학적 형태 (연산 최적화)

`acos()`를 쓰지 않기 위해 **cos 비교**로 바꿉니다.

조건식 (권장)

$$\frac{(p - s) \cdot \hat{b}}{\|p - s\|} \geq \cos\left(\frac{\theta_{\text{bw}}}{2}\right)$$

이걸 만족하면 **beam footprint 내부**입니다.

3️⃣ Python 코드 구현 (바로 사용 가능)

아래 코드는:

- 여러 target을 한 번에 처리
- beam 안 → SNR 유지
- beam 밖 → SNR = 0

✅ 핵심 함수: beam footprint 판정

```
import numpy as np

def unit(v):
    return v / (np.linalg.norm(v, axis=-1, keepdims=True) + 1e-12)


def beam_footprint_mask(
    sensor_pos,           # (3,)
    boresight_dir,         # (3,) or None
    targets,              # (N,3)
    beamwidth_deg=10.0,    # 3 dB beamwidth [deg]
):
    """
    Returns:
        mask (N,) boolean
        True  -> target inside beam footprint
        False -> outside beam
    """
    sensor_pos = np.asarray(sensor_pos, dtype=float)
    targets = np.asarray(targets, dtype=float)

    # 보어사이트가 None이면 target 평균 방향으로 자동 설정
```

```
    if boresight_dir is None:
        boresight_dir = np.mean(targets, axis=0) - sensor_pos

    b = unit(boresight_dir)

    # sensor -> target 방향
    v = targets - sensor_pos[None, :]
    v_hat = unit(v)

    # cos(angle)
    cosang = np.sum(v_hat * b[None, :], axis=1)

    # beam threshold
    cos_th = np.cos(np.deg2rad(beamwidth_deg * 0.5))

    return cosang >= cos_th
```

✅ 예시 사용 (질문에서 준 target)

```
sensor_pos = np.array([0.0, -1000.0, 500.0])   # 센서 위치
targets = np.array([
    [0.0, 0.0, 0.0],
    [50.0, 50.0, 0.0]
])

mask = beam_footprint_mask(
    sensor_pos=sensor_pos,
    boresight_dir=None,        # 자동: 타깃 평균 방향
    targets=targets,
    beamwidth_deg=5.0          # 좁은 빔
)

print(mask)
```

출력 예:

```
[ True False ]
```

→ 첫 번째 타깃만 beam footprint 안에 있음

4️⃣ SNR = 0 처리 (SAR 시뮬레이션에 바로 연결)

(A) target-level (점 산란원 모델)

```
snr = np.array([20.0, 20.0])   # dB or linear scale

mask = beam_footprint_mask(
    sensor_pos, None, targets, beamwidth_deg=5.0
)

snr_eff = snr * mask.astype(float)
```

(B) phase history 생성 전에 적용 (권장)

```
# centers: (N,3) facet center or point targets
# sigma:   (N,) RCS or scattering strength

mask = beam_footprint_mask(
    sensor_pos=tx_pos,
    boresight_dir=boresight_dir,
    targets=centers,
    beamwidth_deg=beamwidth_deg
)

sigma_eff = sigma * mask.astype(float)
```

👉 **beam 밖 facet/target은 아예 phase history에 기여하지 않음**

5️⃣ SAR 파이프라인에서의 정확한 위치

이 로직은 **여기 들어가는 게 물리적으로 맞습니다**:

```
[geometry]
   ↓
beam footprint mask   ← ★ 여기
   ↓
```

```
facet sigma 계산
   ↓
shadowing
   ↓
phase history
```

즉,

- shadowing보다 먼저
- sigma 계산과 곱하거나
- sigma 계산 전에 facet 자체를 제거

6️⃣ 현실적인 확장 (다음 단계)

지금 구현은 **binary beam**입니다.
현실적으로는 다음이 더 좋습니다.

(A) Gaussian beam weighting (권장)

```
w = exp( - (θ / θ_bw)^2 )
sigma *= w
```

(B) azimuth-dependent beam steering (spotlight SAR)

- pulse마다 `boresight_dir[p]` 변경

(C) elevation beam 포함 (2D beam)

7️⃣ 한 줄 요약

> ***beam footprint는 "센서 기준 보어사이트 각도"로 정의하고,***
> ***그 안에 있는 target/facet만 SAR 신호에 기여하게 하면 된다.***
>
> *위 코드는 그걸 가장 단순하고 물리적으로 맞게 구현한 것이다.*

원하시면 다음으로:

- **Gaussian beam 패턴으로 SNR 연속 가중**
- **spotlight SAR용 time-varying boresight**
- **terrain/object 각각 다른 beam 모델**

중에서 바로 이어서 구현해드릴게요.