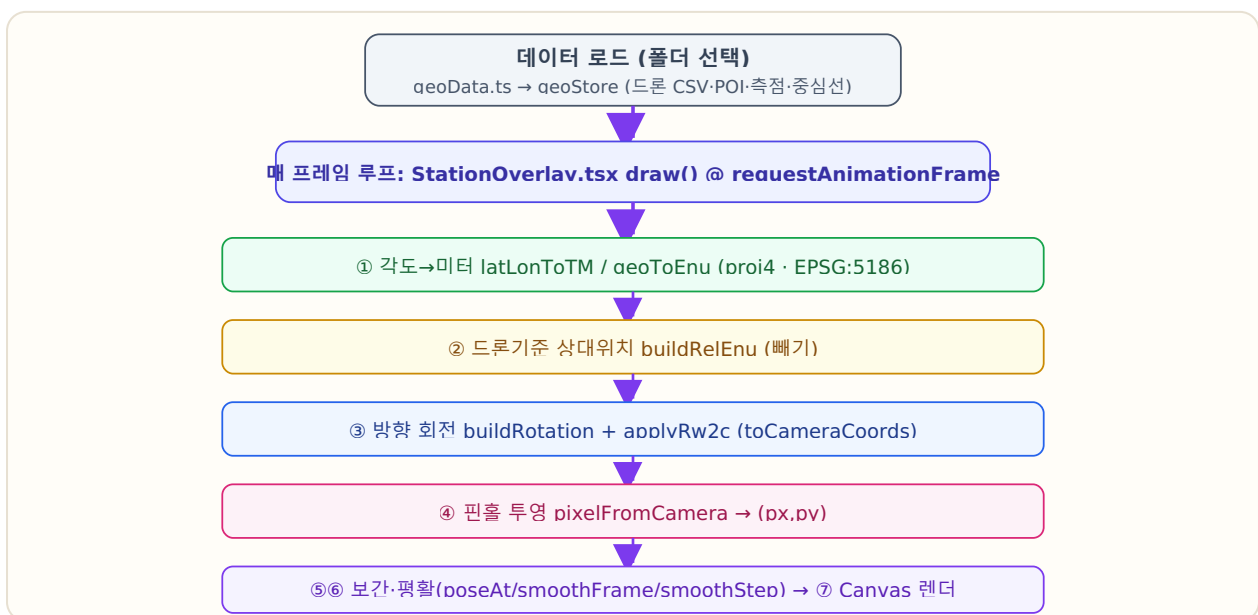


구현상세_드론좌표_영상투영_소스코드매칭

드론 좌표 → 영상 픽셀 투영: 기술 ↔ 소스코드 매칭 (구현 상세)

발표 자료 [발표 핵심기술 드론좌표를 영상에 맞추기 쉬운설명](#) 의 "쉬운 비유"가 실제 어떤 코드로 구현되었는지, 파일·줄 위치와 함께 설명하는 개발자용 문서입니다. 핵심 계산은 대부분 [client/src/utils/geoProjection.ts](#) 한 곳에 모여 있고, 매 프레임 호출·평활·렌더링은 [client/src/components/overlay/StationOverlay.tsx](#) 에 있습니다.

0. 전체 파이프라인 (파일 기준)



아래 표의 기술과 본문의 굵은 용어들은 문서 맨 아래 [용어 사전\(도움말\)](#) 에서 그림과 함께 자세히 풀어 놓았어요. 모르는 말이 나오면 바로 그 항목을 보면 됩니다. (다른 검색 필요 없음!)

단계	기술	파일	핵심 함수	하는 일 (쉽게)
① 위경		geoProjection.ts	latLonToTM, geoToEnu	

단계	기술	파일	핵심 함수	하는 일 (쉽게)
도→ 미터	proj4 · EPSG: 5186			GPS 각도(위도·경도) 를 계산하기 쉬운 미터 지도 로 바꾼다
② 드론 기준 위치	ENU 상 대좌표	geoProjection.ts	<code>buildRelEnu</code>	터널-드론 좌표를 빼 서 "드론에서 동/북/ 위로 몇 m"인지 구한 다
③ 방향 회전	회전행 렬	geoProjection.ts	<code>buildRotation</code> , <code>applyRw2c</code>	카메라 기울기 (yaw·pitch·roll)만큼 방향을 한 번에 돌린 다
④ 화면 투영	핀홀 카 메라 모 델	geoProjection.ts	<code>pixelFromCamera</code>	3D 방향을 납작한 화 면의 점(가로·세로 %) 으로 눌러 담는다
⑤ 사이 채우 기	보간	StationOverlay.tsx	<code>poseAt</code>	사진 30장 사이의 중 간 위치 를 상상해 채 워 부드럽게
⑥ 흔들 림 제거	이동평 균 + EMA	StationOverlay.tsx	<code>smoothFrame</code> , <code>smoothStep</code>	여러 값을 평균 내 드 론 떨림·튐을 없앤다
⑦ 렌더	RAF + Canvas 2D	StationOverlay.tsx	<code>draw</code>	화면 새로 그릴 때마 다($\approx 60/s$) 이름표를 캔버스에 그린다
⑧ fps 자동	프레임 수÷길이	VideoPlayer.tsx	<code>effectiveFps</code>	영상이 1초에 몇 장 인지 스스로 알아내 ①~⑦의 시간 기준을 맞춘다

1. 위경도(각도) → 미터 지도 (proj4 · EPSG:5186)

위치: [geoProjection.ts:74-97](#)

```
proj4.defs('EPSG:5186',
  '+proj=tmerc +lat_0=38 +lon_0=127 +k=1 +x_0=200000 +y_0=600000
  +ellps=GRS80 +units=m +no_defs');
const _toTM = proj4('EPSG:4326', 'EPSG:5186');

function latLonToTM(lat, lon) { // 위경도(각도) → TM(미터)
  const [e, n] = _toTM.forward([lon, lat]); // 주의: proj4 는 [lon,
  lat] 순서
  return [e, n];
}
```

- **무엇:** WGS84 위경도(EPSG:4326) → 한국 TM(EPSG:5186, 중부원점) 미터 좌표로 변환.
- **왜:** 각도로는 거리를 못 재므로, 이후 모든 계산을 미터로 하기 위한 출발점.
- **역방향:** `_toTM.inverse([E,N])` → 다시 위경도 (드래그 보정 `groundPointFromPixel` 등에서 사용).

2. 드론 기준 상대 위치 (ENU 좌표)

위치: [geoProjection.ts:255-273](#) `buildRelEnu`

```
const stEnu = geoToEnu(targetLat, targetLon, targetAlt +
  geoidOffset, ...); // 터널(대상)
const drEnu = geoToEnu(camera.lat, camera.lon,
  camera.altitude, ...); // 드론(카메라)
const drEnuAdj = [drEnu[0]+offX, drEnu[1]+offY, drEnu[2]
  +offZ]; // 위치 미세보정
const relEnu = [stEnu[0]-drEnuAdj[0], stEnu[1]-drEnuAdj[1], stEnu[2]-
  drEnuAdj[2]]; // 빼기
const dist = Math.hypot(relEnu[0],
  relEnu[1]); // 수평거리(m)
```

- **무엇:** 대상과 드론을 미터 좌표로 놓고 빼서 "드론 기준 동/북/상 몇 m"(상대 벡터) 산출.
- **왜:** 카메라에서 본 방향 계산의 입력. (`geoToEnu` 는 [geoProjection.ts:91-97](#))
- **geoidOffset:** 대상 정표고(EL)+지오이드고 → 타원체고. 드론 GPS 고도(타원체고)와 기준을 맞춤(대전≈25.8m).

3. 카메라 기울기 = 회전행렬 (Rotation Matrix)

위치: [geoProjection.ts:275-295](#) (`buildRotation` + `applyRw2c`)

```

function buildRotation(camera, params) { // yaw·pitch·roll → 3×3 행렬
    const yaw = toRad(camera.yaw + params.yawOffset);
    const pitch = toRad(camera.pitch + params.pitch);
    const roll = toRad(camera.roll + params.roll);
    const cy=cos(yaw), sy=sin(yaw), cp=cos(pitch), sp=sin(pitch),
        cr=cos(roll), sr=sin(roll);
    return [[cy*cr+sy*sp*sr, sy*cp, cy*sr-sy*sp*cr], ...]; // R_b2w
}
function applyRw2c(b2w, rel) { // R_w2c · rel → 카메라 좌표
    return { Xc: b2w[0][0]*rel[0]+b2w[1][0]*rel[1]+b2w[2][0]*rel[2],
            Yc: -(b2w[0][2]*rel[0]+...), Zc: b2w[0][1]*rel[0]+... };
}

```

- **무엇:** yaw(좌우)·pitch(상하)·roll(가웃)을 하나의 3×3 행렬로 만들어, 상대벡터를 **카메라 시점 좌표 (Xc,Yc,Zc)** 로 한 번에 회전.
- **왜:** 세 회전을 개별 적용하지 않고 행렬 한 번 곱으로 정확·간결하게. (문서의 "만능 양념장")
- **원본과 동일:** $R_{w2c} = R_{align} \cdot R_{b2w}^T$ (파이썬 `advanced_tuner_v2.py` 이식 — 파일 상단 주석 `geoProjection.ts:1-12`).
- **묶음 함수:** ②+③을 한 번에 = `toCameraCoords (301-317)`. `distH/fwd/side` (거리필터용) 도 여기서 채움.

4. 바늘구멍 사진기 = 핀홀 투영 (초점거리)

위치: `geoProjection.ts:126-137` `pixelFromCamera`

```

export function pixelFromCamera(cc, params) {
    const f = params.focalLen, sW = params.sensorW ?? 36, sH =
        params.sensorH ?? 20.25;
    return {
        pxRaw: (0.5 + params.cx0) + (cc.Xc / cc.Zc) * (f / sW), // 가로 0~1
        pyRaw: (0.5 + params.cy0) + (cc.Yc / cc.Zc) * (f / sH), // 세로 0~1
    };
}

```

- **무엇:** 카메라 좌표 → 화면 정규좌표(0~1). 문서의 "창문 스티커" 계산이 이 두 줄.
- **원근:** Xc/Zc , Yc/Zc — 앞쪽 거리 Zc 로 나누므로 멀수록 가운데로 작게.
- **초점거리:** f/sW , f/sH — f (focalLen)가 클수록(망원) 크게. `sensorH=20.25`는 16:9 기준(=36×9/16).

- **참고:** 원스톱 함수 `projectPoint` (334-427)는 ①~④+FOV판정까지 한 번에 수행(디버그/단건용). 실사용 렌더는 `toCameraCoords` + `pixelFromCamera` 를 프레임마다 호출.

5. 30장 사이 부드럽게 = 보간 (Interpolation)

위치: [StationOverlay.tsx:689-712](#) `poseAt`

```
const poseAt = (estFrame) => { // 연속 프레임번호 → 드론 포즈
  const frac = (estFrame - f1) / (f2 - f1); // 두 실제 프레임 사이 위치 (0~1)
  const L = (x, y) => x + (y - x) * frac; // 선형보간
  let dy = ((b.yaw - a.yaw + 540) % 360) - 180; // 방향은 최단각으로
  return { ...a, lat: L(a.lat, b.lat), lon: L(a.lon, b.lon), yaw: a.yaw + dy*frac, ... };
};
```

- **무엇:** 정수 프레임(사진 30장) 사이의 **중간 드론 위치·자세**를 계산.
- **왜:** 이름표가 30번이 아니라 화면 갱신(≈60번)마다 매끈하게 이동.
- **입력:** 현재 시각 → 연속 프레임번호 `estFrame = estTime * fpsRef.current` ([StationOverlay.tsx:974](#)). `fpsRef` 는 §8에서 주입.

6. 흔들림 제거 = 이동평균 + EMA 평활

위치 A — 원본 데이터 평균: [StationOverlay.tsx:596-620](#) `smoothFrame`

```
// 중심 프레임 기준 ±halfWin 프레임의 GPS·자세를 평균 (회전 경계는 보존)
lat = Σlat/n; yaw = atan2(Σsin, Σcos); // 각도는 sin/cos 평균 후 atan2
```

위치 B — 화면 위치 평활: [StationOverlay.tsx:48-61](#) `smoothStep`

```
if (d > REJECT_DIST && prev.rej < MAX) return {유지}; // 튀는 값(이상치) 무시
const speed = hypot(vx, vy); // 평활된 이동속도
const a = min(maxAlpha, minAlpha + (maxAlpha-minAlpha)*min(1, speed/speedRef));
return { x: prev.x + dx*a, y: prev.y + dy*a, ... }; // 속도적응 EMA
```

- **무엇:** (A) GPS-자세 노이즈를 프레임 평균으로 줄이고, (B) 화면 좌표를 속도적응 EMA로 부드럽게 + 이상치 거부.
- **왜:** 드론 떨림에도 이름표가 안정적으로 붙어 있게. (느릴 땐 강하게 평활, 빠를 땐 즉시 추종)

7. 매 프레임 렌더 = requestAnimationFrame + Canvas 2D

위치: [StationOverlay.tsx:931-1243](#) `draw` 루프

```
const draw = () => {
  rafId = requestAnimationFrame(draw); // 화면 갱신마다
  반복(≈60/s)
  const dronePose = poseAt(estFrame); // §5 보간 포즈
  // POI 마다:
  const cc = toCameraCoords(dronePose, poiA.lat, poiA.lon, pz, params,
    worldOrigin); // §2+§3
  const { pxRaw, pyRaw } = pixelFromCamera(cc, params); // §4
  const d = smoothStep(prevDpoi, pxRaw, pyRaw, ...); // §6 화면 평활
  ctx.strokeText(label, lx, labelY); ctx.fillText(...); // Canvas 에 이
  름표 그림
};
```

- **무엇:** 화면이 새로 그려질 때마다 §2~§6을 다시 계산해 **Canvas 2D**에 이름표·중심선을 그림.
- **호출 지점:** 축점 라벨 [1070-1072](#), POI/구조물 라벨 [1107-1109](#).
- **성능:** 무거운 "가시집합/겹침 판정"은 별도 사전계산(`requestIdleCallback`, 파일 상단 주석), RAF는 투영·그리기 위주.

8. 영상별 fps 자동 산출

위치: [VideoPlayer.tsx:272-286](#) `effectiveFps`

```
const effectiveFps = useMemo(() => {
  if (!storeFrames.length || !duration) return 30000/1001; // 폴백 29.97
  let maxF = 0; for (const f of storeFrames) if (f.frame > maxF) maxF =
    f.frame;
  const raw = maxF / duration; // 마지막 프레임번
  호 ÷ 영상길이
  const STD = [23.976, 24, 25, 29.97, 30, 50, 59.94, 60];
  let best = STD[0], bd = Math.abs(raw-STD[0]);
  for (const s of STD) { const d = Math.abs(raw-s); if (d<bd)
    {bd=d;best=s;} }
  return bd <= best*0.1 ? best : raw; // 표준값 ±10%면
  스냅
}, [storeFrames, duration]);
```

- **무엇:** 드론 CSV에는 시간이 없고 프레임 번호(`frame_cnt`)만 있으므로, **영상 길이**와 결합해 fps 산출 후 표준값에 스냅.

- 연결: `<StationOverlay fps={effectiveFps} />` → [StationOverlay.tsx:260](#) `fpsRef` → 55의 `estFrame`, 최근접 프레임 탐색의 기준.
- 전제: 드론 CSV가 영상 전 구간을 덮는다고 가정(마지막 `frame_cnt` ≈ 영상 끝). 부분만 덮으면 스냅 실패 시 원시값 사용.

부록. 데이터 로딩 (폴더 → 스토어)

위치: [geoData.ts](#) → [geoStore.ts](#)

- `<input webkitdirectory>` 로 고른 폴더의 `<base>.csv` (드론), `_POI.csv`, `building/*.csv` (측점·교량·터널·구교)를 파싱.
- 드론 CSV 헤더:
`frame_cnt,latitude,longitude,altitude,yaw,pitch,roll,focal_len` ([geoData.ts:139-172](#)).
- 인코딩 자동감지(UTF-8 BOM / EUC-KR), KMZ(zip)는 `fflate` 로 해제.
- ENU 기준원점: `getWorldOrigin` ([geoData.ts:576](#)) → 스토어 `origin` → 투영에 `ref` 로 전달.

관련 함수 빠른 색인

함수	파일:줄	역할
<code>latLonToTM</code> / <code>geoToEnu</code>	<code>geoProjection.ts</code> : 81 / 91	각도 → 미터, ENU
<code>buildRelEnu</code>	<code>geoProjection.ts</code> : 255	드론 기준 상대 벡터
<code>buildRotation</code> / <code>applyRw2c</code>	<code>geoProjection.ts</code> : 275 / 289	회전 행렬. 적용
<code>toCameraCoords</code>	<code>geoProjection.ts</code> : 301	②+③ 묶음
<code>pixelFromCamera</code>		

함수	파일:줄	역할
	geoProjection.ts: 126	핀홀 투영
<code>projectPoint</code>	geoProjection.ts: 334	①~④ 원스 톱(디 버그)
<code>worldFromPixel</code> / <code>groundPointFromPixel</code> / <code>solveZForPixelY</code>	geoProjection.ts: 149 / 218 / 194	역투 영(드 래그 보정)
<code>poseAt</code> / <code>smoothFrame</code> / <code>smoothStep</code>	StationOverlay.tsx: 689 / 596 / 48	보간· 평활
<code>draw</code>	StationOverlay.tsx: 931	RAF 렌더 루프
<code>effectiveFps</code>	VideoPlayer.tsx: 272	fps 자동 산출

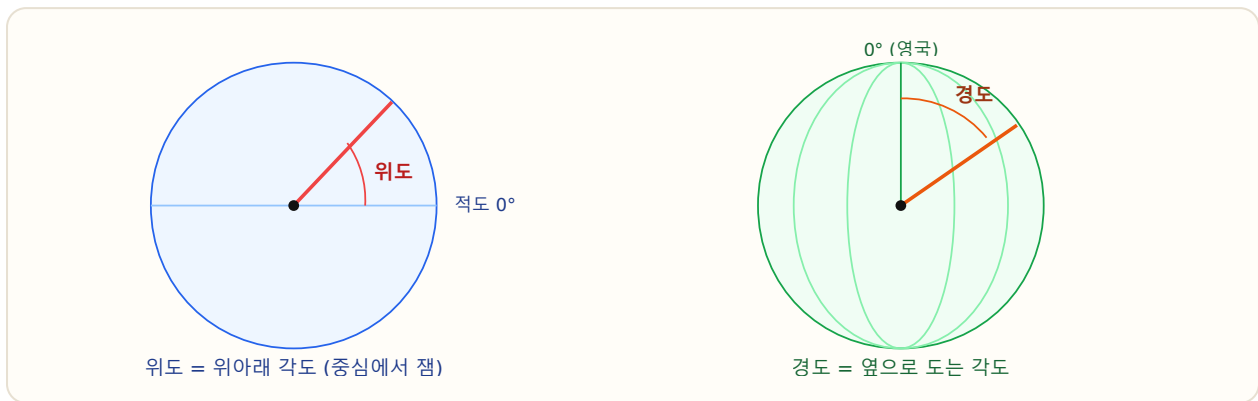
용어 사전 (도움말)

본문에 나온 전문용어를 **초등학생도 이해할 수 있게** 그림과 함께 풀었어요. 모르는 말이 나오면 여기만 보면 돼요. (다른 검색 필요 없음!)

▶ 위도·경도 (latitude / longitude)

지구 위의 위치를 나타내는 두 개의 각도예요. 지구 중심에서 각도기로 잴 각(도, °) 이에요.

- **위도** = 적도(0도)에서 **위/아래로** 몇 도 (북극 90도, 남극 -90도)
- **경도** = 영국(0도)에서 **옆으로 빙 둘러** 몇 도 (동쪽으로 갈수록 커짐)



둘 다 지구 '중심'에서 잴 각도(도). 그래서 단위가 미터가 아니라 '도(°)'다.

▶ WGS84 · EPSG:4326

- **WGS84**: 전 세계 GPS가 쓰는 위도·경도 표준. "지구를 이런 모양·기준으로 본다"는 세계 공통 약속.
- **EPSG**: 세상의 여러 좌표계에 번호표를 붙여 정리한 목록 (도서관 책번호 같은 것).
- **EPSG:4326** = 그 목록에서 **WGS84(위도·경도)** 에 붙은 번호. 코드에서 'EPSG:4326' 이라 쓰면 "위경도 방식"이란 뜻.

▶ EPSG:5186 · 한국 TM (횡축 메르카토르)

- 우리나라 전용 평평한 미터 지도 좌표계. 위경도(각도)를 미터(거리) 로 바꾼 결과가 이 좌표.
- **TM = Transverse Mercator(횡축 메르카토르)**: 둥근 지구에 원통을 옆으로 눕혀 싹뿔 펴는 방법. 원통이 닿는 세로선(경도) 근처가 가장 정확 → 남북으로 길쭉한 한국에 딱.



원통이 '닿는 빨간 선' 근처가 가장 정확하다. 한국은 세로로 길어서 세로선에 맞추는 TM을 쓴다.

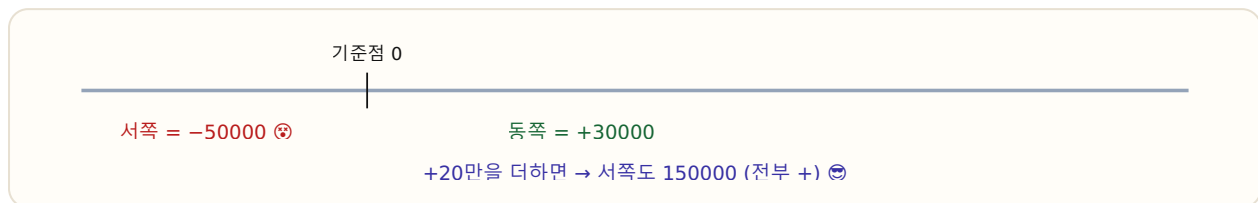
코드의 설정 쪽지 뜻 (+proj=tmerc +lat_0=38 +lon_0=127 +k=1 +x_0=200000 +y_0=600000 +ellps=GRS80):

설정	뜻
proj=tmerc	TM(횡축 메르카토르) 방식

설정	뜻
lat_0=38 lon_0=127	지도 기준점 = 위도38·경도127 (한국 한가운데)
k=1	크기 1배(줄임/늘림 없음)
x_0=2000000 y_0=6000000	false easting/northing (아래 항목 참고)
ellps=GRS80	지구를 GRS80(살짝 찌그러진 굴 모양)으로 봄
units=m	단위 = 미터

▶ false easting / false northing (x_0, y_0)

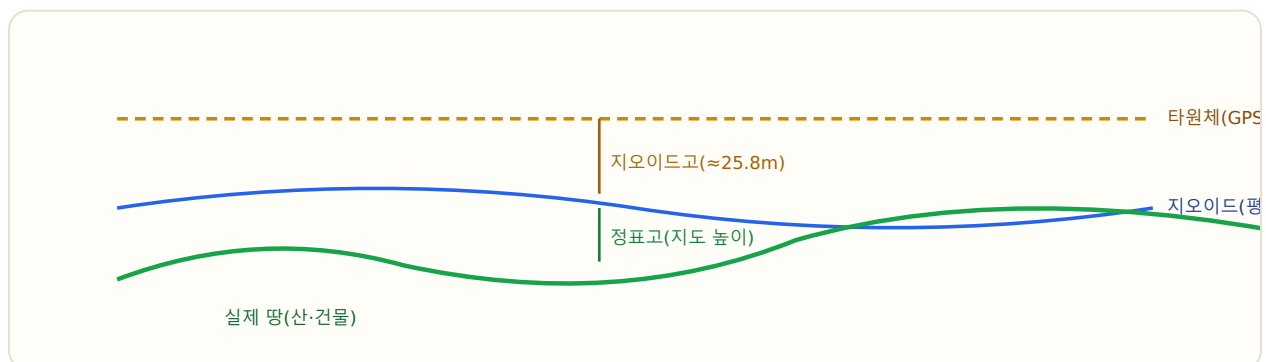
기준점에서 서쪽·남쪽으로 가면 좌표가 **마이너스(-)** 가 돼요. 계산이 헛갈리니까, **처음부터 큰 수를 더해 어디서나 플러스(+)** 가 되게 해요. (한국은 가로 +20만, 세로 +60만 m)



온도에 273을 더해 '절대온도(K)'로 음수를 없애는 것과 같은 아이디어.

▶ 높이 3형제 — 정표고 · 지오이드고 · 타원체고 (geoidOffset)

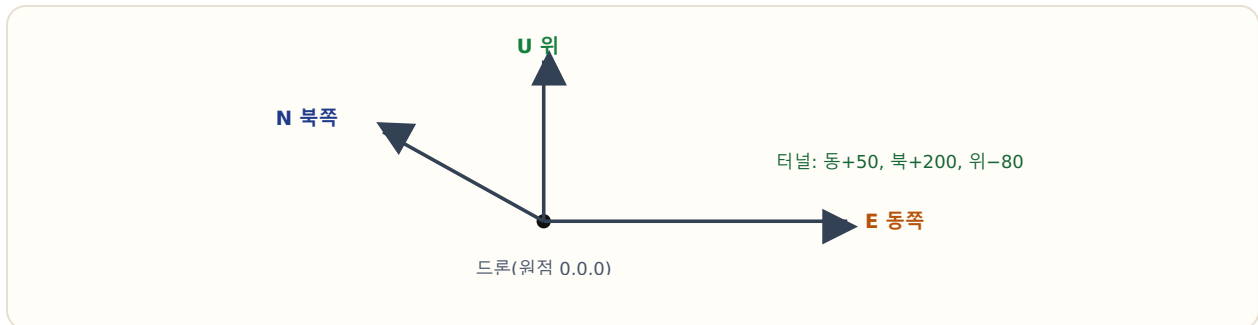
높이를 재는 **기준면이 3가지**라서 서로 변환이 필요해요. 드론 GPS 높이와 지도 높이의 **기준을 맞추려고** geoidOffset (대전≈25.8m)을 더해요.



정표고(우리가 아는 '해발 높이') + 지오이드고 = 타원체고(GPS 높이). 코드는 이 변환으로 기준을 맞춘다.

▶ ENU 좌표 (East-North-Up, 동·북·상)

한 점(기준원점)을 중심으로 **동쪽(E)·북쪽(N)·위(U)** 세 방향으로 "몇 m"인지 나타내는 방식. 드론을 원점에 놓으면 터널이 "동 +50, 북 +200, 위 -80" 처럼 표현돼요(본문 ②).



기준원점(getWorldOrigin)에서 동·북·위로 잰 미터 좌표. 빼기로 '드론 기준 상대위치'를 만든다.

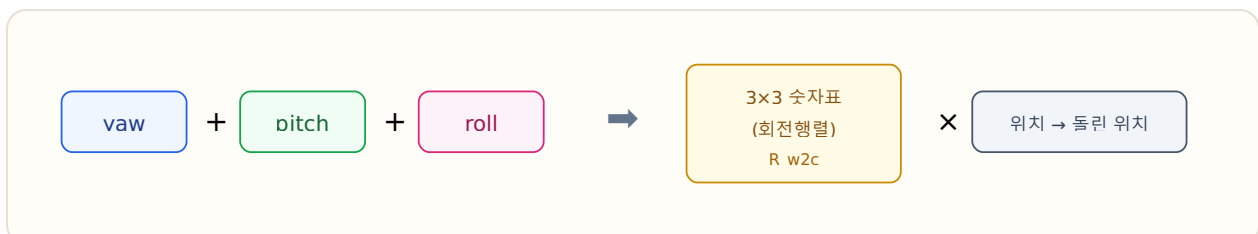
▶ yaw · pitch · roll (카메라가 기운 3방향)

카메라(또는 드론)가 향한 방향을 나타내는 **3개의 회전**. 사람 고개 움직임과 같아요.



▶ 회전행렬 (Rotation Matrix)

위 3개 회전(yaw·pitch·roll)을 **하나의 작은 숫자표(3×3)** 로 미리 합쳐 둔 것. 이 표를 위치에 **한 번 곱하면** 세 방향 회전이 **동시에** 적용돼요. (문서의 "방향 돌리기 만능 양념장")



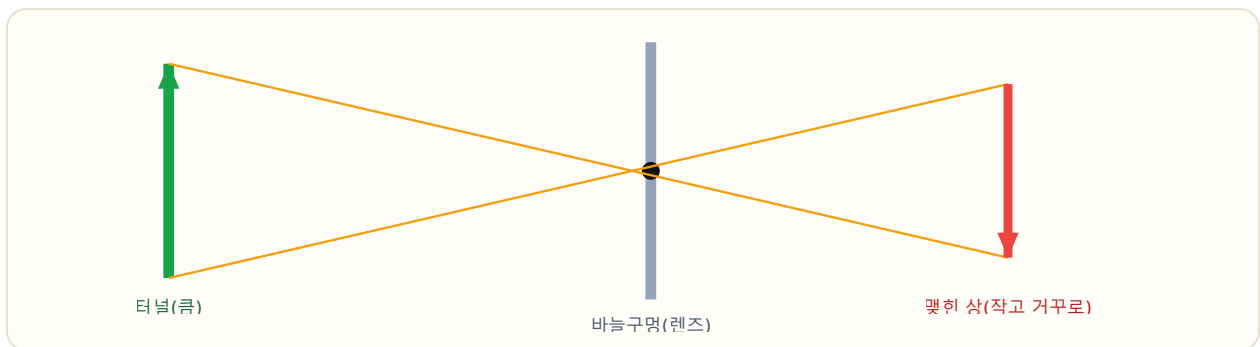
3번 돌릴 걸 표 1개로 합쳐 한 번에 곱한다 → 빠르고 실수 없음. 코드: buildRotation + applyRw2c.

▶ 카메라 좌표 (X_c , Y_c , Z_c)

회전까지 마친 뒤 얻는, **카메라 눈 기준 좌표**. Z_c = 앞쪽 거리, X_c = 좌우, Y_c = 상하. Z_c 가 0보다 커야(앞에 있어야) 화면에 보여요. 다음 단계(핀홀)에서 이 값을 X_c/Z_c , Y_c/Z_c 로 나눠 써요.

▶ 핀홀 카메라 모델 · 초점거리 · 센서

바늘구멍 사진기 원리. 작은 구멍(렌즈)을 지나며 3D가 화면에 맺혀요. 멀면 작게, 가까우면 크게(원근).



모든 빛이 구멍 한 점을 지나 X자로 교차 → 화면엔 작고 거꾸로 맺힘. '초점거리(focal length)'가 클수록 (망원) 크게 보인다.

- **초점거리(focalLen)**: 렌즈가 얼마나 '당겨 찍나'(망원↔광각). 클수록 화면에서 크게.
- **센서(sensorW/H)**: 카메라 필름 크기. 초점거리와 센서 비율이 화각을 정함(코드 f/sW , f/sH).

▶ 정규좌표 (0~1)

화면 위치를 픽셀(예: 1920) 대신 **비율(0~1)**로 나타낸 것. **0=왼쪽/위**, **1=오른쪽/아래**. 화면 크기가 달라져도 그대로 쓸 수 있어 편해요. (예: 가로 0.6 = 화면의 60% 지점)

▶ 보간 (Interpolation)

두 값 **사이의 중간값**을 계산해 채우는 것. 사진이 30장뿐이어도 사이사이를 상상해 채워 **부드럽게** 움직여요.

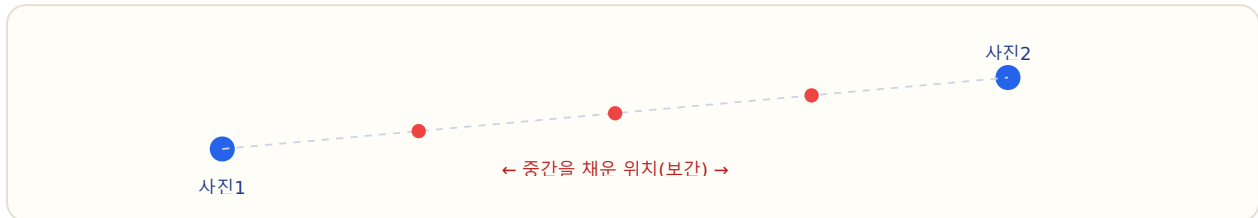
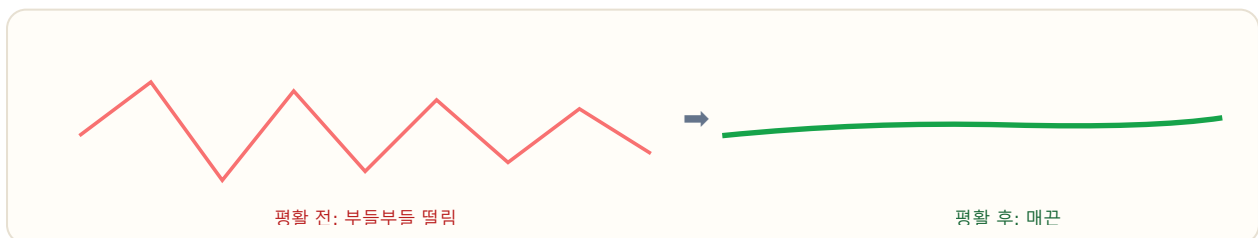


사진1:2 사이 몇 % 지점인지(frac) 계산해 중간 위치를 만든다. 코드: poseAt.

▶ 이동평균 · EMA (지수이동평균)

여러 값을 **평균** 내 들쭉날쭉(떨림)을 매끈하게 만드는 방법.

- **이동평균**: 앞뒤 몇 개를 평균 (코드 `smoothFrame`)
- **EMA**: 최근 값에 더 무게를 둔 평균, 반응이 빠름 (코드 `smoothStep`)



드론 흔들림으로 튀는 위치를 평균으로 눌러 이름표가 안 떨리게 한다.

▶ requestAnimationFrame (RAF)

브라우저에게 "화면 새로 그릴 때마다 이 일을 해줘" 라고 부탁하는 명령. 보통 1초에 약 60번 실행돼요. 그래서 이름표 위치를 매번 다시 계산·그리기 해 부드럽게 따라다녀요.

▶ Canvas 2D

웹페이지 안의 **그림판(도화지)**. 여기에 선·글자를 직접 그려요. 우리는 중심선·측점·POI 이름표를 매 프레임 이 캔버스에 그려요. (선명하고 빠름)

▶ fps · 프레임

- **프레임** = 영상의 **낱장 사진** 한 장.
- **fps(frames per second)** = **1초에 몇 장** 넘기나. 예: 30fps = 1초에 30장.
- 프레임 번호를 시간으로 바꾸려면 fps가 필요해요: `시간 = 프레임번호 ÷ fps` (본문 ⑧).

▶ requestIdleCallback

브라우저가 **한가한 틈**에 무거운 일을 시키는 명령. 무거운 "어떤 라벨을 보여줄지" 준비 작업을 여기서 미리 해 두고, 매 프레임(RAF)에는 가벼운 그리기만 → 60번/초에도 안 버벅여요.