장한진

실시간 관제 시스템 아키텍트 · Hands-on Tech Lead

현장의 요구를 운영 시나리오로 옮기고, 시스템 구조로 만듭니다.
Device–Backend–Cloud–App 전 구간을 설계하고 직접 구현해 납품까지 책임집니다.

> **납품한 시스템 4종 전부 현재 운영 중 · 공군 작전도로 전기요금 −70% · 자율주행·국방·산업·농업 4개 도메인**

010-4793-4444 · anjdi1004@gmail.com · **janghanjin.com** · 전주
여주대학교 관광일본어과 졸업 · 학점은행제 컴퓨터공학 학사 2027년 졸업예정

소개

실시간 관제 시스템을 만들어 온 개발 4년차입니다. 자율주행 골프카트 플릿, 카고크레인 안전, 공군 작전도로, 농업 방제로봇과 대형 건조기까지 현장은 매번 달랐지만 하는 일은 같았습니다. 현장 장비에서 올라오는 데이터를 운영자가 판단할 수 있는 형태로 만드는 일이고, 장비 연동부터 백엔드와 운영 화면, 배포까지 전 구간을 맡아 왔습니다.

고객이 말하는 요구와 시스템이 해야 할 일은 같지 않습니다. 그래서 요구를 먼저 운영 시나리오로 옮깁니다. 하루가 어떻게 흘러가고, 누가 언제 무엇을 보고, 뭐가 잘못되면 누가 개입하는지. 그걸 정한 다음에 시스템 구조를 뽑습니다. 순서를 바꾸면 반드시 다시 만들게 됩니다.

설계만 하고 넘기지 않습니다. 핵심 구간은 직접 짜고, 나머지는 단계로 나눠 맡긴 뒤 완료 기준으로 확인합니다. 실패가 성공처럼 기록되는 구조를 가장 경계합니다.

Skill

Device 장비 연동	MQTT(EMQX), Modbus / PLC, TCP 소켓, LoRa, RTSP
Backend 수집·처리·상태	Python, FastAPI, WebSocket, PostgreSQL / TimescaleDB, Redis, InfluxDB, Graphite, pydantic, aiomqtt
Cloud · Infra 배포·운영	AWS(ECS, RDS, S3, Cognito), Docker, GitHub Actions, 폐쇄망 구축·운영
App 운영 화면	React, Next.js, Preact, Svelte, maplibre (관제 웹) · React Native, Flutter, PWA (모바일)
설계 · 방법	시스템 아키텍처, 연동 규격 정의, 데이터 모델, 운영 시나리오 설계, pytest 기반 완료 기준

경력

(주)비* 2024.09 ~ 현재 · 책임연구원 / 팀장

관제 시스템 개발 소프트웨어 회사. 고객사 장비를 관제하는 시스템을 설계·구축·납품한다.

- **자율주행 골프카트 플릿 관제 시스템** — 제조사·연구기관 규격 협의 창구, 펌웨어 버전별 프로토콜 정규화 설계, 명령 라이프사이클·차간거리 판정 설계, 맵 제작·발행 파이프라인 설계
- **카고크레인 · 안전관리자 관제 시스템** — 크레인 센서값 전량 수집(하중·각도 등), 위험 상황 판정·알림, 현장 안전관리자용 모바일 앱(배치 크레인 현황·서류 관리) 구축. **양산 탑재 · 상용 운영 중**
- **농업 방제로봇 관제 시스템** — 방제량·이동속도·작업 구간 추적, MQTT over LTE 수집 파이프라인 구축, 제조사별 로봇을 단일 규격으로 연결. **현재 운영 중**
- 그 외 — 농작업 대행 매칭 플랫폼 브랜딩·마케팅, 협회·기관 홈페이지 3건 구축

Python · FastAPI · MQTT/EMQX · WebSocket · PostgreSQL/TimescaleDB · Redis · AWS · Preact · Svelte · Next.js · Flutter · Docker

(주)씨* 2022.09 ~ 2024.08 · 선임연구원

열선 제조업 회사. 자사 장비를 원격으로 제어·관제하는 시스템을 개발했다.

- **공군 작전도로 · 비행장 도로열선 관제 시스템** — 적설·습기·온도·수막 센서 수집, on/off로만 동작하던 열선을 센서 기반 자동 제어로 전환. **전기요금 70% 절감, 현재까지 운영 중**
- **컨테이너 대형 건조기 제어 시스템** — 건조기 10여 대 태블릿 제어·관제, 온습도 기반 최적 건조 레시피 반복 제어, 이상 상태 감지 기반 예지보전
- **공공기관 크롤링 서비스** (사내) — 180개 기관 공고 수집 자동화. **현재까지 사용 중**

Python · Modbus/PLC · TCP · LoRa · RTSP · InfluxDB · Graphite · 폐쇄망

경력기술서

(주)비* 2024.09 ~ 현재 · 책임연구원 / 팀장

자율주행 골프카트 플릿 관제 시스템 2025.03 ~ 현재

골프장에서 운전자 없이 코스를 도는 카트들의 플릿 관제 시스템이다. 골프카트 제조사와 국책연구기관이 함께하는 자율주행 사업에서 관제 전 구간을 담당했다. 아키텍처·프로토콜·데이터 모델 설계를 소유하고, 제조사·연구기관 사이에서 규격 협의 창구를 맡았다.

TECH STACK

Python, FastAPI, MQTT(EMQX), WebSocket, PostgreSQL, AWS(ECS·RDS·S3·Cognito), Docker, Preact, maplibre

담당업무

- **기획**: 관제 요구사항 정의, 운영 시나리오 설계, 화면 정보구조 설계
- **설계**: 시스템 아키텍처, MQTT 연동 프로토콜, 데이터 모델, 도메인 규칙, 완료 기준
- **협의**: 카트가 올릴 센서 항목·주기·형식, 상향/하향 토픽 구조, 명령체계를 제조사·연구기관과 협의해 정의. 연동 명세 작성·발송
- **개발**: 관제 프론트엔드(Preact·maplibre), 카트 태블릿 HMI(PWA) 직접 구현. 백엔드는 단계 분해 후 위임·리뷰
- **운영**: 배포 파이프라인·게이트 설계(AWS), 실차 연동 검증

두 버전의 펌웨어를 동시 수용하는 프로토콜 정규화 계층 설계

- **문제**: 펌웨어 v1과 v2.1 두 버전의 카트가 같은 서버에 동시 접속한다. 토픽 구조와 페이로드 형식이 서로 다른데, 현장에 나간 펌웨어는 바꿀 수 없고, 신규 카트를 구버전 규격에 맞추면 새 설계가 과거에 묶인다.
- **해결**: 차량 쪽 수정을 요구하는 대신 버전 차이를 서버가 흡수하기로 결정했다. 수신을 단일 지점으로 모아 토픽과 본문으로 규격을 판별하고 하나의 정규 계약으로 변환해, 도메인 로직은 버전을 모르게 했다. 구버전 코덱은 동결하고, 신버전 코덱은 정의 파일에서 생성해 코드 수정 없이 확장되게 했다. 같은 지점에서 메시지 중복 제거와 시퀀스 역행 거부를 처리했다.
- **결과**: 버전이 다른 카트가 같은 서버에서 함께 운용된다. 전환 작업은 7단계로 분해해 팀원에게 배정했다.

명령 유실이 성공으로 기록되던 구조를 상태 분리로 재설계

- **문제**: 서버가 하행 토픽을 구성하면서 카트의 펌웨어 버전을 가정하고 있었다. 신버전 카트에 구버전 주소로 명령이 발행되어 797건이 유실됐는데, 발행 자체는 성공했으므로 로그에는 정상으로 남았다. 정상 로그만 봐서는 알 수 없는 유실이었고, 수신 거부된 메시지를 따로 쌓던 로그를 분석하는 과정에서 발견했다.
- **해결**: 하행 주소를 카트가 상행에서 선언한 규격에 따라 결정하도록 바꿔 서버가 버전을 추측하지 않게 했다. 미지원 명령은 발행 전에 거부하고, 무응답은 6초 시간초과로, 발행 실패는 실패로 각각 분리해 기록했다. 버려진 수신 메시지는 사유별로 집계했다.
- **결과**: 보낸 것과 도착한 것이 분리되어 기록된다. 여기서 세운 '조용한 폴백 금지' 원칙을 명령·수신·맵 적용·배포 전반에 같은 방식으로 적용했다.

관제는 정보 제공, 안전 책임은 차량 — 경계를 규격으로 명시

- **문제**: 카트는 자기 앞차가 어디 있는지 모르고, 플릿 전체를 보는 것은 서버뿐이다. 그렇다고 서버가 속도를 직접 제어하면 통신이 끊기는 순간 서버가 위험 요소가 된다.
- **해결**: 차간거리를 명령이 아니라 응답을 요구하지 않는 정보 스트림으로 정의하고, 서버 정보만으로 폐루프 제어를 하지 않는 것을 명시 원칙으로 삼았다. 판정은 카트 좌표를 코스 경로선에 투영해 얻은 경로상 진행 위치를 기준으로 앞차를 정하는 방식이고, 두 코스가 같은 물리 도로를 공유하는 구간이 있어 다른 코스의 카트도 경로에 투영해 횡방향 4m 이내면 후보에 포함했다. 판정 결과는 셋으로 나눴다. 앞차 있음과 없음은 매 주기 명시해 보내고, 판정할 수 없을 때는 아무 값도 보내지 않았다. 모름을 없음으로 말하지 않기 위해서다.
- **결과**: 차량 제어기는 이 스트림을 주행 보조 입력으로만 쓴다. 오래된 위치를 근거로 뒤 카트에 지시가 나가는 경로도 함께 차단했다.

재부팅하면 설정을 잊는 장비를 전제로 한 상태 조정 루프 설계

- **문제**: 설정을 한 번 내려보내고 끝내면 카트가 재부팅한 순간부터 서버가 아는 상태와 실제가 어긋난다. 어긋났다는 사실은 아무도 모른다.
- **해결**: 운영 상태의 소유권을 서버에 두고, 설정 변경·카트 재접속·코스 변화 세 시점에 재하달하도록 설계했다. 주기적인 배경 보정을 더해 세 시점을 놓친 경우도 결국 맞춰지게 했다. 재하달에는 안전 게이트를 뒀다 — 속도 배율은 주행 중에는 반영하지 않고 정차 시에만 적용하고, 경로 변경은 홀 경계에서만 적용한다.
- **결과**: 카트가 기억하지 못해도 서버가 기준을 쥐고 맞춘다. 상태 변경이 주행 중 거동을 예고 없이 바꾸는 경로를 막았다.

맵 제작·발행 파이프라인과 3층 검증 설계

- **문제**: 지도가 잘못되면 카트가 가면 안 되는 곳으로 간다. 원본 좌표는 당초 드론 정사촬영으로 확보하려 했지만 오차가 컸다. 그리고 지도를 편집하는 것은 엔지니어가 아니라 운영자다. 사람의 주의력에 기대는 방식은 언젠가 실패한다.
- **해결**: 원본 좌표는 카트를 실제 주행시켜 취득하는 방식으로 바꿨다. 차량이 주행하는 좌표계와 지도의 좌표계가 같아져 변환 오차가 사라진다. 그 위에 맵 에디터에서 경로·속도·홀 지정·대기 구역을 제작하고, 검증을 세 층으로 설계했다. 구간이 충돌하는 편집은 편집 단계에서 아예 만들 수 없게 차단하고, 연결성 등은 자동 검사하며, 마지막으로 실제 카트의 회전 반경과 하드웨어 특성을 동일하게 반영한 시뮬레이터로 주행해 확인한다. 도면상 문제없지만 실제로는 돌 수 없는 커브를 여기서 잡는다. 맵은 버전을 붙여 발행하고, 해당 맵이 없는 카트는 응답이 없어 시간초과로 드러난다.
- **결과**: 규칙 검사는 지도가 말이 되는지를, 시뮬레이터 주행은 그 지도로 실제로 어떻게 움직이는지를 본다. 시뮬레이터를 개발 도구가 아니라 검증 장비로 뒀다.

감속 명령의 의미 불일치를 규격 확정으로 해결

- **문제**: 감속 명령이 양쪽에서 다르게 해석되고 있었다. 관제는 "10km/h에서 20% 감속"을 미리 계산해 절대값 8km/h로 내려보냈는데, 카트는 받은 값에 다시 비율 계산을 적용하고 있었다. 양쪽 다 자기 기준으로는 정상이라 값만 봐서는 오류로 보이지 않았고, 로그를 대조하는 과정에서 충돌을 발견했다.
- **해결**: 한쪽 숫자를 고쳐 맞추는 대신 의미를 확정해야 하는 사안으로 판정했다. 감속 명령을 비율값으로 통일해 계산 주체를 카트로 확정하고, 그 정의를 연동 규격에 명시했다.
- **결과**: 같은 종류의 충돌이 재발할 경로가 막혔다. 이후 규격을 정할 때 값의 형식보다 값의 의미를 먼저 확정하는 것을 원칙으로 삼았다.

결과 실차 3대로 시험장과 실제 골프장 코스를 주행해 제조사·연구기관의 검증 완료 판정을 받았다. 파일럿 운영을 앞두고 있으며, 프로토콜·관제 코어·맵 파이프라인이 가동 중이다. 로그 전송 정책과 현장 인력 배정은 설계를 마치고 구현 전이다.

카고크레인 · 안전관리자 관제 시스템 2025.02 ~ 2026.02

이미 판매되고 있는 크레인 모델에 관제 시스템을 추가로 개발한 프로젝트다. 크레인의 위험 상황을 판정해 운전자와 현장 안전관리자에게 알린다.

TECH STACK

Python, MQTT, LTE, PostgreSQL / TimescaleDB, Redis, Svelte, Flutter, AWS, Docker, CI/CD

담당업무 — 개발 전 구간 1인 수행

- **기획**: 관제 요구사항 정의, 안전관리자 앱 기능 범위 결정
- **설계**: 수집 파이프라인, 위험 판정 로직, 알림 체계, 앱 구조
- **협의**: 크레인 제조업체와 연동 규격 협의
- **개발**: 백엔드, 관제 프론트엔드(Svelte), Flutter 기반 안전관리자 앱 전부 직접 구현
- **운영**: AWS 배포, CI/CD 구성, 시운전 현장 직접 참여(설치는 제조사 담당)

출고 장비를 개조하지 않는 수집 구조 설계

- **문제**: 대상 크레인은 이미 상용 판매 중인 모델이었다. 센서를 새로 붙이거나 장비를 개조하면 이미 출고된 물량에는 적용할 수 없고, 원가와 인증에도 영향이 간다.
- **해결**: 제어기가 이미 산출하고 있는 값을 그대로 받아오는 방향으로 잡았다. 상용 제어기가 하중과 각도를 포함한 센서값을 이미 전부 수집하고 있었기 때문에, 새로 붙일 것이 없었다. 연동 규격은 크레인 제조업체와 맞춰 나갔다.
- **결과**: 장비를 바꾸지 않고 기존 판매 모델에 관제를 추가할 수 있게 됐다.

통신량 제약에 맞춘 수집 주기 결정

- **문제**: 이동식 장비라 LTE로 데이터를 올려야 했고, 센서값 전량을 상시 올리기에는 회선 부담이 컸다.
- **해결**: 먼저 프로토콜 경량화를 검토했지만 적용하지는 못했다. 대신 수집 주기를 1Hz로 잡았다. 초당 1회면 하중 변화를 놓치지 않으면서 통신이 안정적으로 유지되는 선이었다.
- **결과**: 1Hz 수집으로 현재까지 안정적으로 운영되고 있다.
- **남은 과제**: 프로토콜 경량화는 끝내지 못했다.

시계열과 관계형 데이터를 한 저장소로 통합

- **문제**: 센서값은 1Hz로 계속 쌓이는 시계열이고, 크레인 정보와 사용자, 서류는 관계형이다. 시계열 전용 저장소를 따로 두면 데이터가 두 곳으로 갈라지고, 조회와 운영도 따라서 나뉜다.
- **해결**: PostgreSQL에 TimescaleDB를 얹어 한 저장소에서 처리했다. 시계열은 시간 기준으로 분할해 조회 속도를 확보하고, 크레인 정보나 서류 같은 관계형 데이터는 같은 DB에 그대로 두어 조인이 가능한 상태를 유지했다.
- **결과**: 데이터가 쌓여도 기간 조회 속도가 유지되고, 저장소는 하나만 운영하면 된다.

제조사 정격표 기반 위험 판정, 관제 역할을 통지로 한정

- **문제**: 위험 여부를 판정하려면 기준이 필요한데, 크레인 안전 기준을 관제 쪽에서 새로 만드는 것은 위험하다. 제어기의 전복 방지 로직은 물리적 한계만 막을 뿐, 정격 범위를 넘겨 작업하는 것은 걸러지지 않았고 그 사실이 크레인 밖으로 전달되지도 않았다.
- **해결**: 제조업체 정격표를 그대로 기준으로 삼아 하중 초과와 각도로 판정했다. 물리적 전복 방지는 제어기가 이미 맡고 있으므로, 관제의 역할은 정격 초과의 판정과 통지로 한정하고 결과를 크레인 운전자와 현장 안전관리자 양쪽에 보냈다.
- **결과**: 제어는 장비가 하고 관제는 알린다는 경계가 분명해졌다. 안전 기준의 근거는 제조사 정격표로 일원화된다. 위험 상황은 장비 안에 머무르지 않고 현장 인력에게 도달한다.

산업 현장 기준으로 화면 UX 방향 전환

- **문제**: 화면 설계가 일반 소비자용 플랫폼 앱의 문법을 따라가고 있었다. 시각적으로는 정돈돼 보였지만 정보 구조가 뒤집히거나 흐려지는 지점이 생겼다. 현장에서 쓰는 관제 화면은 보기 좋은 것보다 지금 무엇이 어떤 상태인지가 먼저 읽혀야 한다.
- **해결**: 심미성보다 판독성을 앞에 두자고 제안했다. 조금 투박하더라도 정보 위계가 흐트러지지 않는 쪽, 현장에서 바로 이해되는 쪽으로 기준을 잡고 디자이너와 방향을 맞췄다.
- **결과**: 현장에서 긍정적인 반응을 얻었다. 이후 자율주행 카트 관제에서도 같은 기준으로 화면을 잡았다.

반복 서류 업무를 앱으로 통합

- **문제**: 안전관리자는 크레인별 안전 서류를 그때그때 받아야 했다. 운전자 쪽에서는 현장에 갈 때마다 요구받는 서류가 대체로 비슷한데도 매번 사람이 직접 주고받는 일이 반복됐다.
- **해결**: 자주 쓰는 서류를 앱에 저장해 두고, 운전자는 저장된 것을 보내기만 하면 되도록 했다. 안전관리자는 앱에서 바로 받는다. 위험 알림과 같은 앱에 넣었다. 현장 인력이 용도별로 앱을 오가지 않는다고 봤기 때문이다.
- **결과**: 서류 전달이 앱 안에서 끝난다. 안전 알림을 받는 앱과 서류를 주고받는 앱이 같아서, 안전관리자가 하나만 열어두면 된다.

결과 양산 모델에 탑재되어 판매되고 있으며, 현재 상용 운영 중이다.

농업 방제로봇 관제 시스템 2024.12 ~ 2025.03

농업 방제로봇의 작업 상태를 원격 관제하는 시스템이다. 고정 레일형과 노지 주행형, 서로 다른 두 유형의 로봇을 하나의 연동 규격으로 연결했다.

TECH STACK

Python, MQTT, LTE, PostgreSQL / TimescaleDB, Next.js(관제 웹), Flutter(현장 앱)

담당업무 — 개발 전 구간 1인 수행

- **설계**: MQTT 기반 연동 규격, 수집 파이프라인, 데이터 모델
- **협의**: 로봇 제조사에 프로토콜 추가·커스텀을 요청하고 연동 규격 전달
- **개발**: 백엔드, 관제 화면(Next.js), 현장 앱(Flutter) 직접 구현

제조사가 맞춰 오는 연동 규격 설계

- **문제**: 로봇마다 제조사 프로토콜이 제각각이고, 유형도 고정 레일형과 노지 주행형으로 달랐다. 제조사별로 서버에 어댑터를 만들면 로봇이 늘어날 때마다 서버가 함께 커진다.
- **해결**: MQTT 기반 연동 규격을 정의하고, 제조사 쪽에 프로토콜을 추가·커스텀해 규격에 맞춰 올리도록 요청했다. 프로토콜의 버전 관리와 유지도 제조사 몫으로 확정했다. 서버는 규격만 지키면 어느 제조사의 로봇이든 받는다.
- **결과**: 유형이 다른 로봇 3대를 같은 서버에 연결했다. 새 제조사의 로봇도 규격만 맞추면 서버 수정 없이 연결된다.

방제량과 이동속도를 묶어 구간별 살포 상태 산출

- **문제**: 방제량만 봐서는 약액이 고루 뿌려졌는지 알 수 없다. 같은 양이라도 이동이 빠르면 얇게, 느리면 짙게 뿌려진다.
- **해결**: 방제량과 이동속도, 작업 구간을 함께 추적해 구간별 살포 상태를 계산했다. 판정 기준은 로봇 제조사 기준을 따랐다.
- **결과**: 약액이 고루 뿌려졌는지를 구간 단위로 확인할 수 있게 됐다. 축적된 작업 데이터는 살포 효율을 분석하는 기초 자료가 됐다.

결과 현재까지 운영 중이다. 두 유형의 로봇을 단일 규격으로 연결한 구조는 이후 프로젝트의 연동 설계로 이어졌다.

(주)씨*** 2022.09 ~ 2024.08 · 선임연구원

공군 작전도로 · 비행장 도로열선 관제 시스템 2022.09 ~ 2023.09

공군 작전도로와 비행장에 설치된 결빙 방지 열선을 원격 제어·관제하는 시스템이다. 병사가 현장까지 내려가 켜고 끄던 열선을 도로 센서 기반의 자동 제어로 전환했다. 군 폐쇄망 안에 구축해 현재까지 운영 중이다.

TECH STACK

Python, Modbus, RTSP(CCTV·열화상), InfluxDB, React, 유선 LAN(LoRa에서 전환), 폐쇄망

담당업무 — 개발 전 구간 1인 수행

- **설계**: 수집·제어 파이프라인, 가동 구간 설계(예열·잔열 반영), 관제 화면 정보구조
- **개발**: 백엔드, 관제 화면(React) 직접 구현
- **운영**: 폐쇄망 현장 구축, 외부망·휴대폰이 차단된 서버실에서 리눅스 콘솔로 직접 작업, 현장 출입 대응

병사가 700m를 오가며 조작하던 열선을 원격 자동 제어로 전환

- **문제**: 열선 조작 수단이 현장 스위치뿐이라, 병사가 약 700m를 내려가 켜고 껐다. 결빙을 놓치면 안 되는 도로라 넉넉하게 켜 두는 쪽으로 운영될 수밖에 없었고, 전기요금이 그대로 쌓였다.
- **해결**: 센서가 로거를 거쳐 PLC로 모이고 서버가 PLC에서 Modbus로 읽는 구조를 잡아, 적설·습기·온도·수막 값을 수집하고 임계값 기반으로 자동 가동하도록 바꿨다. 결빙 판정 임계값은 발주처가 정한 기준을 그대로 적용했다. 안전 기준을 관제 쪽에서 새로 만들지 않는다는 원칙은 이후 크레인 프로젝트에서도 유지했다.
- **결과**: 전기요금 70% 절감. 현재까지 운영 중이다.

예열과 잔열을 계산에 넣어 가동 구간을 앞뒤로 줄임

- **문제**: 열선은 켜자마자 뜨거워지지 않고, 끈다고 바로 식지도 않는다. 결빙이 시작된 뒤에 켜면 늦고, 다 녹은 뒤에도 켜 두면 낭비다.
- **해결**: 임계 조건에 도달하기 전에 선제 가동해 예열을 확보하고, 종료는 반대로 앞당겨 끊은 뒤 잔열로 마저 녹이도록 가동 구간을 잡았다.
- **결과**: 가동 시간이 실제 필요한 구간으로 줄었다. 70% 절감의 실체가 이 가동 구간 설계다.

열화상 카메라와 CCTV로 판정과 독립된 확인 경로 확보

- **문제**: 요금을 줄이는 변경은 전부 "덜 켜는" 쪽인데, 못 켜서 결빙되면 작전도로와 활주로가 마비된다. 실패 비용이 비대칭이라 센서와 판정 로직만 믿을 수 없었다.
- **해결**: RTSP 기반 CCTV를 거의 실시간으로 관제 화면에 띄우고, 열화상 카메라로 도로 표면 온도를 측정해 함께 표시했다. 센서 판정과 별개로 실제 노면 상태를 사람이 확인할 수 있는 독립 경로를 뒀다.
- **결과**: 자동 제어를 도입하면서도 결빙 위험을 판정 로직 하나에 걸지 않게 됐다.

LoRa를 유선으로 걷어내고, 통신 두절 시 자동 차단을 설계

- **문제**: 처음에는 LoRa로 구축했지만 안정적이지 않았다. 제어 명령이 오가는 회선이라 통신 불안은 곧 제어 불안이었다.
- **해결**: 유선 LAN을 포설해 전환했다. 함께 PLC에 가동 타이머를 둬, 통신이 끊겨도 설정 시간이 지나면 PLC가 자체 판단으로 열선을 내리게 했다. 폴백 판단을 서버가 아니라 현장 제어기에 둔 것이다. 통신이 죽었을 때 켜진 채 방치되는 경로가 막힌다.
- **결과**: 전환 이후 통신 문제는 없었다.

결과 공군 부대에 납품되어 현재까지 운영 중이다.

컨테이너 대형 건조기 제어 시스템 2023.09 ~ 2024.08

컨테이너형 대형 건조기 10여 대를 태블릿으로 제어·관제하는 시스템이다. 고추 건조는 문을 여는 순간 상품 가치가 갈리기 때문에, 감에 의존하던 건조를 데이터 기반 반복 작업으로 바꾸는 것이 목표였다.

TECH STACK

Python, TCP 소켓(펌웨어 바이너리 패킷 직접 파싱), Graphite, React Native → Flutter(전환)

담당업무 — 개발 전 구간 1인 수행

- **설계**: 수집·제어 파이프라인, 레시피 반복 제어 구조, 이상 감지 로직
- **개발**: 백엔드(펌웨어 패킷 수신·파싱), 태블릿 제어·관제 앱 직접 구현

전문가의 레시피 탐색을 데이터로 받치고, 시스템은 재현을 맡음

- **문제**: 건조 품질의 기준은 소프트웨어가 아니라 식품 지식의 영역이다. 어떤 온습도 곡선이 좋은 고추를 만드는지를 시스템이 정할 수는 없다.
- **해결**: 레시피는 식품 전공 연구소장이 담당하고, 시스템은 온습도 데이터를 수집·시각화해 접점을 찾는 근거를 제공했다. 연구소장이 데이터를 보며 찾아낸 레시피를 시스템이 그대로 반복 재현하도록 만들었다.
- **결과**: 감에 의존하던 건조가 데이터로 접점을 찾고, 확정된 레시피를 매번 같게 반복할 수 있게 됐다.

진동·온도 데이터로 부품 이상을 가동 중에 조기 감지

- **문제**: 건조 중 장비가 고장 나면 그 배치 전체의 상품 가치가 떨어진다. 고장 난 뒤에 알면 늦다.
- **해결**: 온도 이상과 진동을 함께 관측해 내부 부품의 이상 징후를 가동 중에 잡았다. 실제로 한쪽 면의 순환이 약해진 징후가 데이터에 잡혀 가동을 멈추고 점검했고, 순환팬 불량을 확인해 교체했다.
- **결과**: 고장이 건조 배치를 망치기 전에 부품을 교체한 실사례를 확보했다. 예지보전이 구호가 아니라 동작으로 확인됐다.

태블릿 앱을 React Native에서 Flutter로 전환

- **문제**: 태블릿 앱을 React Native로 시작했는데, 현장 태블릿 환경에서 성능과 라이브러리 지원, 업데이트 유지 측면을 다시 볼 필요가 생겼다.
- **해결**: 현장 요구 기준으로 비교 평가한 끝에 Flutter로 전환했다.
- **결과**: 이후 크레인 안전관리자 앱은 처음부터 Flutter로 구축했다. 전환에서 얻은 판단이 다음 프로젝트의 기본값이 됐다.

현장 제어 무조건 우선 원칙

- **문제**: 태블릿(현장)과 원격이 같은 장비를 제어하면 명령이 충돌할 수 있다.
- **해결**: 현장 제어가 무조건 우선하도록 설계했다. 장비 앞에 있는 사람의 판단을 원격이 덮어쓸 수 없다.
- **결과**: 제어 충돌의 우선순위가 규칙으로 고정됐다.

결과 고추 건조 현장에 실배치되어 현재까지 안정적으로 운영되고 있다.