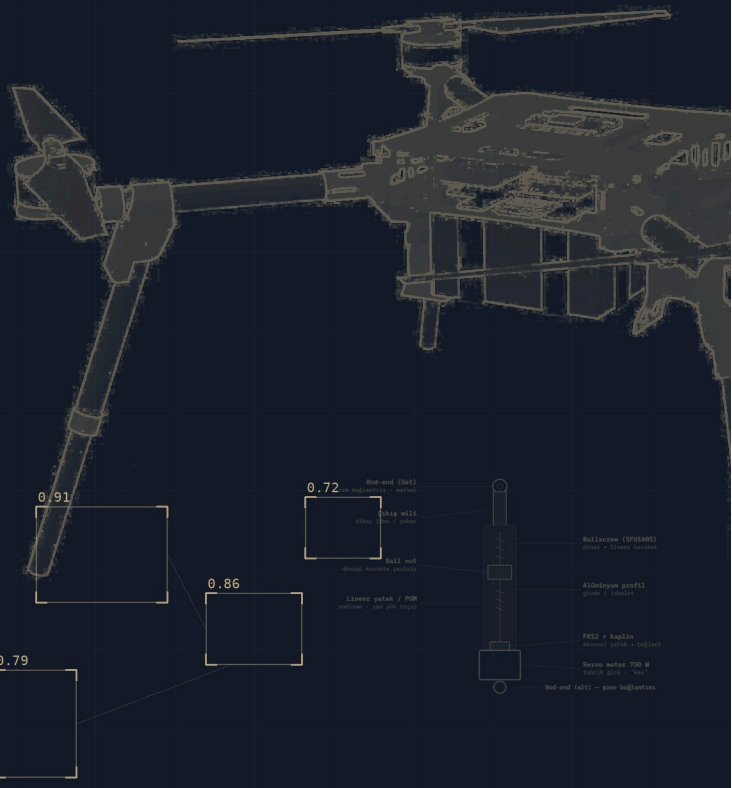
PROJECT PORTFOLIO · AUGUST 2026



Serdar Anıl Demirbaş

Software Engineer · Real-time image processing, computer vision and autonomous systems.

Since 2021 I have been building embedded image processing pipelines, target detection and tracking algorithms and operator interfaces in the defence industry.

0,904	5	5	9
MAP50 · AERIAL OBJECT DETECTION	ENGINEER SOFTWARE TEAM · TEAM LEAD	YEARS IN THE DEFENCE INDUSTRY	PROJECTS IN THIS PORTFOLIO

SERDAR.ANIL33@GMAIL.COM · LINKEDIN.COM/IN/SANILDEMIRBAS · ANKARA, TÜRKİYE

OVERVIEW

What I do

I build software on a mechatronics engineering foundation. My specialism is **real-time image processing** and **getting that processing to run on embedded hardware**: detection and tracking algorithms, dataset preparation and model training, operator interfaces, and porting all of it onto resource-constrained platforms such as NVIDIA Jetson.

CONTENTS

TECHNOLOGIES I WORK WITH

Python · C++ · C# · MATLAB · OpenCV · YOLOv11 · ByteTrack · PyTorch · Albumentations · TensorRT · CUDA C · OpenMP · ROS 2 · NVIDIA Jetson · Raspberry Pi · ESP32 · Qt Creator · Unity · Django · Next.js · PostgreSQL · Redis · Docker · TCP/UDP/UART

PROJECT 01

Autonomous UAS Image Processing and Mission Software

ROLE	ORGANISATION	PERIOD	TEAM	COMPETITION
Software Team Lead	Atılım UAV	2025 – Present	5-engineer software team	RoboNation SUAS 2026

KA-FA1500 · the unmanned aerial vehicle the team built for the competition

PROBLEM

RoboNation SUAS requires the aircraft to fly autonomously at 20–40 m altitude, detect and classify targets on the ground and act on them according to the mission. The difficulty: at that altitude targets occupy only 80–120 pixels in the frame, and detection has to run at near real time on an embedded computer with a tight power budget.

MY RESPONSIBILITY

- Architecting the target detection/classification, autonomous navigation and mission planning software
- Leading the five-engineer software team
- Defining the data preparation pipeline and the training hyperparameters
- The Technical Design Report and the rules-compliant team website

0,904	0,691	~15.000	Sep 2026
MAP50	MAP50-95	TRAINING IMAGES	TULSA, OKLAHOMA

Target detection from 20–40 m altitude

Validation output of the trained YOLOv11m model, with confidence scores across different altitudes, ground surfaces and lighting

0,904	0,691	~15.000	33
MAP50	MAP50-95	TRAINING IMAGES	CONVERGENCE EPOCH

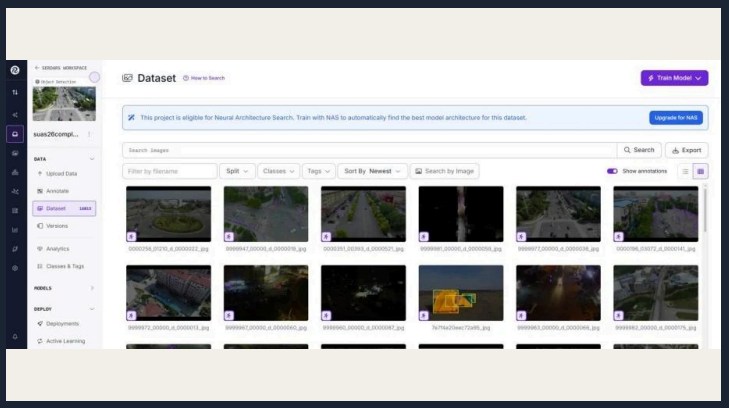
1 · DATA PREPARATION

To bring raw 4K drone frames down to YOLO's fixed input size I designed a pipeline that splits them into **1280×1280 tiles with 40% overlap**. The overlap matters: when a target falls on a tile boundary it guarantees the target appears whole in at least one tile. I made the train/validation split **at image level**: if tiles from the same scene end up on both sides, the model learns through leakage and the validation metric lies.

2 · CLASS BALANCING

The raw data was class-imbalanced. Using offline augmentation I applied a **per-class multiplier**: 8× for the under-represented class, 3× for the over-represented one. That turned ~2,100 raw frames into a balanced ~15,000-image training set at a 1:1.2 ratio. I deliberately left colour augmentation to training rather than this step: applied in both layers it becomes double augmentation.

From dataset to embedded inference



Managing and versioning the labelled dataset

Training loop and loss curve

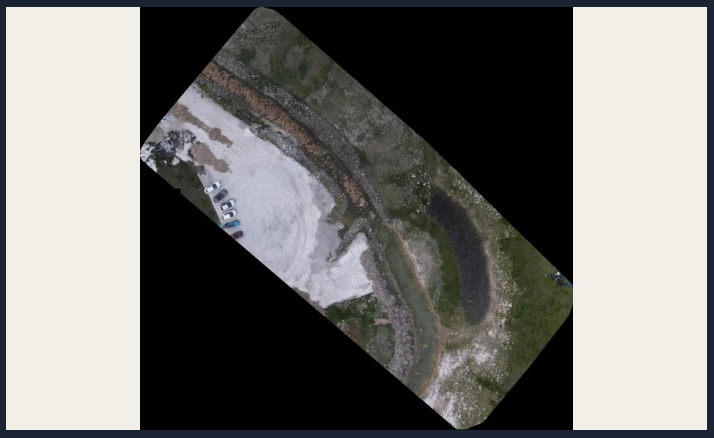
3 · TRAINING

Transfer learning on YOLOv11m. The model converged at epoch 33 and early stopping fired at 48, which means **the data ceiling was reached**: what is needed is more varied data, not more epochs. Having made that diagnosis I wrote a 13-item improvement plan: higher input resolution, collecting data across more altitudes and lighting conditions, and partial-occlusion augmentation.

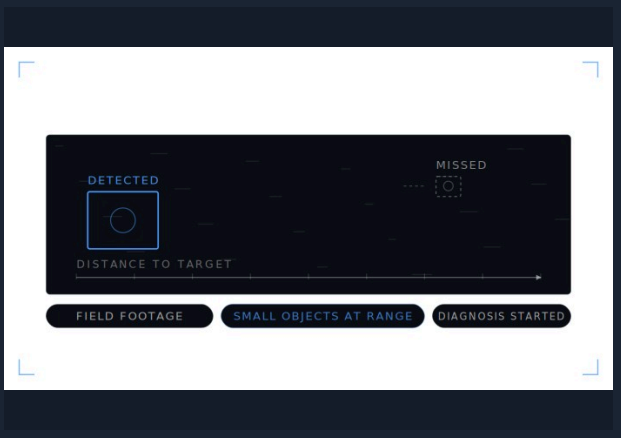
4 · EMBEDDED DEPLOYMENT

The model was positioned to run in real time on NVIDIA Jetson via **TensorRT FP16**. Training and export resolution have to match here; left different, the model behaves unpredictably in the field.

YOLOv11m · Albumentations · PyTorch · TensorRT · Jetson · OpenCV · ROS 2

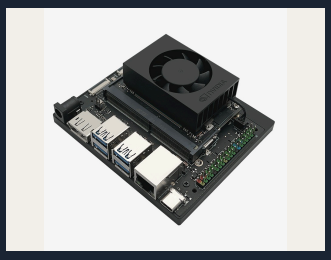


Post-flight orthomosaic: individual frames stitched into a single map



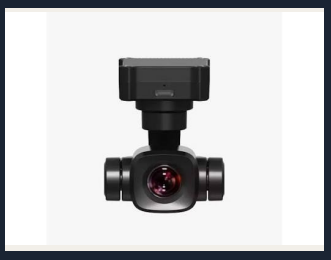
Detection performance against range

Avionics and compute architecture



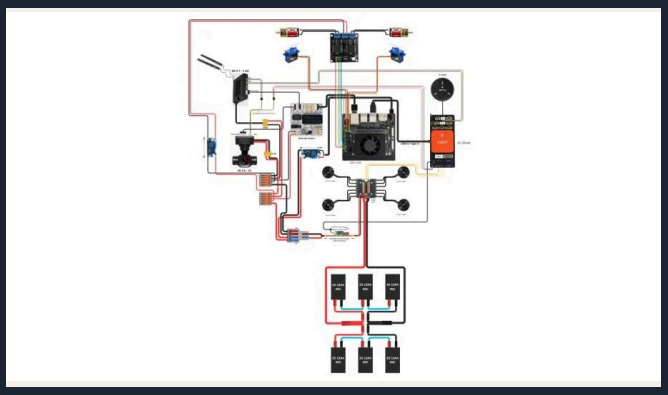
Jetson Orin NX
Companion vision computer

Cube Orange+
Flight controller

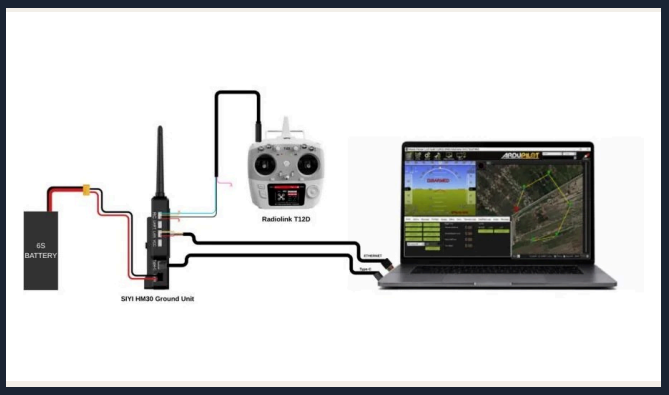


SIYI A8 Mini
Gimballed camera

T-Motor U7
Propulsion



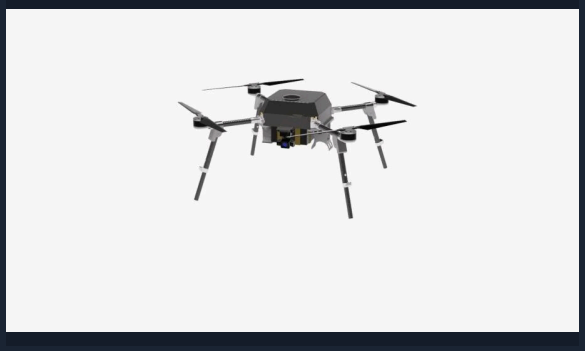
Power distribution and wiring architecture



Ground station communication chain: HM30 datalink, transmitter, mission planner

A SOFTWARE ARCHITECTURE DECISION

We split image processing and flight control across separate processors: the flight controller does nothing but fly, the companion computer carries the vision workload, and the two talk over ROS 2. That separation keeps flight safety unaffected when the vision load spikes, which is one of the most critical design decisions in an autonomous system.



KA-FA1500 · integrated design

Payload housing

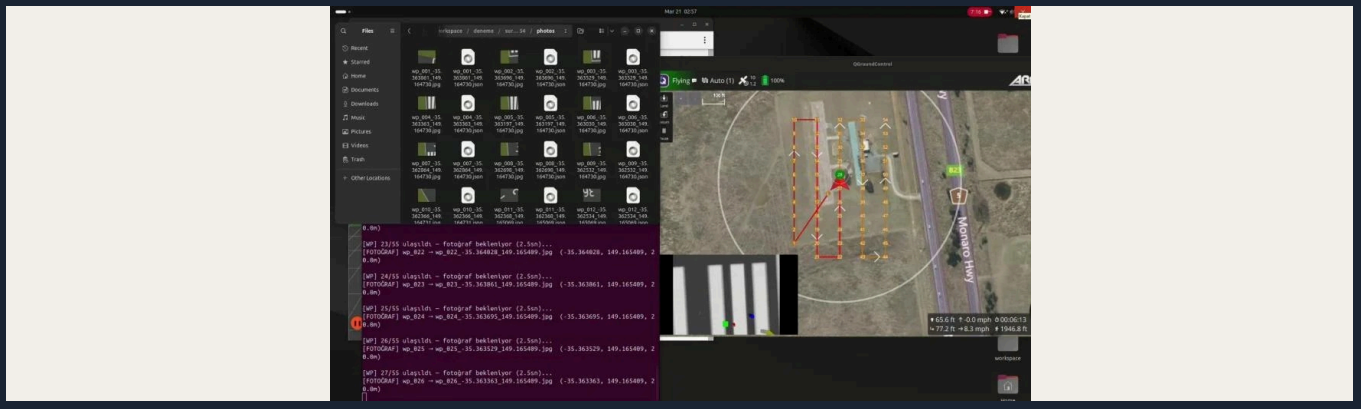
Release mechanism, assembled

PROJECT 01 · FIELD

Flight testing and project documentation

KA-FA1500 during field testing

Pre-flight preparation and equipment setup



Autonomous mission planning and post-flight image geolocation: ground station screen

ADDITIONAL DELIVERABLES

Alongside the software work I produced the team's **Technical Design Report** and the **team website** that satisfies the competition rules' web requirements. The site was designed in Framer and then migrated to its own codebase; it holds 51 pages, 28 technical blog posts and five technical documents.

TDR | 28 technical posts | 51 pages | atilimuav.com

Project one-pager

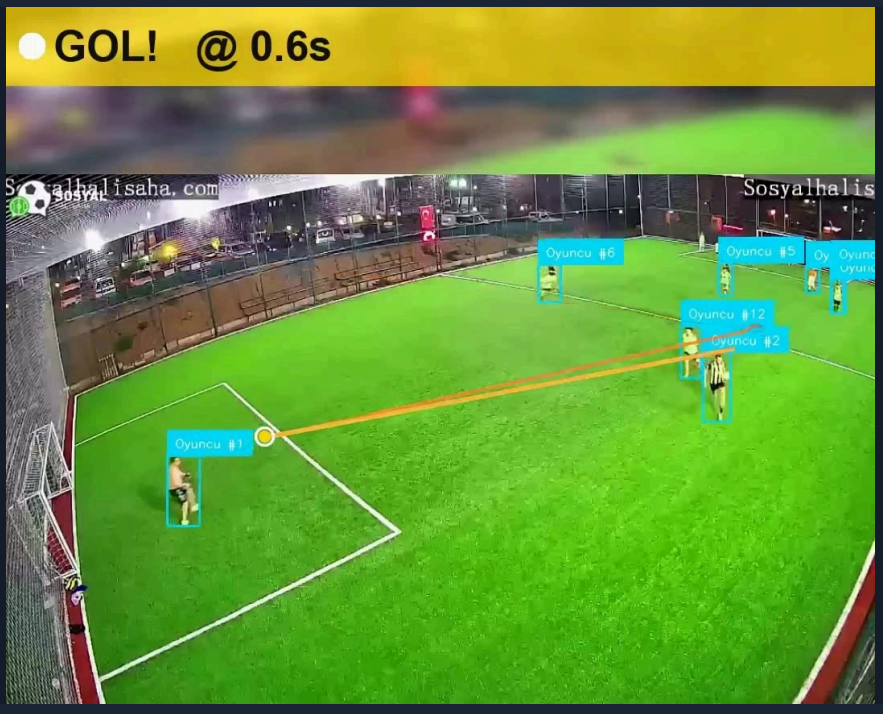
PROJECT 02

Sports Field Video Analytics

ROLE
Freelance developer

CLIENT
France-based sports technology startup

PERIOD
2026 – Present



A goal detected automatically and stamped with its timestamp: a frame from the system output

PROBLEM

Five-a-side pitches record matches with a single fixed wide-angle camera. What players actually want is not hours of raw footage but the 15-second clip of their own goal. Editing that by hand does not scale.

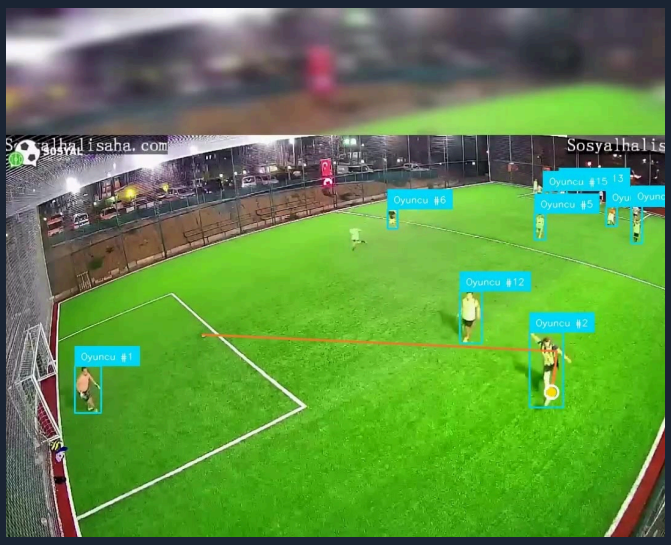
SOLUTION

An end-to-end pipeline that detects players and the ball from a single camera, tracks them with persistent IDs, infers events from the ball trajectory and cuts clips automatically around those events. Output: an annotated match video, per-event clips and a combined highlight reel.

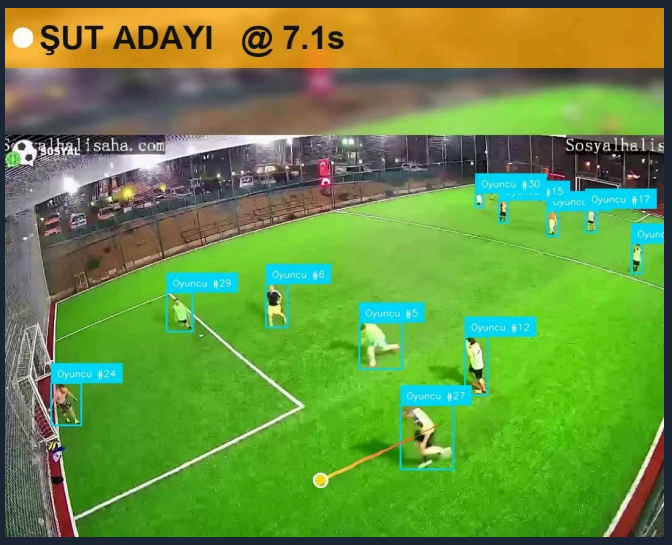
1	3	H.264
CAMERA · FIXED WIDE ANGLE	OUTPUT TYPES · VIDEO, CLIP, REEL	BROWSER-COMPATIBLE ENCODING

YOLO · ByteTrack · OpenCV · PyTorch · supervision · scikit-learn · ffmpeg · Streamlit

Detection, tracking and event inference

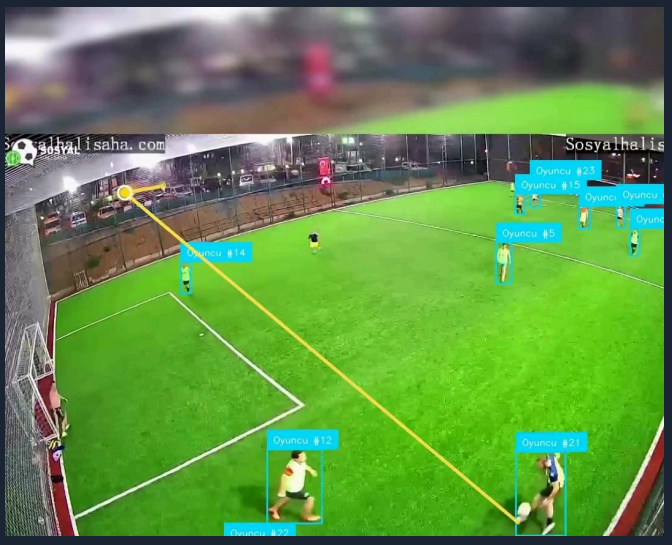


Player detection and persistent ID assignment (ByteTrack)



Shot candidate event: detected from a velocity spike in the ball trajectory

Team classification from jersey colour using KMeans



Ball trajectory tracking: linking positions across frames

PIPELINE

- **Detection:** players and ball with YOLO (Ultralytics)
- **Tracking:** persistent player IDs with ByteTrack
- **Event:** shot/goal candidates from sudden velocity change in the ball trajectory
- **Editing:** clip cutting around events plus a combined reel
- **Output:** browser-compatible H.264 via ffmpeg
- **Interface:** a demoable front end in Streamlit

ENGINEERING NOTE

The sample videos were shot handheld/on a gimbal, so the goal area could not be calibrated; events are therefore flagged with a **shot candidate** heuristic rather than as goals. The event detection function was written to accept a goal-area parameter: on the fixed camera rig in production the same code becomes full goal detection. Declaring the limitation up front is cheaper than rearchitecting later.

YOLO · ByteTrack · OpenCV · PyTorch · supervision · scikit-learn · ffmpeg · Streamlit

PROJECT 03

Multi-Vendor E-Commerce Marketplace

ROLE	PERIOD	SCOPE	SCALE
Freelance · Full stack	2026	Backend + Frontend + Deployment	10 domain modules · 36 commits

A marketplace with money moving between buyer, seller and platform. In systems like this the most expensive mistake is a money mistake, so most architectural decisions were made around **correctness and auditability**. Interface screenshots are withheld out of client confidentiality; the account here stays at architecture level. **The system can be demonstrated in person on request.**

FOUR CRITICAL DECISIONS

MULTI-VENDOR PIVOT

The buyer's single payment is split automatically into per-seller sub-orders. Each seller sees only their own order; shipping, returns and earnings all run through that sub-order. Start with a single-order model and this separation cannot be retrofitted.

DOUBLE-ENTRY LEDGER

Seller earnings are not held in a single balance field. Every movement is its own row: a successful payment writes a CREDIT, cancelled logistics writes an automatic DEBIT. The balance is summed each time and never overwritten. Accounting's centuries-old method gives you auditability for free.

PAYMENT PROVIDER ABSTRACTION

Payment logic is fully separated from the view layer behind an Adapter pattern. Swapping provider leaves business logic untouched. Synchronous verifications are guarded by a double-spend lock and asynchronous webhooks by an idempotency lock, so payment state stays consistent even if the browser is closed mid-checkout.

SINGLE-SCHEMA API

A custom DRF renderer and exception handler make **every** response, 200 or 500 alike, come back in the same JSON schema: `success` / `data` / `error`. There is no case left where the frontend has to guess the response shape.

INFRASTRUCTURE

Media uploads are streamed to Cloudflare R2 (S3-compatible) through boto3 without ever touching server disk, which keeps the application stateless and horizontally scalable. Delivery runs on six services under Docker Compose, with CI on GitHub Actions.

STACK

`Django 5` `DRF` `PostgreSQL` `Redis` `Celery` `Django Channels` `Next.js` `TypeScript` `Tailwind` `NextAuth` `Docker` `Nginx` `Cloudflare R2` `GitHub Actions`

10	6	2	1
DOMAIN MODULES	DOCKER SERVICES	REPOS · API + FRONTEND	API RESPONSE SCHEMA

PROJECT 04

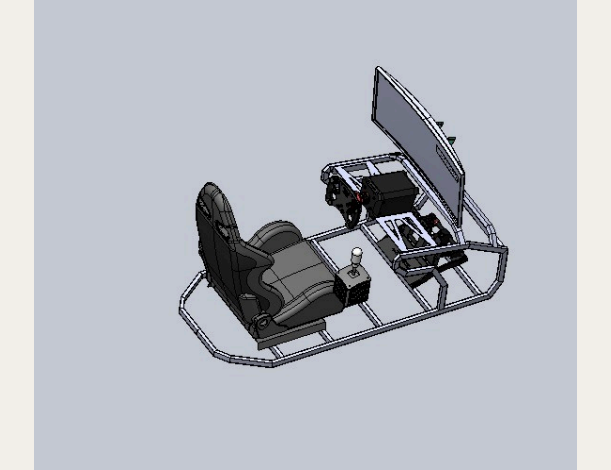
Racing Simulator Motion Platform

ROLE
Product owner and developer

STATUS
Working prototype

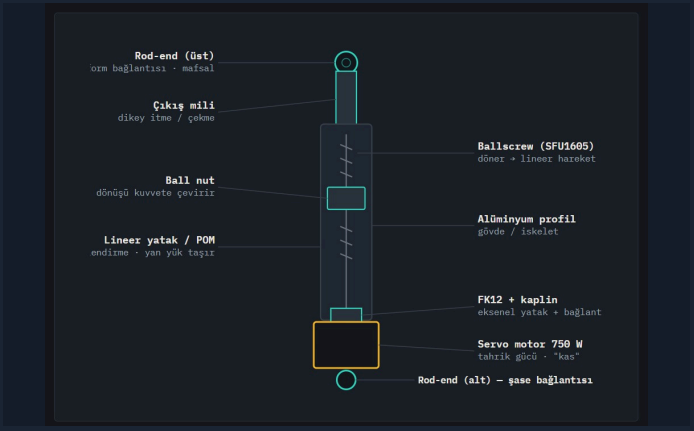
ARCHITECTURE
3DOF / 4DOF

DRIVE
750 W AC servo · 50:1



Platform CAD design: seat, wheel, display and chassis layout

Welded steel chassis during fabrication



Actuator cross-section: converting rotation into linear thrust

ACTUATOR DESIGN

Each actuator converts the servo motor's rotation into linear push and pull through an **SFU1605 ballscrew**. Side loads are carried by a linear bearing and POM bushings; the motor connects via an FK12 axial bearing and a coupling. Rod-end joints at the top and bottom leave the changing angle between platform and chassis free: without them the ballscrew would take a bending load and burn through its service life.

SFU1605 | 750 W AC servo | 50:1 planetary | AASD-15A | Thanos AMC

ENGINEERING DECISION

I compared 3DOF and 4DOF architectures and derived centre-of-gravity-aware torque calculations; because driver weight and seat position set the torque requirement directly, that calculation drove the entire configuration choice.

THE LATENCY PROBLEM

On motion platforms perceived quality depends directly on **latency**. To remove the telemetry chain that runs through the PC I developed a **standalone ESP32 + UDP telemetry** architecture, cutting a layer out of the command path and reducing end-to-end latency.

PROJECT 05

Defence Industry Experience

CONFIDENTIALITY NOTE

The work in this section is subject to confidentiality obligations. Client and product names, performance parameters, screenshots and test data are deliberately omitted; the account stays at the level of the engineering problem solved and the technology used.

Technical detail can be described and shown in person, to the extent confidentiality obligations allow. Written or digital sharing requires permission from the relevant organisations.

REAL-TIME TARGET DETECTION AND TRACKING SYSTEM

Digitest Defence and Aerospace · 2021–2024 · 6–10 engineer software team

I developed real-time image processing based target detection and tracking software for a defence industry client. The detection side was written in C++, the tracking side in Python. I built the data-driven symbol overlay rendering on live video and the operator interface in Qt Creator/C++.

To meet the real-time throughput requirement I built parallel processing pipelines with C++ multithreading, CUDA C and OpenMP. The whole pipeline was integrated to run on NVIDIA Jetson embedded platforms, with data exchange carried over TCP, UDP and UART.

C++ · Python · CUDA C · OpenMP · Qt Creator · NVIDIA Jetson · TCP/UDP/UART · Multithreading

MIXED REALITY SIMULATION AND VISUALISATION

Wupsoft Defence & Aerospace · 2024 – Present

I develop VR/AR mixed reality applications for the defence industry using Unity C# and Oculus. I built a mixed reality showcase application that was exhibited at a defence industry trade fair.

On a TÜBİTAK-funded external ballistics simulation project I develop simulation modelling in MATLAB and large-data visualisation in XR. I have also delivered XR/VR development training to 6–20 engineers.

Unity · C# · MATLAB · Oculus · VR/AR · Simulation modelling · Data visualisation

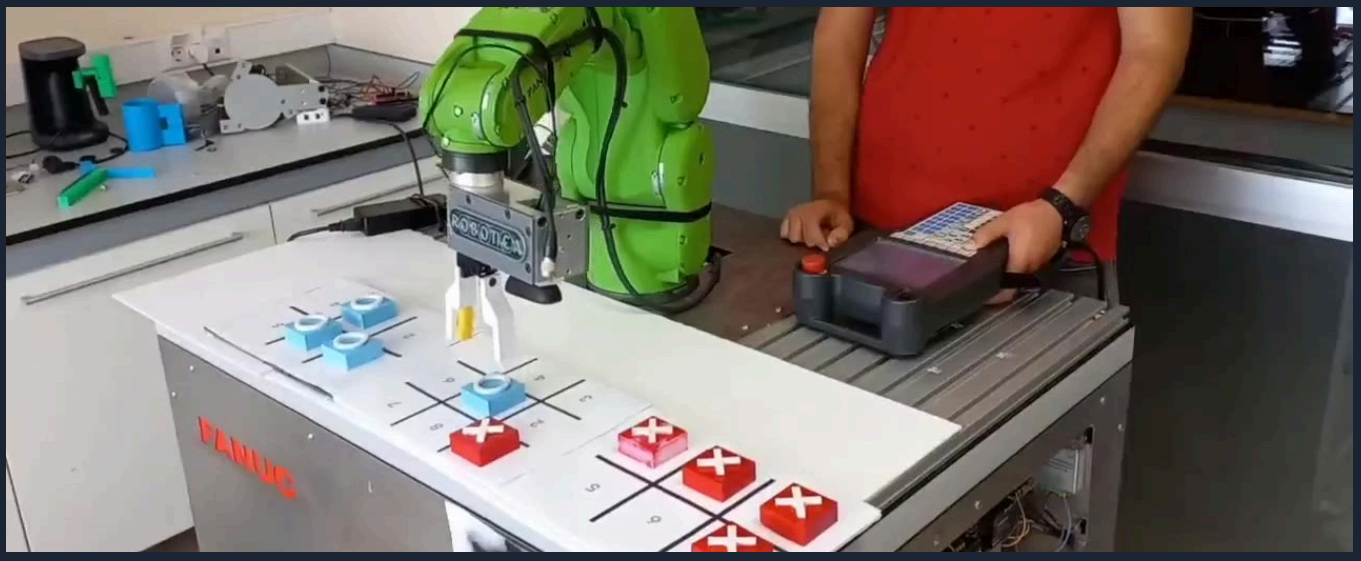
MSC THESIS

User stress measurement and adaptive application via a biocybernetic loop in virtual reality: Atılım University, MSc Software Engineering (thesis track), currently in the thesis phase. A closed-loop system in which the virtual reality environment adapts itself to biosignals read from the user. A subject with direct application in simulation and training systems.

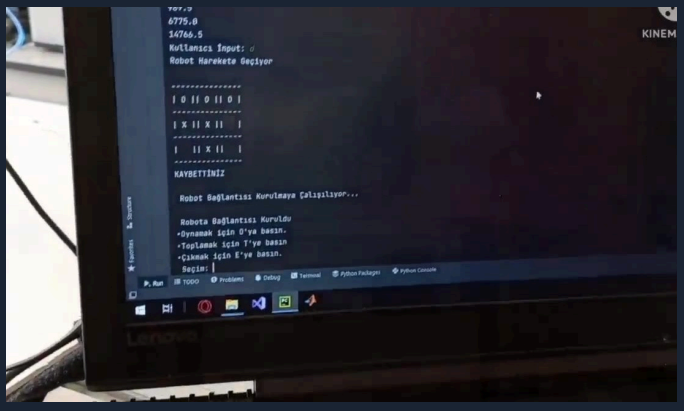
PROJECT 06

Industrial Robot Playing Tic-Tac-Toe via Computer Vision

ROLE	CONTEXT	PERIOD	TEAM
Software unit	Undergraduate capstone project	2021-2022	4 people



The robot makes its move based on the board state it reads through computer vision



Vision output: the board matrix read from the camera and the robot's move decision

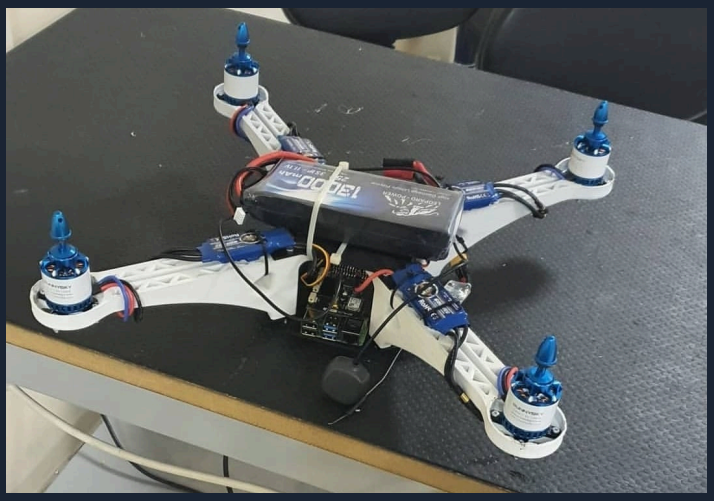
MY CONTRIBUTION

- The image processing pipeline that **turns the camera frame into a board matrix**, in Python
- Designing and implementing the **Minimax tree** based game solving algorithm
- The operator interface: prototyped in Kodular Creator, then rewritten in Qt Creator
- Translating raw robot data into language the user actually understands

Python · Computer vision · Minimax · Qt Creator · Android · Industrial robot

> This project was the first version of what I do with YOLO today: turning a camera frame into a data structure a machine can decide on. It used classical image processing rather than deep learning, but the problem was the same one: **turning an image into an action**.

Autonomous Drone and Electric Vehicle



Avionics layout: Raspberry Pi + Navio 2, GPS antenna, ESCs and a 13,000 mAh LiPo

AUTONOMOUS MULTIROTOR DRONE

TÜBİTAK International Drone Competition · Atılım University Robotics Society · 2020–2021

I designed and assembled the avionics system, integrated the **Navio 2** flight controller as a HAT on Raspberry Pi, established telemetry and GPS communication and enabled autonomous flight.

I performed motor selection and the thrust and endurance calculations that drove ESC and battery selection, designed the payload mechanism and developed mission-specific image processing in Python.

Navio 2 | Raspberry Pi | Python | Telemetry | GPS | Thrust calculation

ELECTRIC VEHICLE TELEMETRY AND VCU/ECU

TÜBİTAK Efficiency Challenge · Kocaeli 2021 · **Top 10**

I developed the Vehicle Control Unit (VCU/ECU) and telemetry software in Python. Battery management system data was relayed wirelessly over **LoRa** and rendered on a **Nextion** touchscreen driven by Raspberry Pi.

Alongside the software I worked on the vehicle's electrical assembly: motor drivers, electronics and the onboard charging unit. This project is where my habit of reading the hardware underneath the software started.

Python | LoRa | Raspberry Pi | Nextion | VCU / ECU | Power electronics

Efficiency Challenge 2021, Kocaeli: the team's race vehicle

CONTACT

Thank you

EXPERTISE SUMMARY

Real-time image processing and computer vision systems; getting those systems to run on embedded platforms; autonomous aerial vehicle software; dataset engineering and model training. My mechatronics background means I can also read the hardware underneath the software, from sensor selection to power budget, from torque calculation to communication protocol.

CURRENTLY

Senior software engineer at Wupsoft Defence & Aerospace working on XR/VR and simulation projects, Software Team Lead at Atılım UAV, and writing my MSc software engineering thesis at Atılım University.

GET IN TOUCH

serdar.anil33@gmail.com

linkedin.com/in/sanildemirbas

Ankara, Türkiye

IN THIS DOCUMENT

9 projects · 6 sections

Defence work has been sanitised

Version: August 2026

Work under confidentiality can be shown in person, to the extent obligations allow.