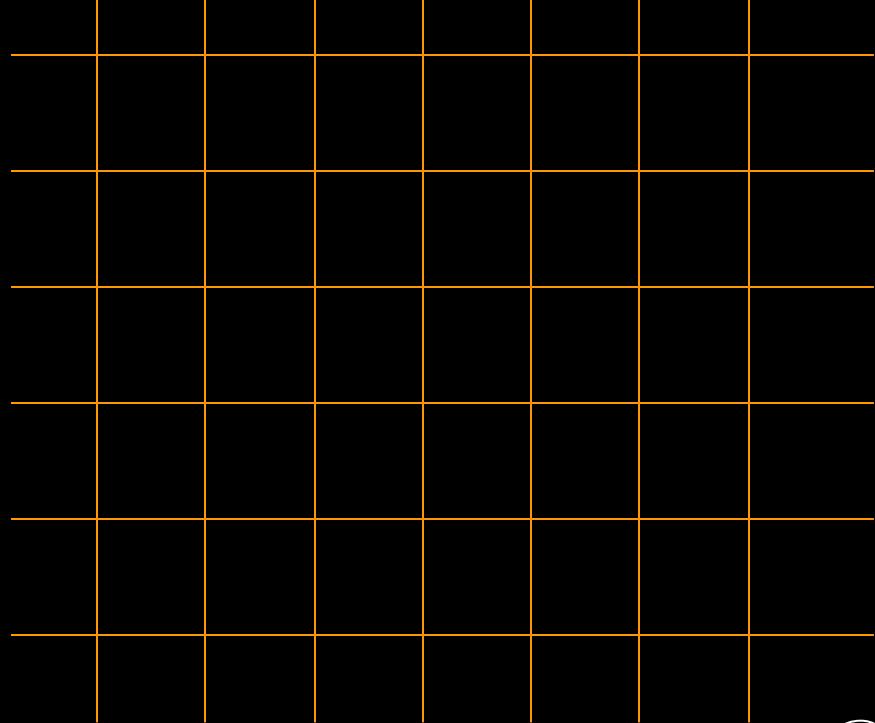





# Machine Perception Tracking

## Projektprüfung

Jule Naßenstein, Fouad Ghazal, Aleem  
Hussain, Zayd Badla



2025

# TABLE OF CONTENTS

**01**

**Aufgabenstellung**

**02**

**Module**

**03**

**Arbeitsaufteilung**

**04**

**Demo**



# 01

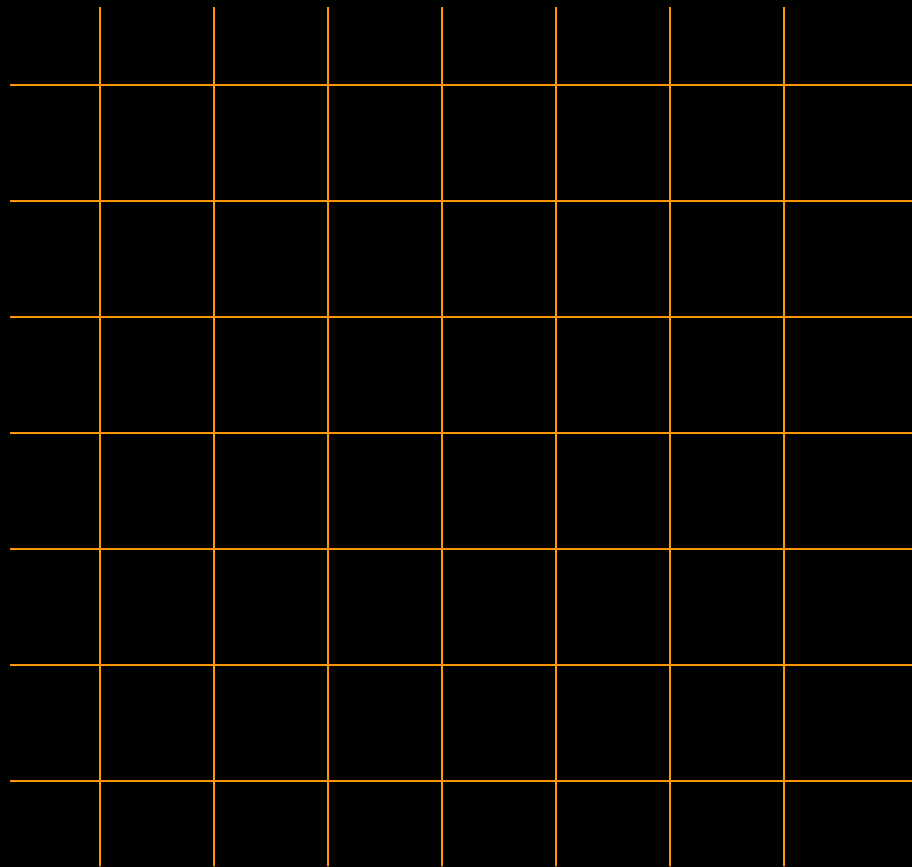
## **Aufgabenstellung**

# Aufgabenstellung

Entwicklung eines Algorithmus um  
Fußballspiele zu analysieren

In verschiedene Module unterteilt

- Detektor
- Optical Flow
- Tracker
- Shirt Classifier



# 02

## Module

Module



# Module

**01** Detektor

**02** Optischer Fluss

**03** Tracker

**04** Shirt Classifier

# Detektor

## Aufgabe

Erkennung des Balles, der Torhüter, der Spieler und der Schiedsrichter

## Implementation

Definierung einer Bounding Box für jedes Objekt



# YOLO Netzwerke

- Echtzeit-Objekterkennungsnetzwerk
  - lokalisiert und klassifiziert alle Objekte im Bild in einem einzigen Vorwärtsthroughlauf
1. Eingabebild wird in ein festes Raster aus Zellen unterteilt
  2. Jede Rasterzelle ist verantwortlich Objekte zu erkennen deren Mittelpunkt in die Zelle fällt
  3. für jedes potenzielle Objekt gibt YOLO
    - ( $x, y$ ) – Mittelpunkt der Box (relativ zur Zelle)
    - ( $w, h$ ) – Breite und Höhe der Box (relativ zum Bild)
    - Confidence-Score – Wie sicher ist YOLO, dass sich dort ein Objekt befindet
    - Klassenscores – Wahrscheinlichkeiten für jede Klasse (z. B. Ball, Spieler)

## Vorteile von YOLO

- Schnell: Echtzeitfähig – ideal für Videos und Roboteranwendungen
- Kontextwahrnehmung: Das gesamte Bild wird betrachtet, nicht nur lokale Regionen



# Aufbau des Moduls

## **start Methode**

Initialisiert das YOLO-Modell

## **step Methode**

Verarbeitet ein Bild pro Zeitschritt, erkennt darin relevante Objekte und gibt sie strukturiert zurück.

## **stop Methode**

Setzt das YOLO Modell auf None

# Aufbau der step Methode

```
47 def step(self, data):
48     """Process a single step of the pipeline.
49     Args:
50         data (dict): Input data containing the image to be processed.
51     Returns:
52         dict: A dictionary containing the detections and their corresponding classes.
53     """
54     image = data["image"]
55     if self.model is None:
56         self.initialise_yolo()
57
58     results = self.model(image)[0]
59     if results.bboxes is None or len(results.bboxes) == 0:
60         return {
61             "detections": np.empty((0, 4), dtype=np.float32),
62             "classes": np.empty((0, 1), dtype=np.int64),
63         }
64
65     boxes = results.bboxes.xywh.cpu().numpy() # Center-based boxes: (X, Y, W, H)
66     classes = results.bboxes.cls.cpu().numpy() # Klassen
67
68     valid_classes = [
69         0,
70         1,
71         2,
72         3,
73     ] # Valid classes: 0=Ball, 1=Player, 2=Referee, 3=Goalkeeper
74     # Filter indices for valid classes
75     indices = [i for i, cls in enumerate(classes) if int(cls) in valid_classes]
76
77     # If no valid detections, return empty arrays
78     if not indices:
79         return {
80             "detections": np.empty((0, 4), dtype=np.float32),
81             "classes": np.empty((0, 1), dtype=np.int64),
82         }
83
84     # Extract the filtered detections and their corresponding classes
85     detections = np.array([boxes[i] for i in indices], dtype=np.float32)
86     class_tensor = np.array([[int(classes[i])] for i in indices], dtype=np.int64)
87
88     return {"detections": detections, "classes": class_tensor}
```

lädt das aktuelle Bild

initialisiert das Modell falls noch nicht

übergibt das Bild an das YOLO Modell

wenn nichts erkannt wird ein leeres array zurückgegeben

konvertiert die Bounding Boxes und Klassen in NumPy-Arrays

Filtert nur Objekte die zu den interessanten Klassen gehören

wenn keine gültigen Objekte wird wieder ein leeres Array zurückgegeben

baut arrays für die gültigen Erkennungen

gibt das Ergebnis zurück

# Was ist ein Optischer Fluss

- Misst, wie sich Bildpunkte von Frame zu Frame verschieben
- Gibt für jeden Pixel einen Bewegungsvektor ( $v_x$ ,  $v_y$ ) an, der zeigt, wie sich der Pixel bewegt hat.
- liefert die Infos, die andere Systeme nutzen, um Bewegung zu verstehen oder zu kontrollieren.



# Optionale GPU-Beschleunigung für Farnebäck

## CPU vs. GPU – Was ändert sich?

### CPU-Modus (Standard)

Läuft auf jeder Maschine  
Gut für Tests und kleine Auflösungen

### GPU-Modus (optional)

Nutzt NVIDIA-CUDA  
Deutlich schneller bei hohen Auflösungen und vielen Bildern pro Sekunde



# Farneback vs. Lucas-Kanade

Merkmal	Farneback	Lucas-Kanade
Dichte	Vektor für jeden Pixel	Nur für ausgewählte Punkte
Feature-Detektion	Nicht nötig	Erst Punkte finden & filtern
Robustheit	Gut bei Rauschen & Helligkeitsänderungen	Kann bei starkem Rauschen versagen
GPU-Support	Direkte CUDA-Implementierung verfügbar	Keine offizielle GPU-Variante
Rechenaufwand	Höher (dicht), aber mit GPU sehr schnell	Niedriger pro Punkt, aber Verwaltungsaufwand

# Warum Farneback für uns ideal ist

Wir berechnen die Durchschnittsbewegung aller Pixel für die Gesamtbewegung des Bildes.

## **Automatisch & vollständig:**

Für **alle** Pixel wird direkt ein Vektor berechnet.

## **Robust & zuverlässig:**

Weniger empfindlich gegen Rauschen und Lichtänderungen.

## **Echtzeit-Einsatz:**

Mit `cuda_FarnebackOpticalFlow` hohe Bildraten möglich.

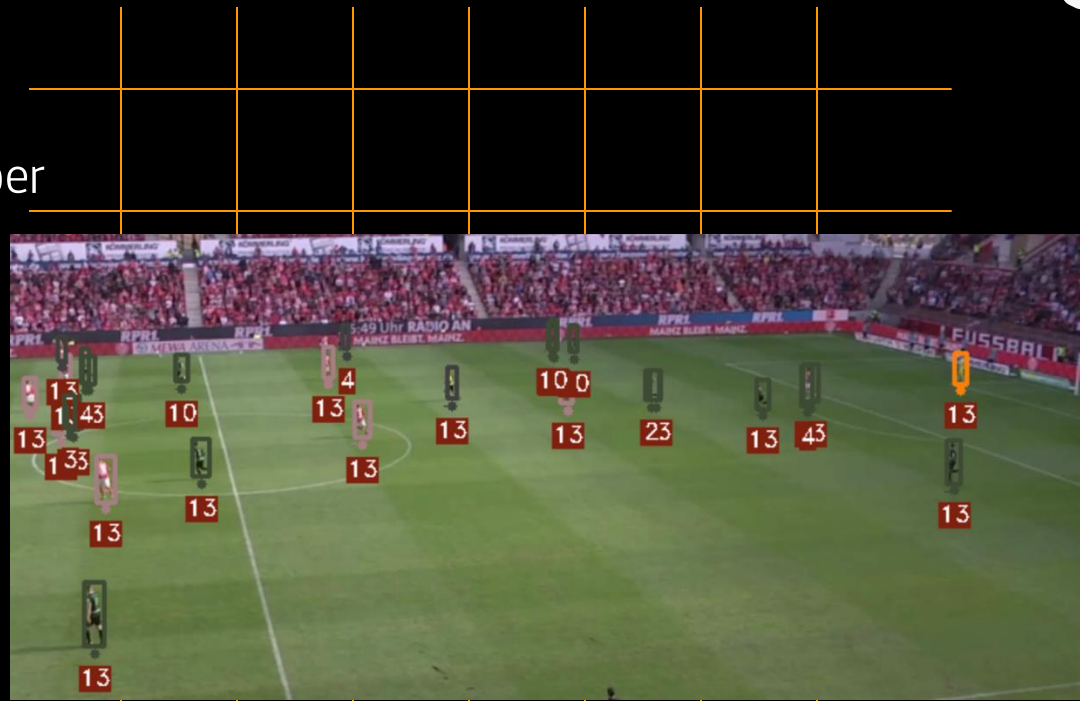
## **Schnelles Prototyping:**

Standard-Parameter liefern sofort gute Ergebnisse.

# Was ist ein Objekt-Tracker?

## Aufgabe

- Erkenntnisse (Ball, Spieler, etc.) über mehrere Bilder verfolgen
- Jedes Objekt bekommt eine eindeutige ID
- Entscheidung: Neuer Track oder vorhandener Track?



# Filter (Einzelobjekt-Verfolgung)

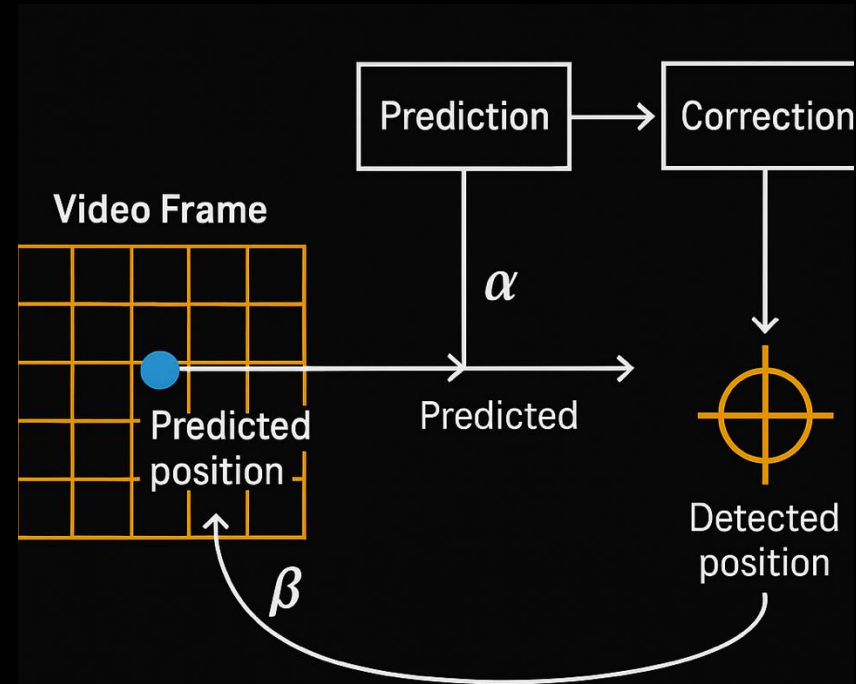
## Alpha-Beta-Filter

Arbeitet direkt mit Bounding Boxes im Bild ( $[cx, cy, w, h]$ )

Benötigt **nur zwei Parameter**:

$\alpha \rightarrow$  Gewichtung der  
Positionskorrektur

$\beta \rightarrow$  Anpassung der Geschwindigkeit



## Alpha-Beta Filter



# Shirt Classifier

**Ziel: Klassifikation aller Spieler anhand ihrer Trikotfarbe in Team A und Team B.**

- Analysiert Trikotfarben getrackter Spieler
- Erkennt automatisch zwei Teamfarben
- Weist Spielern Team A (1), Team B (-1) oder unklar (0) zu
- Liefert Basis für visuelle Trennung & taktische Analysen



Team A



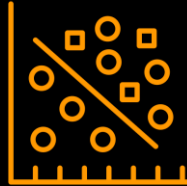
Team B

# Benutzte Techniken



## **Bildverarbeitung**

ROI-Extraktion & Farbmittelwert



## **Clustering**

KMeans zur Teamfarben-Trennung



## **Feature-Vergleich**

L2-Distanz zur Farbzuzuordnung



## **Thresholding**

Schwellenwert für sichere Klassifikation



## **Data Association**

Farbinformation mit Tracking-Daten verknüpft

# Code-Aufbau und Logik

## Daten empfangen

data["image"], data["track  
s"], data["trackClasses"]

## Spieler erkennen & ROI extrahieren

Trikotbereich  
ausschneiden, Farbmittel  
berechnen

## Genug Farben?

→ Ja → weiter  
→ Nein → neutrale  
Rückgabe

## KMeans-Clustering (2 Farben)

Teamfarben bestimmen  
(Gruppe links = Team A)

## Ergebnis zurückgeben

teamAColor, teamBColor,  
teamClasses

## (Optional) Farbverstärkung

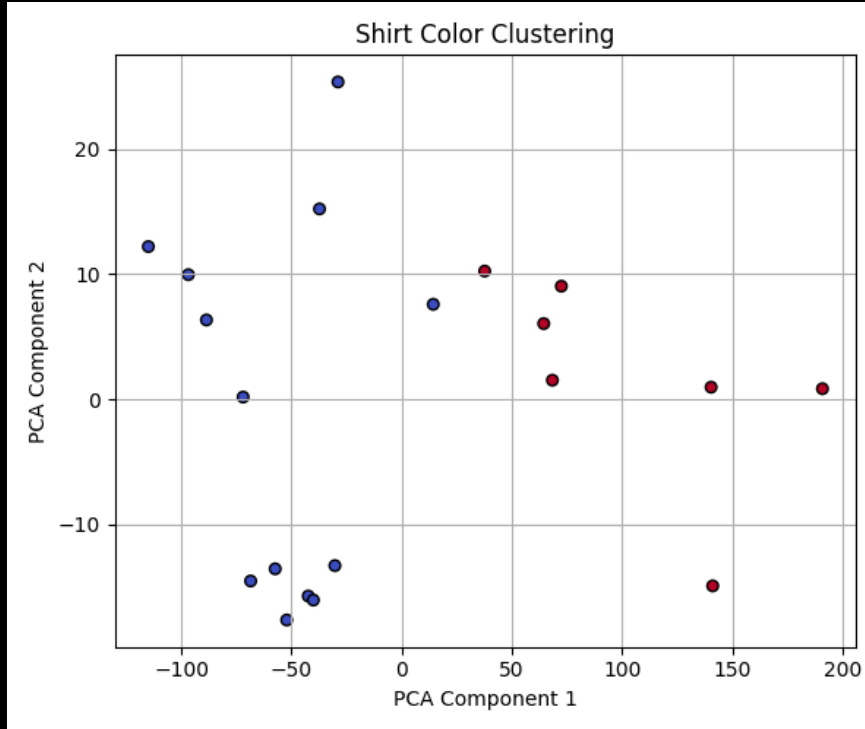
## Sicherheitsfix

Mindestens ein Spieler  
pro Team sicherstellen

## Farbvergleich je Spieler

→ L2-Distanz zu Team  
A/B  
→ Zuweisung oder 0  
(unklar)

## KMeans-Cluster der Shirtfarben



## Oberkörper-ROI





**03**

# **Aufgabenteilung**

# Aufgabenteilung



**Jule**

detector.py



**Fouad**

tracker.py



**Zayd**

opticalflow.py



**Aleem**

shirtClassifier.py



**04**

**Demo**