ShuttleView: A Badminton Shuttlecock Landing Detection System

Overview

This project aims to develop an embedded system for real-time detection of shuttlecock landing positions ("inside," "outside," or "on the line") in a badminton game. Four embedded devices, each equipped with a high-speed camera, will be placed at the corners of the court to capture footage and predict landing outcomes using a lightweight machine learning model.

Project Requirements

- **Setup:** Four embedded devices with high-speed cameras positioned at the court's corners (e.g., two at the back, two at the front or sides).
- **Task:** Real-time detection of shuttlecock landing position ("inside," "outside," potentially "on the line").
- **Key Needs:**
  - **Resolution:** High enough to resolve the shuttlecock (~6-7 cm) and line (1-4 cm wide) from a corner perspective.
  - **Accuracy:** Reliable classification/detection under fast motion (shuttlecock speeds up to 400+ km/h) and varying conditions (lighting, angles).
  - **Real-Time:** Processing within milliseconds per frame on embedded hardware (e.g., Raspberry Pi, Jetson Nano).
- **Dataset Requirements:**
  - Match the camera resolution and angle of the embedded system.
  - Provide sufficient detail for accurate detection.
  - Support a lightweight, fast model (e.g., YOLOv8 or MobileNetV2) suitable for embedded deployment.

Dataset Generation Options

To avoid physical data capture, synthetic dataset generation options were evaluated:

1. Generative AI (e.g., Stable Diffusion)

- **Resolution:** Flexible (e.g., 512x512, 1024x1024), but fine details (shuttlecock feathers, line edges) may blur without fine-tuning.
- **Accuracy:** Struggles with precise spatial relationships and lacks motion context (e.g., high-speed blur).
- **Suitability for Embedded:** Poor generalization to real high-speed footage due to synthetic artifacts.
- **Time to Generate:** 6-8 hours for 1,000 images.
- **Verdict:** Poor fit due to limited accuracy and realism for high-speed, corner-based detection.

2. 3D Simulation (e.g., Blender) **[Recommended]**

- **Resolution:** Fully controllable (e.g., 720p, 1080p, 4K), matching camera specs with fine detail (shuttlecock shape, line width).
- **Accuracy:** High precision in shuttlecock placement (millimeter-level) and ability to simulate motion blur, lighting, and textures.
- **Suitability for Embedded:** Ideal for training a lightweight object detection model (e.g., YOLOv8) optimized for embedded hardware (30-60 fps on Jetson Nano).
- **Time to Generate:** 6-9 hours for 1,000 images, scalable with scripting.
- **Pros:**
    - Matches high-speed camera needs (e.g., 120-240 fps with blur).
    - Replicates four corner perspectives for accurate training.
- **Cons:** Requires effort to match real-world textures/lighting.
- **Verdict:** Excellent fit for resolution, accuracy, and real-time embedded needs.

3. AI-Augmented Small Real Dataset

- **Resolution:** Limited by seed data (e.g., 1080p), matching high-speed camera output if collected.
- **Accuracy:** High realism from real data, but positional accuracy depends on seed quality/quantity.
- **Suitability for Embedded:** Good with 200-400 real images augmented to 1,000+, but collection is time-intensive.
- **Time to Generate:** 4-8 hours for seed data + 1-2 hours augmentation.
- **Pros:** Directly applicable to real cameras.
- **Cons:** Conflicts with goal of avoiding physical capture.
- **Verdict:** Strong but impractical without initial effort.

Recommended Approach: 3D Simulation with Blender

Why Blender?

- **Resolution:** Render at exact camera specs (e.g., 1280x720 at 120 fps, 1920x1080 at 60 fps), matching devices like Raspberry Pi High Quality Camera or industrial high-speed USB cameras (e.g., 240 fps at 640x480).
- **Accuracy:** Precise shuttlecock positioning and motion simulation (e.g., 50-100 km/h landing blur) ensure reliable line call detection.
- **Embedded Compatibility:** Supports training YOLOv8n (nano version), achieving:
    - Jetson Nano: ~10-20 fps at 640x480.
    - Raspberry Pi 4 with Coral USB Accelerator: ~15-30 fps.
- **Dataset Size:** 1,000-2,000 images (250-500 per corner), generated in 6-9 hours.