# **Real-Time Panoramic Sports Tracking on ESP32-S3: An Architectural and Implementation Guide**

## **I. Strategic Analysis: Architecting a Real-Time Tracking System for Embedded Vision**

### **1.1 Defining the System: From Pixel to Panorama**

The objective is to engineer an autonomous computer vision system capable of recording team sports in real time. This system will be built upon a resource-constrained microcontroller, the Espressif ESP32-S3, tasked with driving a panoramic camera mount. The core functionality involves leveraging object tracking techniques to follow the main action of a game, ensuring it remains centered within the camera's field of view. This creates a closed-loop cyber-physical system where the input is a continuous video stream from a camera module, the central processing unit is the ESP32-S3 executing a neural network, and the output is a precise series of Pulse Width Modulation (PWM) signals directed to servo motors. The feedback loop is inherently closed by the camera itself, which continually observes the results of its own motion, allowing for dynamic adjustments. The primary goal is to track the centroid of collective player activity, providing a stable and automated method for capturing game footage without a human operator.1

### **1.2 Core Challenges: Latency, Memory, and Power on a Microcontroller**

Deploying such a system on an embedded platform introduces a set of fundamental constraints that dictate every architectural and implementation decision. These challenges must be addressed to create a viable solution.

* **Latency:** The system's ability to process video frames must outpace the dynamic movement of the game. A hard real-time requirement emerges from the need to analyze a frame, calculate a corrective camera movement, and execute that movement before the on-field action has shifted significantly out of the frame. The key performance metric for this constraint is Frames Per Second (FPS), which represents the number of complete perception-action cycles the system can execute per second. Insufficient FPS will result in a system that cannot keep up with the game, rendering it ineffective.3
* **Memory:** The ESP32-S3 microcontroller possesses a limited amount of high-speed on-chip Static RAM (SRAM), typically 512 KB.6 Modern computer vision models, characterized by their large weight matrices and the need for substantial memory buffers (known as the tensor arena) for intermediate calculations, can easily exceed this capacity. This limitation forces the use of external Pseudo-Static RAM (PSRAM), which offers a much larger memory pool (often 8 MB or more) but at the cost of significantly slower access speeds due to its connection via an SPI bus. This trade-off between memory capacity and access latency is a central challenge in embedded AI.8
* **Power:** While the primary focus of the query is functional, any practical embedded system must consider power consumption. The computational load of continuous neural network inference, combined with the mechanical work performed by servo motors, creates a non-trivial power draw. For any potential deployment scenario that is not permanently tethered to a mains power source, optimizing for power efficiency becomes a critical design consideration.12

### **1.3 The Three Pillars of the Solution: Perception, Optimization, and Control**

To manage the complexity of this project, the problem can be deconstructed into three distinct but deeply interconnected sub-problems. These pillars will form the foundational structure of this report.

* **Perception:** This pillar concerns the "eyes" of the system. It involves the selection, training, and deployment of a neural network model capable of processing an input image and extracting meaningful spatial information—specifically, the location of players or the center of the game's action.4
* **Optimization:** This pillar addresses the core challenge of running a computationally expensive perception model on resource-constrained hardware. It encompasses a range of techniques, from model quantization and the use of hardware-specific acceleration libraries to sophisticated memory management strategies, all aimed at meeting the system's stringent real-time performance requirements.14
* **Control:** This pillar represents the "muscles" of the system. It involves translating the abstract coordinate data generated by the perception block into smooth, stable, and accurate physical movement of the camera platform. This requires the implementation of a robust control system, such as a Proportional-Integral-Derivative (PID) controller, to manage the servo motors.20

The success of the project is not merely a function of excellence in one of these areas but rather the result of a holistic integration where each pillar is designed with the constraints and capabilities of the others in mind. The project is fundamentally a real-time systems integration challenge, where achieving a "good enough" perception result within a strict time budget is more valuable than a state-of-the-art model that fails to meet latency requirements. The stability of the control system is as critical to the final output as the accuracy of the neural network. Therefore, the engineering effort must be balanced across all three pillars, with a pragmatic focus on performance and efficiency over theoretical perfection.

### **1.4 High-Level Architectural Blueprint**

A conceptual blueprint for the system illustrates the flow of data and control. This model serves as a reference for the detailed discussions in subsequent sections.

1. **Input Stage:** A camera module captures a video frame.
2. **Data Transfer:** A Direct Memory Access (DMA) controller efficiently transfers the image data into a frame buffer, which will likely reside in the slower, but larger, external PSRAM.
3. **Perception Stage (AI Core):** The ESP32-S3's processing core loads the neural network model and the input frame data. It performs inference to detect objects (players).
4. **Post-Processing:** The raw output tensor from the neural network is decoded. This involves identifying candidate detections, applying Non-Maximum Suppression (NMS) to remove redundant detections, and calculating a single "center of action" coordinate (e.g., the mean coordinate of all detected players).
5. **Control Stage:** A PID controller receives the calculated center of action. It compares this to the desired setpoint (the center of the screen) to compute an error value. Based on this error, the PID controller calculates the necessary velocity adjustments for the pan and tilt servos.
6. **Actuation Stage:** The calculated adjustments are translated into updated PWM duty cycles, which are sent to the servo motors via the ESP32-S3's hardware PWM driver.
7. **Feedback Loop:** The servos move the camera, changing its field of view. The next captured frame reflects this new view, closing the control loop.

This entire sequence must be completed within a tight time budget (e.g., under 100ms) to achieve a frame rate of 10 FPS or higher, which is necessary for smooth tracking of a live sports game.

## **II. The ESP32-S3 as an Edge AI Platform: A Hardware-Software Deep Dive**

The selection of the ESP32-S3 is a strategic choice for this application, as its architecture offers specific advantages for embedded machine learning that are absent in many other microcontrollers, including its predecessors. A thorough understanding of its hardware and the associated software toolchains is paramount to unlocking its full potential.

### **2.1 The Xtensa LX7 Dual-Core Architecture and its Implications**

The ESP32-S3 is powered by a dual-core 32-bit Xtensa LX7 microprocessor, capable of running at frequencies up to 240 MHz.7 This architecture provides several key features relevant to computer vision workloads:

* **Vector Instructions (SIMD):** A significant enhancement over older ESP32 chips is the inclusion of vector instructions, also known as Single Instruction, Multiple Data (SIMD) capabilities.12 These instructions allow the processor to perform the same mathematical operation on multiple data points simultaneously. This is exceptionally well-suited for the matrix and vector arithmetic that forms the backbone of neural network operations like convolutions and dot products, leading to substantial performance gains.17
* **Floating-Point Unit (FPU):** The LX7 core includes a single-precision floating-point unit, which accelerates calculations involving non-integer numbers.7 While most optimized embedded models are quantized to use integer arithmetic, the FPU is still valuable during development and for any operations that remain in floating-point. This is a notable advantage over chips like the standard ESP32 (LX6), which lacks a hardware FPU and must perform slower software-based emulation for floating-point math.6
* **Dual-Core Parallelism:** The dual-core design enables true parallel processing. For a complex application like this, a logical and powerful approach is to partition tasks between the two cores. A common strategy is to pin the computationally intensive AI inference pipeline to one core (Core 0), while dedicating the other core (Core 1) to managing the system, handling the camera control loop, and any potential connectivity tasks (e.g., Wi-Fi streaming). This separation prevents the demanding inference task from introducing jitter or latency into the real-time control loop, a critical consideration for system stability.11

### **2.2 Memory Hierarchy: The Critical Role of SRAM, PSRAM, and Cache**

Memory is arguably the most critical resource to manage in this project. The ESP32-S3's memory architecture is a hierarchy of different types of memory, each with distinct characteristics of size and speed.

* **SRAM (Static RAM):** This is the fast, on-chip memory, typically 512 KB in size.6 Due to its low latency, it is the ideal location for performance-critical data, such as task stacks and frequently accessed variables. However, its limited size means it can be quickly exhausted by the large memory requirements of computer vision models.
* **PSRAM (Pseudo-Static RAM):** To overcome the limitations of on-chip SRAM, the ESP32-S3 supports external PSRAM, connected via an SPI interface. This provides a much larger memory pool, with development boards commonly featuring 8 MB of PSRAM.8 However, this extra capacity comes at a significant performance cost. Accessing PSRAM is an order of magnitude slower than accessing on-chip SRAM, making it a potential bottleneck for data-intensive operations.11 For this project, both the camera frame buffer (e.g., ~750 KB for an 800x480 RGB565 display) and the model's tensor arena (potentially several megabytes) will almost certainly need to be placed in PSRAM.10
* **Cache:** To mitigate the performance penalty of accessing slower external memory (both Flash and PSRAM), the ESP32-S3 incorporates configurable instruction and data caches.7 When the CPU needs to access an address in external memory, it first checks the cache. If the data is present (a "cache hit"), access is nearly as fast as internal RAM. If not (a "cache miss"), the data must be fetched from the slower external memory, incurring a significant latency penalty. Properly configuring and maximizing the size of these caches via the development framework is a crucial optimization technique.10

A particularly challenging aspect of this architecture is memory bus contention. The CPU cores, DMA controllers (used for camera and display data transfers), and external memory all share the same SPI bus.27 When multiple components attempt to access PSRAM concurrently, a "traffic jam" occurs, diluting the available bandwidth and introducing latency.11 Furthermore, certain operations, such as writing to the main flash memory, can temporarily disable the cache, effectively stalling any code or data access from external memory. This can be catastrophic for a real-time system. Advanced ESP-IDF configurations like

CONFIG\_SPI\_FLASH\_AUTO\_SUSPEND can help manage this, but they introduce their own timing complexities and are not a silver bullet.28

### **2.3 The ESP-IDF as the Professional's Toolchain**

While the Arduino IDE offers a simplified entry point for microcontroller development, a project of this complexity and with such demanding performance requirements necessitates the power and control of the Espressif IoT Development Framework (ESP-IDF).26 The ESP-IDF is the official development framework from Espressif and provides the low-level access required for system optimization.

The primary tool for configuration within the ESP-IDF is idf.py menuconfig, a terminal-based interface that allows developers to configure hundreds of core system parameters.17 For this project,

menuconfig is essential for:

* Setting the CPU frequency (e.g., to its maximum of 240 MHz).
* Configuring the PSRAM speed and access mode (e.g., Octal SPI at 80 MHz).
* Adjusting the size of the instruction and data caches.
* Enabling and configuring performance-critical libraries, most notably ESP-NN.

The standard development workflow with ESP-IDF involves a few simple commands: idf.py set-target esp32s3 to configure the build for the correct chip, idf.py build to compile the project, and idf.py flash monitor to upload the firmware and view serial output.31 Using ESP-IDF is not merely a matter of preference; it is a hard requirement for unlocking the performance needed for real-time vision tasks.

### **2.4 Unlocking Performance: An Introduction to the ESP-NN Acceleration Library**

The single most important software component for achieving real-time inference speed is the ESP-NN library.17 This is a library provided by Espressif that contains highly optimized implementations of common neural network operations (kernels), such as 2D convolution, depthwise convolution, and pooling.

For the ESP32-S3, these kernels are not written in standard C. Instead, they are hand-optimized in Xtensa assembly language to take full advantage of the LX7 core's vector (SIMD) instructions.17 The performance improvement is dramatic. Benchmarks show that using the ESP-NN optimized kernels can result in speedups of 3x to over 10x for certain operations compared to the generic C implementations provided by standard TensorFlow Lite.17

ESP-NN is designed to integrate seamlessly with TensorFlow Lite for Microcontrollers (TFLM). When enabled in menuconfig (Component config -> ESP-NN -> NN\_OPTIMIZATIONS -> Optimized), the build system automatically links these accelerated kernels, replacing the default, slower ones.17 Failing to enable ESP-NN would mean leaving a potential 10x performance gain on the table, which is the difference between a system that runs at a non-viable 1 FPS and a functional 10 FPS.

| Feature | Specification | Source(s) |
| --- | --- | --- |
| **CPU** | Xtensa® dual-core 32-bit LX7, up to 240 MHz | 7 |
| **AI Acceleration** | Vector Instructions (SIMD) for NN kernels | 23 |
| **On-Chip SRAM** | 512 KB | 6 |
| **External PSRAM** | Up to 32MB Octal SPI support; 8MB common | 8 |
| **External Flash** | Up to 16MB Quad/Octal SPI support | 8 |
| **Peripherals** | DVP Camera Interface, I2C, SPI, LEDC (PWM) | 24 |
| **Connectivity** | 2.4GHz Wi-Fi, Bluetooth LE 5.0 | 12 |
| **Table 1: ESP32-S3 Key Specifications for AI/CV Applications** |  |  |

## **III. Model Selection for On-Device Object Tracking: A Comparative Analysis**

The choice of the neural network architecture is a critical decision that will profoundly impact the system's performance, accuracy, and complexity. The "perception" component must be carefully selected to provide sufficient information for the control system while adhering to the strict resource constraints of the ESP32-S3. Three primary paradigms for object localization are considered: traditional bounding box detection, lightweight centroid detection, and detailed keypoint regression.

### **3.1 The Tracking Paradigm: Bounding Boxes vs. Centroids vs. Keypoints**

Before comparing specific models, it is essential to understand the conceptual differences in what they output and the computational trade-offs involved.

* **Bounding Boxes (e.g., YOLO):** This is the classic approach to object detection. The model outputs four coordinates (x\_min, y\_min, width, height) that define a rectangle enclosing the detected object.39 This provides rich information, including the object's location and its scale, which can be useful for estimating distance. However, regressing these four continuous values is computationally more demanding than simple classification.
* **Centroids (e.g., FOMO):** This is a newer, more lightweight paradigm. Instead of predicting a full bounding box, the model only identifies the center point (centroid) of each object. It achieves this by reframing the detection task as a classification problem on a grid of the output feature map, where each cell in the grid classifies whether it contains an object's center or not. This approach is significantly faster and less memory-intensive.40
* **Keypoints (e.g., Pose Estimation Models):** This is the most granular approach, typically used for tasks like human pose estimation. The model regresses the specific (x, y) coordinates of predefined points of interest on an object, such as the joints of a human body.42 While this provides extremely detailed information and can be robust to occlusions, it is often computationally expensive and may provide an unnecessary level of detail for the task of tracking the general center of action in a sports game.44

### **3.2 Option A: YOLO-based Architectures (e.g., esp-detection)**

The You Only Look Once (YOLO) family of models represents the state-of-the-art in single-shot object detection.

* **Architecture:** YOLO models consist of three main parts: a **backbone** for extracting features from the input image (e.g., CSPDarknet), a **neck** for aggregating features from different scales (e.g., PANet), and a **head** for making the final bounding box and class predictions.46 The  
  esp-detection project from Espressif provides an ultra-lightweight model variant called espdet\_pico, which is based on YOLOv11 and highly optimized for ESP chips.3
* **Pros:** This approach has the potential for high detection accuracy and provides full bounding box information, which could be used for more advanced analytics like estimating player scale. YOLO is an industry-standard architecture with a vast amount of research and documentation available.39
* **Cons:** The primary drawback is the high resource consumption. Even a highly optimized "nano" or "pico" YOLO model can require a tensor arena of several megabytes.10 This large memory footprint forces almost all operations, including model weights, activations, and input buffers, into the slower external PSRAM, creating a significant performance bottleneck and making real-time operation a major challenge.
* **Performance:** The espdet\_pico model is benchmarked at over 6 FPS on an ESP32-S3 with a 224x224 pixel input size.3 While promising, this frame rate is at the lower boundary of what might be considered "real-time" for tracking a fast-paced sport and leaves little margin for other system overhead.

### **3.3 Option B: Centroid-Based Detection (Edge Impulse FOMO)**

FOMO (Faster Objects, More Objects) from Edge Impulse is a novel algorithm designed specifically for object detection on resource-constrained devices.

* **Architecture:** FOMO's innovation lies in its simplicity. It takes a standard, efficient image classification backbone (like MobileNetV2), truncates it before the final classification layers, and appends a simple convolutional head. This head produces a probability heatmap where each cell in the map corresponds to a region in the input image. Instead of regressing bounding box coordinates, FOMO simply classifies each cell as containing an object's centroid or background. This fundamentally changes the problem from complex regression to simple, parallel classification.40
* **Pros:** The main advantage is **extreme speed and efficiency**. FOMO is benchmarked as being up to 30x faster than comparable MobileNet-SSD models and can run in under 200 KB of RAM.40 On an ESP32-S3, inference times of around 40 milliseconds (equivalent to 25 FPS) have been demonstrated, providing a substantial performance margin for the control loop and other system tasks.5 Its fully convolutional nature also means it is flexible with input image resolutions.41
* **Cons:** The most significant limitation is that FOMO **does not output bounding box dimensions**. It only provides the location of the object's center. This makes it impossible to determine the size or scale of the detected object directly from the model's output. Additionally, FOMO performs best when all objects in the scene are of a similar size and are not too close together, as each cell in the output grid can only classify one object.40

### **3.4 Option C: Keypoint Regression Models**

This approach borrows from the domain of human pose estimation, where the goal is to identify the coordinates of specific joints.

* **Architecture:** These models typically use a lightweight CNN backbone, like MobileNet, to extract features from the image. The head of the network is then designed to regress the (x, y) coordinates for a predefined set of keypoints.51
* **Pros:** This method can be very robust to partial occlusions. If some players are hidden, the model can still track the overall group as long as some keypoints (players) are visible. It also provides rich data on the spatial distribution of players.
* **Cons:** This approach is likely overly complex for the stated goal. The objective is to find the "center of action" to aim the camera, not to analyze the precise pose of every individual player. This added complexity comes with a higher computational cost and a more challenging post-processing pipeline, as the detected keypoints would need to be clustered to determine a single center point for the camera to follow.

### **3.5 Expert Recommendation for the Game Tracking Application**

Based on a comparative analysis of the architectures and with the primary constraint of real-time performance on the ESP32-S3 in mind, a clear recommendation emerges.

**Primary Recommendation: Start with FOMO.**

The most significant risk to this project's success is failing to meet the real-time latency requirements. FOMO's demonstrated performance of ~25 FPS on similar hardware provides the largest performance headroom, effectively de-risking this critical aspect of the project.5 While it does not provide bounding box sizes, this is an acceptable trade-off for the initial goal. The "center of action" can be robustly and efficiently calculated in the post-processing step by finding the mean coordinate of all detected player centroids within a given frame. This single coordinate is precisely what is needed to feed the PID control system for the pan-tilt mount. The limitations of FOMO (e.g., preference for similar-sized objects) are also less critical in a sports context, where players on a field generally appear at a similar scale from a fixed camera position.

**Secondary Option: YOLO (espdet\_pico).**

The espdet\_pico model is a strong alternative. Its 6+ FPS performance is viable, and the availability of bounding box data offers more advanced tracking possibilities.3 However, its higher resource requirements present a greater implementation risk. It should be considered the fallback or "upgrade" path if, during prototyping, the lack of object scale information from FOMO proves to be a critical and insurmountable limitation for the control system.

| Architecture | Principle | Pros | Cons | Est. ESP32-S3 Performance | Suitability for Project |
| --- | --- | --- | --- | --- | --- |
| **YOLO (espdet\_pico)** | Single-shot bounding box regression | High accuracy potential; provides object location and scale. | High memory (PSRAM required) and computational cost; lower FPS. | >6 FPS @ 224x224 3 | **Viable, but High Risk.** A good secondary option if scale is essential. |
| **FOMO (Edge Impulse)** | Centroid detection via grid classification | Extremely fast and low memory usage; high FPS; flexible input size. | Does not provide object scale; less effective for overlapping objects. | ~25 FPS @ 96x96 5 | **Highly Recommended.** Best choice for initial prototyping to guarantee real-time performance. |
| **Keypoint Regression** | Regresses (x, y) coordinates of key points | Robust to partial occlusion; provides detailed spatial data. | Computationally expensive; complex post-processing (clustering); likely overkill. | Slower than FOMO/YOLO | **Not Recommended.** Unnecessary complexity for the primary goal. |
| **Table 2: Comparative Analysis of Object Tracking Architectures for ESP32-S3** |  |  |  |  |  |

## **IV. The Development and Deployment Pipeline: From Data to Device**

Successfully deploying a neural network onto a microcontroller is a multi-stage process that begins with data collection and ends with an optimized, executable model running on the target hardware. This section details the complete workflow, covering data preparation, model training and quantization, and the specifics of on-device inference and optimization.

### **4.1 Data Acquisition and Preparation: Leveraging Sports-Specific Datasets**

A high-quality, relevant dataset is the foundation of any successful machine learning model. For this project, several public, large-scale datasets are available that are exceptionally well-suited for training a sports-tracking model.

* **TeamTrack:** This is a premier dataset for this application, featuring over 150 minutes of high-resolution (4K to 8K) video and more than 4 million annotated bounding boxes across soccer, basketball, and handball.53 Crucially, it includes footage from multiple viewpoints, including drone (top-down) and fisheye (side-view), which provides diverse training examples.53 The annotations are provided in standard formats like the MOT Challenge style, which includes  
  frame\_number, object\_id, and bounding box coordinates (x, y, width, height). This object\_id is particularly valuable for developing more advanced tracking algorithms that can follow individual players across frames.56
* **SportsMOT:** This is another large-scale dataset containing over 1.6 million bounding boxes from basketball, volleyball, and football videos.57 It is specifically designed to present challenges common in sports, such as players with very similar appearances (uniforms) and fast, erratic motion, making it an excellent resource for training a robust model.58
* **Handball-Specific Datasets:** For more focused applications, smaller datasets specifically for handball player detection can be found on platforms like Roboflow, which are useful for fine-tuning a model on a specific sport.48

Should custom data be required, the process involves capturing images or video of the target sport and then manually labeling each player in each frame with a bounding box using annotation tools. This is a labor-intensive but necessary step for custom applications.15

### **4.2 Model Training and Quantization**

Once a dataset is prepared, the next step is to train the model and optimize it for the target hardware through quantization.

* The Core Trade-off: Post-Training Quantization (PTQ) vs. Quantization-Aware Training (QAT):  
  Quantization is the process of reducing the precision of the model's weights and activations, typically from 32-bit floating-point numbers (FP32) to 8-bit integers (INT8). This reduces the model size by a factor of four and allows for faster integer-based arithmetic on hardware like the ESP32-S3.14
  + **Post-Training Quantization (PTQ):** This is the simpler method. A fully trained FP32 model is converted to INT8 after training is complete. This process is fast and does not require access to the original training pipeline. However, it can lead to a noticeable drop in model accuracy, as the model was not trained to be robust to the loss of precision.14
  + **Quantization-Aware Training (QAT):** This is a more advanced and generally superior method. The effects of quantization are simulated during the training process itself. This allows the model to learn weights that are more resilient to the precision reduction, resulting in a quantized model with much higher accuracy, often very close to the original FP32 model.14  
      
    For a performance-critical application where every bit of accuracy counts, QAT is the highly recommended approach to achieve the best balance of speed and reliability.
* The ESP-DL and ESP-PPQ Toolchain for Optimized Deployment:  
  Espressif provides a dedicated, high-performance toolchain for deploying AI models, which serves as a powerful alternative to the standard TFLM pipeline.
  + **ESP-DL:** This is a library specifically designed for efficient neural network inference on ESP chips. It uses a proprietary, lightweight model format with the .espdl extension, which is based on FlatBuffers for zero-copy deserialization.67 ESP-DL includes advanced features like a static memory planner to optimize RAM usage and automatic dual-core scheduling for computationally intensive layers like convolutions.67
  + **ESP-PPQ:** This is Espressif's Post-training and Quantization tool. It takes a trained model (in ONNX, PyTorch, or TensorFlow format) and converts it into the optimized .espdl format. ESP-PPQ contains quantization rules specifically tailored to the hardware characteristics of ESP chips, ensuring optimal performance.67
  + **The Workflow:** The recommended high-performance workflow is as follows:
    1. Train your chosen model (e.g., a custom YOLOv11 variant) in a standard framework like PyTorch.
    2. Export the trained model to the ONNX (Open Neural Network Exchange) format.
    3. Use the esp-ppq command-line tool, providing the ONNX model and a small calibration dataset, to perform quantization and generate the final .espdl model file.
    4. Deploy this .espdl file to the device and use the ESP-DL library to run inference.3

### **4.3 On-Device Inference and Post-Processing**

After the model is quantized and converted, it must be integrated into the device's firmware and its raw output must be processed into usable information.

* **The Standard TFLM Workflow:** The conventional approach, often used by platforms like Edge Impulse, involves converting the trained model into the TensorFlow Lite format (.tflite). This file is then converted into a C-style byte array (const unsigned char model\_data = {... };) using a tool like xxd. This array is compiled directly into the application's binary, and the TFLM interpreter is used to load and run the model from this array.26
* **Decoding Raw Model Output Tensors:** A neural network's output is not a clean list of detections. It is a large, multi-dimensional array of numbers, or a tensor. For a YOLOv8-style model, this output might have a shape like ``.74 This tensor must be decoded. The structure is typically  
  [batch\_size, 4\_bbox\_coords + num\_classes, num\_predictions].75 A C++ post-processing function must iterate through this tensor, extract the raw values for the bounding box center (  
  xc, yc), width (w), height (h), and the confidence scores for each class.
* **Implementing Lightweight Non-Maximum Suppression (NMS) in C++:** Object detectors inherently produce multiple, redundant, and overlapping bounding boxes for a single object. NMS is an essential post-processing algorithm that filters these redundant detections to produce a single, high-confidence box per object.39 A lightweight C++ implementation of NMS is critical for performance. The algorithm proceeds as follows:
  1. Discard all predicted boxes with a confidence score below a predefined threshold (e.g., 0.5).
  2. Sort the remaining boxes in descending order based on their confidence scores.
  3. Initialize an empty list for the final, filtered boxes.
  4. Take the box with the highest confidence from the sorted list, add it to the final list, and remove it from the sorted list.
  5. Calculate the Intersection over Union (IoU) of this selected box with all other boxes remaining in the sorted list.
  6. Remove any boxes from the sorted list that have an IoU with the selected box greater than a predefined threshold (e.g., 0.5).
  7. Repeat steps 4-6 until the sorted list is empty.  
     Several lightweight C++ implementations of NMS are available on GitHub that can serve as a reference for creating an efficient version for this project.80

### **4.4 Performance Optimization in Practice**

Achieving real-time performance requires a multi-faceted optimization strategy that goes beyond just the model itself.

* **Maximizing ESP-NN Throughput:** As established, the most significant performance gain comes from ESP-NN. It is imperative to ensure that the "Optimized" versions of the kernels are selected in menuconfig to leverage the hand-tuned assembly code.17 The performance difference is not incremental; it is transformative.

| Function | ANSI C (ticks) | Optimized (ticks) | Opt Ratio |
| --- | --- | --- | --- |
| **convolution** (10x10, 64x1x1x64) | 4,642,259 | 461,398 | **10.06x** |
| **depthwise conv** (18x18, 1x3x3x16) | 1,192,832 | 191,931 | **6.20x** |
| **max pool** (16x16, 1x3x3x16) | 485,714 | 76,747 | **6.33x** |
| **fully connected** (len: 265, ch=3) | 12,290 | 4,439 | **2.77x** |
| **Table 3: ESP-NN Kernel Performance Gains on ESP32-S3 (240MHz)** 17 |  |  |  |

* **Advanced Memory and Cache Management:**
  + **Tensor Arena Placement:** The tensor\_arena is the large block of memory TFLM uses for storing all intermediate tensors during inference.26 For a vision model, this arena will almost certainly be allocated in PSRAM.10 TFLM employs sophisticated memory planning algorithms to reuse memory buffers within this arena, minimizing its peak size. However, its placement in PSRAM remains a performance consideration.19
  + **Manual Memory Placement with IRAM\_ATTR:** The ESP-IDF allows developers to manually place critical functions and data into the fast internal RAM (IRAM/DRAM) using the IRAM\_ATTR attribute. This is a powerful technique for ensuring that performance-critical code, such as the NMS algorithm or other post-processing logic, is not slowed down by being fetched from external flash.10
  + **Cache Tuning:** The ESP32-S3's data and instruction caches can be configured in menuconfig. For a system that relies heavily on external PSRAM and flash, maximizing the cache sizes can significantly improve performance by reducing the number of slow memory accesses.10
  + **Stack Size Optimization:** Each task in FreeRTOS (the underlying operating system of ESP-IDF) has its own stack. These stacks are allocated in SRAM by default. By carefully analyzing and reducing the stack size required for each task, precious SRAM can be freed up for other uses. Tools like uxTaskGetStackHighWaterMark() can be used to determine the actual peak stack usage of a task at runtime, allowing for safe and effective optimization.85

A prudent development strategy would involve two phases. **Phase 1: Rapid Prototyping.** Use a managed platform like Edge Impulse with the FOMO model. This abstracts away much of the complexity of training and deployment, allowing for quick validation of the core concept, especially the camera and control system integration.8

**Phase 2: Performance and Feature Enhancement.** If the prototype proves the concept but requires higher accuracy or more advanced features (like individual player tracking), transition to a custom pipeline. This would involve training a YOLO-based model on the TeamTrack dataset and deploying it using the high-performance ESP-DL/ESP-PPQ toolchain, along with custom C++ post-processing logic.

## **V. Implementing the Pan-Tilt Control System**

With a perception system in place to determine the location of the action, the final step is to translate this information into physical camera movement. This requires interfacing with the camera and servo motors at a low level and implementing a stable control algorithm to ensure smooth, responsive tracking.

### **5.1 Interfacing with the Camera Module via ESP-IDF**

Most ESP32 camera development boards use common camera modules like the OV2640 or OV3660, which connect via a DVP (Digital Video Port) parallel interface.29 The ESP-IDF provides a dedicated camera driver to manage this interface.

The process involves configuring a camera\_config\_t structure with the specific GPIO pins used for the camera's data lines (D0-D7), clock (XCLK), and control signals (SIOD, SIOC). This configuration is then passed to the esp\_camera\_init() function to initialize the driver. Once initialized, a frame can be captured using esp\_camera\_fb\_get(), which returns a pointer to a frame buffer structure containing the image data and its metadata.38

A typical initialization sequence in C++ would look like this:

C++

#**include** "esp\_camera.h"  
  
// Define camera pin configuration based on the specific board  
#**define** CAM\_PIN\_PWDN -1  
#**define** CAM\_PIN\_RESET -1  
#**define** CAM\_PIN\_XCLK 15  
//... other pins  
  
static camera\_config\_t camera\_config = {  
 .pin\_pwdn = CAM\_PIN\_PWDN,  
 .pin\_reset = CAM\_PIN\_RESET,  
 .pin\_xclk = CAM\_PIN\_XCLK,  
 //... other pin assignments  
 .pixel\_format = PIXFORMAT\_RGB565, // Format suitable for many displays and models  
 .frame\_size = FRAMESIZE\_QVGA, // e.g., 320x240  
 .jpeg\_quality = 12,  
 .fb\_count = 2, // Use 2 frame buffers for smoother capture  
 .fb\_location = CAMERA\_FB\_IN\_PSRAM // Allocate frame buffers in PSRAM  
};  
  
esp\_err\_t err = esp\_camera\_init(&camera\_config);  
if (err!= ESP\_OK) {  
 // Handle error  
}

### **5.2 Servo Motor Control using ESP32's PWM Peripherals**

Standard hobby servo motors are controlled by a 50 Hz PWM signal. The angle of the servo's output shaft is determined by the duration of the high pulse within each 20-millisecond period. Typically, a 1 ms pulse corresponds to 0 degrees, a 1.5 ms pulse to 90 degrees (center), and a 2 ms pulse to 180 degrees.20

The ESP32 is equipped with a versatile LEDC (LED Control) peripheral that is perfect for this task. It can generate up to 16 independent, hardware-timed PWM signals, allowing for precise control of multiple servos without consuming CPU cycles.89 Using the ESP-IDF's

ledc driver API is the recommended low-level approach, providing more control than the abstracted Arduino Servo library.90

The C++ implementation involves three main steps:

1. **Configure a Timer:** Set up an LEDC timer with a frequency of 50 Hz.
2. **Configure a Channel:** Associate an LEDC channel with the timer and the specific GPIO pin connected to the servo's signal wire.
3. **Set Duty Cycle:** Calculate and set the duty cycle for the channel to produce the desired pulse width. This is done using ledc\_set\_duty() and then applying the change with ledc\_update\_duty().

### **5.3 Designing a Stable PID Controller in C++ for Smooth Tracking**

Simply commanding the servo to move to a position directly proportional to the tracking error will result in jerky, unstable, and oscillating movement. The camera will constantly overshoot the target and hunt back and forth. To achieve the desired smooth, responsive tracking, a Proportional-Integral-Derivative (PID) controller is essential.22

* **PID Theory:** A PID controller calculates an output value to apply to the system (in this case, a velocity command for the servo) based on three terms calculated from the error (the difference between the desired setpoint and the current measured value).
  + **Proportional (P) Term:** This term provides a response that is directly proportional to the current error. A larger error results in a larger corrective action. Its primary role is to drive the system towards the setpoint. The calculation is P\_out = Kp \* error.94
  + **Integral (I) Term:** This term addresses steady-state error by accumulating past errors over time. If the system consistently falls short of the setpoint, the integral term will gradually increase, providing the extra push needed to eliminate the residual error. The calculation is I\_out += Ki \* error \* dt.95
  + **Derivative (D) Term:** This term acts as a damper by responding to the rate of change of the error. It anticipates the future behavior of the error, slowing the system's response as it approaches the setpoint to prevent overshoot and reduce oscillations. The calculation is D\_out = Kd \* (error - last\_error) / dt.95
* **C++ Implementation:** A simple, self-contained, header-only C++ PID controller class is a practical and reusable solution. The implementation can be based on well-established open-source examples.94 The core of the class will be a  
  calculate() method that takes the setpoint (the center of the camera frame, e.g., coordinate 0) and the process\_variable (the current tracking error in pixels from the perception system) and returns a corrective output value to be sent to the servo.
* **Tuning the PID Gains:** The most challenging part of implementing a PID controller is tuning the gain constants (Kp, Ki, Kd). A practical, manual tuning process is as follows:
  1. Set Ki and Kd to zero.
  2. Gradually increase Kp until the camera's movement begins to oscillate around the target. This is the "proportional-only" response.
  3. Once it oscillates, increase Kd to dampen the oscillations until the movement is smooth and settles quickly without overshooting.
  4. If there is a persistent error (e.g., the camera consistently stops slightly off-center), introduce a very small amount of Ki to correct this steady-state drift.22

| Parameter | Initial Value | Tuning Effect |
| --- | --- | --- |
| **Kp (Proportional Gain)** | Start low (e.g., 0.1) | Governs overall responsiveness. Too high causes rapid oscillation. Too low results in a slow, sluggish response. |
| **Ki (Integral Gain)** | Start at 0 | Corrects for persistent, steady-state error. Too high causes "integral windup," leading to large overshoots. |
| **Kd (Derivative Gain)** | Start at 0 | Dampens oscillations and reduces overshoot. Acts as a "brake." Too high can make the system overly sluggish and sensitive to noise in the error signal. |
| **Table 4: Recommended PID Tuning Starting Points for Camera Pan-Tilt Control** |  |  |

## **VI. Synthesis and Final Recommendations**

This report has detailed an architectural and implementation strategy for developing a real-time, AI-driven panoramic sports tracking system on an ESP32-S3 microcontroller. The solution is built upon three pillars: a perception system to understand the visual scene, an optimization strategy to meet real-time constraints, and a control system to translate digital commands into physical motion.

### **6.1 A Complete Bill of Materials (Hardware and Software Components)**

**Hardware:**

* **Microcontroller:** An ESP32-S3 based development board with integrated PSRAM is essential. The Seeed Studio XIAO ESP32S3 Sense is an excellent choice due to its compact form factor, onboard camera connector, and 8 MB of PSRAM.8
* **Camera Module:** A compatible camera module such as the OV2640 or OV3660, which are widely available and supported by the ESP-IDF camera driver.29
* **Actuators:** Two standard hobby servo motors for the pan and tilt axes.
* **Mechanicals:** A pan-tilt camera bracket to mount the camera and servos.
* **Power Supply:** A dedicated external 5V power supply for the servo motors. Drawing high stall currents from the ESP32's 3.3V or 5V pins can damage the board.91

**Software:**

* **Development Framework:** The Espressif IoT Development Framework (ESP-IDF) is required for low-level control and performance optimization.31
* **Machine Learning Framework:** Either a managed platform like Edge Impulse for rapid prototyping or a custom pipeline using PyTorch/TensorFlow for training and the ESP-DL library for deployment.37
* **Dataset:** A large-scale sports video dataset such as TeamTrack or SportsMOT for training a robust detection model.53
* **Version Control:** Git for managing source code.

### **6.2 Step-by-Step Project Roadmap**

A structured, iterative approach is recommended to manage complexity and mitigate risk.

1. **Environment Setup:** Install and configure the ESP-IDF toolchain. Ensure you can successfully build, flash, and monitor a basic "hello world" example on the ESP32-S3 board.
2. **Control System Validation:** Isolate the mechanical components. Implement the PID controller in C++ and write a simple test program to make the pan-tilt mount follow a predefined pattern (e.g., a slow sine wave). Tune the PID gains (Kp, Ki, Kd) until the movement is smooth and accurate. This validates the control system independently of the perception system.
3. **Perception System Prototyping:** Use a managed platform like Edge Impulse with the FOMO architecture. Collect a small custom dataset or use a subset of a public dataset to train an initial model. Deploy this model as an Arduino library and verify that it can detect objects and output centroid coordinates at a high frame rate (>15 FPS).
4. **Full System Integration:** Integrate the output of the FOMO model (the error in pixels from the screen center) as the input to the PID controller. The system should now be able to track objects in real-time. Fine-tune the PID gains to account for the dynamics of the complete system.
5. **Advanced Enhancement (Optional):** If the performance or features of the FOMO-based prototype are insufficient, begin the transition to a high-performance custom pipeline. Train a lightweight YOLO model (e.g., espdet\_pico) on the full TeamTrack dataset. Use the ESP-PPQ tool to perform Quantization-Aware Training and generate an .espdl model. Replace the TFLM inference code with the ESP-DL library and custom C++ post-processing logic (NMS, etc.).

### **6.3 Anticipating and Mitigating Potential Pitfalls**

* **Performance Under-delivery:** The model runs too slowly to be effective.
  + **Mitigation:** This is the most likely failure mode. Rigorously apply all optimization techniques detailed in Section 4.4. Verify in menuconfig that ESP-NN optimized kernels are enabled. Use the ESP-IDF's profiling tools to identify bottlenecks in custom code. If performance is still inadequate, reduce the input image resolution or choose a smaller model architecture.
* **Control System Instability:** The camera movement is jerky, oscillates wildly, or is unresponsive.
  + **Mitigation:** This is typically a PID tuning issue. Systematically re-tune the Kp, Ki, and Kd gains following the procedure in Section 5.3. Ensure that the main control loop is running at a consistent, predictable interval; use timers rather than simple delays to manage loop timing.
* **Memory Errors and Crashes:** The application crashes with out-of-memory errors or other exceptions.
  + **Mitigation:** The ESP32-S3 has limited SRAM. Use ESP-IDF tools to profile memory usage. Scrutinize all tasks to minimize their stack sizes.85 Check for memory leaks in any custom C++ code, especially where dynamic allocation is used. Ensure the TFLM tensor arena is correctly sized and allocated; an undersized arena is a common cause of crashes. A message like "12 bytes lost due to alignment" suggests the tensor arena pointer should be 16-byte aligned for optimal performance.100

### **6.4 Future Work: Expanding System Capabilities**

Once a stable tracking system is functional, several avenues for enhancement become possible:

* **Advanced Multi-Object Tracking:** Leverage the unique player IDs available in datasets like TeamTrack to train a model capable of re-identification. This would allow the system to track specific players even after temporary occlusions.
* **On-Device Action Recognition:** Implement a temporal model (e.g., a CNN-LSTM) that analyzes a sequence of frames to classify the type of action occurring, such as a shot on goal, a pass, or a foul. This could be used to automatically generate game highlights.1
* **Cloud Connectivity and Analytics:** Utilize the ESP32-S3's built-in Wi-Fi to stream game statistics, event logs, or short video clips of key moments to a cloud-based dashboard for remote analysis and storage.

#### Sources des citations

1. RS2002/ESP32-Realtime-System: A Realtime Wi-Fi Sensing System Demo - GitHub, consulté le juin 14, 2025, <https://github.com/RS2002/ESP32-Realtime-System>
2. ESP32 cam Person Detection - Eloquent Arduino, consulté le juin 14, 2025, <https://eloquentarduino.com/posts/esp32-cam-person-detection>
3. espressif/esp-detection: Lightweight real-time object ... - GitHub, consulté le juin 14, 2025, <https://github.com/espressif/esp-detection>
4. Building a TinyML Object Detection Model for Autonomous Mobile Robots, consulté le juin 14, 2025, <https://seymur.net/post/tinyml-object-detection/>
5. ESP32(S3) cam Object Detection with Edge Impulse FOMO - YouTube, consulté le juin 14, 2025, <https://www.youtube.com/watch?v=4WepPKM3Vh0>
6. Why can seeed studio xiao nrf52840 sense run machine learning models but esp32 cannot, consulté le juin 14, 2025, <https://forum.seeedstudio.com/t/why-can-seeed-studio-xiao-nrf52840-sense-run-machine-learning-models-but-esp32-cannot/276124>
7. ESP32S3 Series - Datasheet - Adafruit, consulté le juin 14, 2025, <https://cdn-shop.adafruit.com/product-files/5477/esp32-s3_datasheet_en.pdf>
8. Seeed XIAO ESP32 S3 Sense - Edge Impulse Documentation, consulté le juin 14, 2025, <https://docs.edgeimpulse.com/docs/edge-ai-hardware/mcu/seeed-xiao-esp32s3-sense>
9. Embedding ML training on MCU - Hackster.io, consulté le juin 14, 2025, <https://www.hackster.io/sologithu/embedding-ml-training-on-mcu-b3cfd2>
10. Using a large model on a ESP32 S3 N16R8 (TFMIC-37) · Issue #94 · espressif/esp-tflite-micro - GitHub, consulté le juin 14, 2025, <https://github.com/espressif/esp-tflite-micro/issues/94>
11. Balancing AI Inference & HMI on ESP32-S3 - Elecrow, consulté le juin 14, 2025, <https://www.elecrow.com/blog/taming-concurrent-ai-inference-and-high-resolution-hmi-on-the-esp32-s3.html>
12. Understanding ESP32-C3 and ESP32-S3: A Comprehensive Guide to Espressif's IoT Powerhouses - OpenELAB, consulté le juin 14, 2025, <https://openelab.io/blogs/learn/understanding-esp32-c3-and-esp32-s3>
13. Top 8 TinyML Frameworks and Compatible Hardware Platforms (TensorFlow Lite, Edge Impulse, PyTorch Mobile, etc.) - DFRobot, consulté le juin 14, 2025, <https://www.dfrobot.com/blog-13921.html>
14. Neural Network Model quantization on Mobile - AI blog - Arm Community, consulté le juin 14, 2025, <https://community.arm.com/arm-community-blogs/b/ai-blog/posts/neural-network-model-quantization-on-mobile>
15. ESP32 cam Object Detection - Eloquent Arduino, consulté le juin 14, 2025, <https://eloquentarduino.com/posts/esp32-cam-object-detection>
16. XIAO ESP32S3 Object Detection - Machine Learning Systems, consulté le juin 14, 2025, <https://mlsysbook.ai/contents/labs/seeed/xiao_esp32s3/object_detection/object_detection.html>
17. espressif/esp-nn: Optimised Neural Network functions for Espressif chipsets - GitHub, consulté le juin 14, 2025, <https://github.com/espressif/esp-nn>
18. Performance - ESP32 - — ESP-IDF Programming Guide v5.4.1 documentation, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/performance/index.html>
19. Optimizing TensorFlow Lite Runtime Memory, consulté le juin 14, 2025, <https://blog.tensorflow.org/2020/10/optimizing-tensorflow-lite-runtime.html>
20. DC Servo Motor Guide - With ESP32 & Arduino - DroneBot Workshop, consulté le juin 14, 2025, <https://dronebotworkshop.com/servoguide/>
21. Servo PID Control of DC Motors Using Encoders, C++, and Arduino - Complete Tutorial, consulté le juin 14, 2025, <https://www.youtube.com/watch?v=3nqAUPBh3AM>
22. owhite/Arduino-PID-control: Use of the PID library to send commands to dc servo and view results using pyplot - GitHub, consulté le juin 14, 2025, <https://github.com/owhite/Arduino-PID-control>
23. ESP32-S3: SIMD Acceleration (ESP-NN/ESP-DSP) Not Effective for run\_classifier - Help, consulté le juin 14, 2025, <https://forum.edgeimpulse.com/t/esp32-s3-simd-acceleration-esp-nn-esp-dsp-not-effective-for-run-classifier/14264>
24. ESP32S3 Development Boards List with Features, Pinouts and more - ESPBoards, consulté le juin 14, 2025, <https://www.espboards.dev/esp32/microcontroller/esp32s3/>
25. DaveBben/esp32-llm: Running a LLM on the ESP32 - GitHub, consulté le juin 14, 2025, <https://github.com/DaveBben/esp32-llm>
26. TensorFlow Lite On ESP32 - OpenELAB, consulté le juin 14, 2025, <https://openelab.io/blogs/learn/tensorflow-lite-on-esp32>
27. Support for External RAM - ESP32-S3 - — ESP-IDF Programming Guide v5.4.1 documentation - Espressif Systems, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/external-ram.html>
28. Concurrency Constraints for Flash on SPI1 - ESP32-S3 - — ESP-IDF Programming Guide v5.4.1 documentation - Espressif Systems, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/peripherals/spi_flash/spi_flash_concurrency.html>
29. Machine Learning at the Edge with ESP32 - Bouvet Norge, consulté le juin 14, 2025, <https://www.bouvet.no/bouvet-deler/machine-learning-with-esp32>
30. XIAO ESP32S3 Sense (Image Classification) - Seeed Wiki, consulté le juin 14, 2025, <https://wiki.seeedstudio.com/tinyml_course_Image_classification_project/>
31. espressif/esp-tflite-micro - 1.2.0 - Example hello\_world, consulté le juin 14, 2025, <https://components.espressif.com/components/espressif/esp-tflite-micro/versions/1.2.0/examples/hello_world?language=en>
32. How to set up and use ESP-NN? - ESP32 Forum, consulté le juin 14, 2025, <https://esp32.com/viewtopic.php?t=42312>
33. ESP32-S3 example - Plumerai Docs, consulté le juin 14, 2025, <https://docs.plumerai.com/2.0/people_detection_micro/esp32s3/>
34. Getting Started With ESP-IDF - Espressif Developer Portal, consulté le juin 14, 2025, <https://developer.espressif.com/blog/getting-started-with-esp-idf/>
35. espressif/esp-tflite-micro: TensorFlow Lite Micro for Espressif Chipsets - GitHub, consulté le juin 14, 2025, <https://github.com/espressif/esp-tflite-micro>
36. espressif/esp-nn • v1.1.1 - ESP Component Registry, consulté le juin 14, 2025, <https://components.espressif.com/components/espressif/esp-nn>
37. Announcing Official Support for the Espressif ESP-EYE (ESP32) - Edge Impulse, consulté le juin 14, 2025, <https://www.edgeimpulse.com/blog/announcing-official-support-for-the-espressif-esp-eye-esp32/>
38. ESP32-S3 Camera Example, consulté le juin 14, 2025, <https://esp32.com/viewtopic.php?t=28126>
39. YOLO Object Detection Explained: Evolution, Algorithm, and Applications - Encord, consulté le juin 14, 2025, <https://encord.com/blog/yolo-object-detection-guide/>
40. Detect objects with centroid (FOMO) - Edge Impulse Documentation, consulté le juin 14, 2025, <https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/computer-vision/object-detection/detect-objects-using-fomo>
41. FOMO: Object detection for constrained devices | Edge Impulse ..., consulté le juin 14, 2025, <https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices>
42. Deep Learning - 020 CNN for keypoints regression - YouTube, consulté le juin 14, 2025, <https://www.youtube.com/watch?v=qeHTQqm3QkQ>
43. Human Pose Estimation aboard Nano-drones Using Tiny Vision Transformers - Webthesis, consulté le juin 14, 2025, <https://webthesis.biblio.polito.it/33873/>
44. Keypoint-MoSeq: parsing behavior by linking point tracking to pose dynamics - bioRxiv, consulté le juin 14, 2025, <https://www.biorxiv.org/content/10.1101/2023.03.16.532307v3.full>
45. Keypoint-MoSeq: parsing behavior by linking point tracking to pose dynamics - PMC, consulté le juin 14, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC11245396/>
46. YOLOv11: An Overview of the Key Architectural Enhancements - arXiv, consulté le juin 14, 2025, <https://arxiv.org/html/2410.17725v1>
47. Guide on YOLOv11 Model Building from Scratch using PyTorch - Analytics Vidhya, consulté le juin 14, 2025, <https://www.analyticsvidhya.com/blog/2025/01/yolov11-model-building/>
48. DEEP YOLOv8-BASED HANDBALL DETECTION SYSTEM WITH TRANSFER LEARNING APPROACH - Jatit.org, consulté le juin 14, 2025, <http://www.jatit.org/volumes/Vol101No22/34Vol101No22.pdf>
49. YOLOv8 Object Tracking and Counting with OpenCV - LearnOpenCV, consulté le juin 14, 2025, <https://learnopencv.com/yolov8-object-tracking-and-counting-with-opencv/>
50. FOMO Self-Attention - Edge Impulse, consulté le juin 14, 2025, <https://www.edgeimpulse.com/blog/fomo-self-attention/>
51. LiPE: Lightweight Human Pose Estimator for Mobile Applications Towards Automated Pose Analysis - ResearchGate, consulté le juin 14, 2025, <https://www.researchgate.net/publication/387063382_LiPE_Lightweight_Human_Pose_Estimator_for_Mobile_Applications_Towards_Automated_Pose_Analysis>
52. Lightweight Human Pose Estimation with Enhanced Knowledge Review, consulté le juin 14, 2025, <https://bmva-archive.org.uk/bmvc/2024/papers/Paper_427/paper.pdf>
53. TeamTrack - Kaggle, consulté le juin 14, 2025, <https://www.kaggle.com/datasets/atomscott/teamtrack>
54. TeamTrack: A Dataset for Multi-Sport Multi-Object Tracking in Full-pitch Videos - CVF Open Access, consulté le juin 14, 2025, <https://openaccess.thecvf.com/content/CVPR2024W/CVsports/papers/Scott_TeamTrack_A_Dataset_for_Multi-Sport_Multi-Object_Tracking_in_Full-pitch_Videos_CVPRW_2024_paper.pdf>
55. TeamTrack: A Dataset for Multi-Sport Multi-Object Tracking in Full-pitch Videos - arXiv, consulté le juin 14, 2025, <https://arxiv.org/html/2404.13868v1>
56. AtomScott/TeamTrack: TeamTrack: An Algorithm and ... - GitHub, consulté le juin 14, 2025, <https://github.com/AtomScott/TeamTrack>
57. SportsMOT - Dataset Ninja, consulté le juin 14, 2025, <https://datasetninja.com/sports-mot>
58. SportsMOT: A Large Multi-Object Tracking Dataset in Multiple Sports Scenes - CVF Open Access, consulté le juin 14, 2025, <https://openaccess.thecvf.com/content/ICCV2023/papers/Cui_SportsMOT_A_Large_Multi-Object_Tracking_Dataset_in_Multiple_Sports_Scenes_ICCV_2023_paper.pdf>
59. Handball\_Player\_Detection Object Detection Dataset and Pre-Trained Model by project, consulté le juin 14, 2025, <https://universe.roboflow.com/project-k1mun/handball_player_detection>
60. Player and Handball Detection Computer Vision Project - Roboflow Universe, consulté le juin 14, 2025, <https://universe.roboflow.com/handball-detection/player-and-handball-detection>
61. A new dataset for spatial, temporal, and tactical analysis of female handball player movements - PMC, consulté le juin 14, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC11609672/>
62. ESP32-CAM Object Detection with Edge Impulse - DroneBot Workshop, consulté le juin 14, 2025, <https://dronebotworkshop.com/esp32-object-detect/>
63. Introducing Post-Training Model Quantization Feature and Mechanics Explained - Datature, consulté le juin 14, 2025, <https://www.datature.io/blog/introducing-post-training-quantization-feature-and-mechanics-explained>
64. Post-training Quantization vs. Quantization-Aware Training - Edge AI and Computing Study Guide 2024 | Fiveable, consulté le juin 14, 2025, <https://library.fiveable.me/edge-ai-and-computing/unit-6/post-training-quantization-vs-quantization-aware-training/study-guide/MvXKnMtRCdos9XsJ>
65. Post-training quantization | Google AI Edge - Gemini API, consulté le juin 14, 2025, <https://ai.google.dev/edge/litert/models/post_training_quantization>
66. Quantization aware training - Model optimization - TensorFlow, consulté le juin 14, 2025, <https://www.tensorflow.org/model_optimization/guide/quantization/training>
67. espressif/esp-dl: Espressif deep-learning library for AIoT applications - GitHub, consulté le juin 14, 2025, <https://github.com/espressif/esp-dl>
68. Getting Started - - — ESP-DL User Guide latest documentation, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-dl/en/latest/getting_started/readme.html>
69. How to deploy YOLO11n - - — ESP-DL User Guide latest documentation, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-dl/en/latest/tutorials/how_to_deploy_yolo11n.html>
70. YOLOX Nano on ESP32 HMI: Real-time Person Counting Guide (Edge AI) - Elecrow, consulté le juin 14, 2025, <https://www.elecrow.com/blog/real-time-person-counting-on-esp32-hmi-with-yolox-nano.html>
71. How to quantize model - - — ESP-DL User Guide latest documentation - Espressif Systems, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-dl/en/latest/tutorials/how_to_quantize_model.html>
72. How to use TensorFlow Lite Micro on UNIHIKER K10 (ESP32 S3) with Arduino IDE, consulté le juin 14, 2025, <https://www.unihiker.com/new-17372.html>
73. TensorFlow Lite With Platform.io and the ESP32 - atomic14, consulté le juin 14, 2025, <https://www.atomic14.com/videos/posts/kZdIO82059E>
74. Understanding YOLOv11 Tensor Ouput Shape for Post-Processing - Stack Overflow, consulté le juin 14, 2025, <https://stackoverflow.com/questions/79426206/understanding-yolov11-tensor-ouput-shape-for-post-processing>
75. Interpreting YOLOv8->TFlite output - Community Help - Roboflow, consulté le juin 14, 2025, <https://discuss.roboflow.com/t/interpreting-yolov8-tflite-output/1562>
76. How to Obtain Bounding Boxes from YOLOv8 Output Tensor Results? - Intel, consulté le juin 14, 2025, <https://www.intel.com/content/www/us/en/support/articles/000098123/software.html>
77. YOLO Object Detection Explained: A Beginner's Guide | DataCamp, consulté le juin 14, 2025, <https://www.datacamp.com/blog/yolo-object-detection-explained>
78. Selecting the Right Bounding Box Using Non-Max Suppression - Analytics Vidhya, consulté le juin 14, 2025, <https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/>
79. Non-Maximum Suppression (NMS) Explained - Ultralytics, consulté le juin 14, 2025, <https://www.ultralytics.com/glossary/non-maximum-suppression-nms>
80. Implementation of Non-Maximum Suppression algorithm - GitHub, consulté le juin 14, 2025, <https://github.com/Nuzhny007/Non-Maximum-Suppression>
81. How to Code Non-Maximum Suppression (NMS) in Plain NumPy - Roboflow Blog, consulté le juin 14, 2025, <https://blog.roboflow.com/how-to-code-non-maximum-suppression-nms-in-plain-numpy/>
82. gdroguski/FastNMS: Fast and simple implementation of Non Maximum Suppression algorithm in C++ with DLL for python usage. - GitHub, consulté le juin 14, 2025, <https://github.com/gdroguski/FastNMS>
83. martinkersner/non-maximum-suppression-cpp - GitHub, consulté le juin 14, 2025, <https://github.com/martinkersner/non-maximum-suppression-cpp>
84. Optimizations of TFLite-micro Memory Management and Allocation Policy, consulté le juin 14, 2025, <https://service.jices.cn/EN/Y2022/V22/I10/11>
85. Minimizing RAM Usage - ESP32-S3 - — ESP-IDF Programming Guide v5.0 documentation, consulté le juin 14, 2025, <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32s3/api-guides/performance/ram-usage.html>
86. Reducing the Power Consumption of Edge Devices Supporting Ambient Intelligence Applications - MDPI, consulté le juin 14, 2025, <https://www.mdpi.com/2078-2489/15/3/161>
87. ESP32-S3 AI Camera Module (Edge Image Recognition, Night Vision, ChatGPT Voice Interaction) - DFRobot, consulté le juin 14, 2025, <https://www.dfrobot.com/product-2899.html>
88. limengdu/SeeedStudio-XIAO-ESP32S3-Sense-camera - GitHub, consulté le juin 14, 2025, <https://github.com/limengdu/SeeedStudio-XIAO-ESP32S3-Sense-camera>
89. Mastering ESP32 Servo Control: Easy Step-by-Step Guide - atomic14, consulté le juin 14, 2025, <https://www.atomic14.com/videos/posts/GcOlBfT7UkM>
90. Control a Servo With ESP32-S3 Sense : 4 Steps - Instructables, consulté le juin 14, 2025, <https://www.instructables.com/Control-a-Servo-With-ESP32-S3-Sense/>
91. Servo Motor Basics with Arduino, consulté le juin 14, 2025, <https://www.arduino.cc/en/Tutorial/Sweep>
92. PID controller with accelerometer and servo motor - General Guidance - Arduino Forum, consulté le juin 14, 2025, <https://forum.arduino.cc/t/pid-controller-with-accelerometer-and-servo-motor/580783>
93. Introduction to PID Controllers - DigiKey, consulté le juin 14, 2025, <https://www.digikey.com/en/maker/projects/introduction-to-pid-controllers/763a6dca352b4f2ba00adde46445ddeb>
94. nicholastmosher/PID: A simple PID library for use with Arduino or other C++ applications., consulté le juin 14, 2025, <https://github.com/nicholastmosher/PID>
95. PID Controllers - Arxterra, consulté le juin 14, 2025, <https://www.arxterra.com/lecture-6-pid-controllers/>
96. PID Controller in C: How to Implement - Simone Bertoni, consulté le juin 14, 2025, <https://simonebertonilab.com/pid-controller-in-c/>
97. cinaral/pid\_control: Header-only C++ library for discrete ... - GitHub, consulté le juin 14, 2025, <https://github.com/cinaral/pid_control>
98. Majid-Derhambakhsh/PID-Library: C/C++ PID Controller library for ARM Cortex M (STM32), consulté le juin 14, 2025, <https://github.com/Majid-Derhambakhsh/PID-Library>
99. Implementation of a simple PID controller in C++ - GitHub, consulté le juin 14, 2025, <https://github.com/LukasLeonardKoening/PID-Controller>
100. ESP32-S3 TensorFlow Lite Model Issues - Stack Overflow, consulté le juin 14, 2025, <https://stackoverflow.com/questions/72770227/esp32-s3-tensorflow-lite-model-issues>
101. Explore Embedded AI with ESP32-S3 and Free PCBA Prototypes - NextPCB Accelerator #4, consulté le juin 14, 2025, <https://www.nextpcb.com/blog/esp32-s3-free-pcba-prototypes-nextpcb-accelerator>