International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

# Vision-Based Tracking Turret using ESP32-CAM and YOLO: A Decoupled Real-Time Computer Vision and Embedded Actuation System

**Sudhamalla Mallesh[1], Suraya Mubeen[2], Ch.Sudhamani [3], R. Mounika[4], K. Bharghavi[5], P. Akhil[6]**
[1,2,3,4,5,6]Department of ECE, CMR Technical campus, Hyderabad, India.
Corresponding Author mail id: mallesh.ece4@gmail.com

**Abstract:**

Autonomous pan-tilt tracking platforms are widely used in surveillance, robotic guidance and human-robot interaction. Commercial Pan-Tilt-Zoom (PTZ) trackers run video decoding, tracking and motor control on one resource-limited processor, which causes control-loop delays of 200–500 ms and limits them to rudimentary detectors such as frame differencing. Running a deep neural network on a microcontroller is not practical either, since it takes hundreds of milliseconds per frame. To overcome this, we propose a vision-based tracking turret with a decoupled architecture. An ESP32-CAM streams live MJPEG video over Wi-Fi to a host PC. The PC runs a You Only Look Once (YOLO) detector through OpenCV, computes the centroid of the detected target and converts the pixel error into pan and tilt servo angles. These angles are sent as short serial commands to an ESP32 board, which drives two SG90 servos and a laser indicator.

The prototype costs about INR 1,940. It achieves person-detection confidence of 0.85–0.98 at 0.5–2 m and an end-to-end latency of about 80–150 ms using only a CPU.

**Keywords: YOLO, ESP32-CAM, OpenCV, pan-tilt tracking, embedded vision**

## I. INTRODUCTION

In the past few years, combining computer vision with low-cost embedded hardware has become an important focus area in robotics and security. Real-time tracking turrets were once built only with expensive hardware such as DSPs, FPGAs or dedicated vision processors, which kept them inside well-funded industrial and research laboratories. Open deep learning frameworks and inexpensive microcontrollers now make it possible for a student team to build a working turret for a few thousand rupees.

However, commercial PTZ trackers still have a basic weakness. They decode video, run the tracking algorithm and generate motor commands on a single embedded processor. This produces control-loop delays of 200–500 ms, which is too slow for fast-moving targets. Most of them also detect motion by background subtraction or frame differencing. These methods fail with changing backgrounds, variable lighting, partial occlusion and several moving objects, and they cannot tell a person

from an animal or a vehicle. Their proprietary firmware also prevents the user from changing the detector or the tracking logic.

To overcome these problems, this work separates intelligence from actuation. The host PC performs all neural-network inference, and the ESP32 hardware is used only for video streaming and servo control. The objectives are: (i) stream video at 15 FPS or more at 320×240 resolution, (ii) detect targets with YOLO at a confidence threshold of 0.50, (iii) map the bounding-box centroid to pan and tilt angles, (iv) drive two servos and a laser indicator through a serial link, and (v) measure latency and detection accuracy of the assembled system.

The main contributions of this work are as follows. First, a decoupled architecture is presented in which a host PC performs all YOLO inference while a low-cost ESP32 board handles only actuation, and a second ESP32 module handles only video capture. Second, a simple and reproducible mapping from the bounding-box centroid to servo angles is described, including a dead zone and a target-lock rule for the laser indicator. Third, the assembled prototype is evaluated for detection confidence, detection rate and stage-wise latency, and its cost is compared with existing approaches. The rest of the paper is organised as follows. Section II reviews related work, Section III describes the hardware and software design, Section IV presents the experimental results, Section V discusses the findings and limitations, and Section VI concludes the paper.

## II. LITERATURE REVIEW

As we are working on a real-time tracking turret, earlier work on object detection and embedded tracking helped us find the limitations of existing methods and decide where to focus. Early detectors relied on hand-crafted features. The Viola–Jones detector [1] used Haar-like features with a cascade of classifiers and gave real-time face detection, but it needs an exhaustive multi-scale sliding-window search, so full-frame latency was 100–400 ms. The combination of Histogram of Oriented Gradients (HOG) and a Support Vector Machine [2] improved pedestrian detection but has the same sliding-window cost, and its accuracy is limited by the hand-designed features.

Deep learning changed this situation. The R-CNN family (2014–2015) reached high accuracy with a two-stage pipeline of region proposals followed by classification. The original R-CNN needed about 47 s per image on a GPU, and even Faster R-CNN [12] reached only about 5 FPS, so the two-stage design sets a latency floor that is too high for a control loop. Single-stage detectors removed the proposal stage. The Single Shot MultiBox Detector (SSD) [3] reached 59 FPS at 74.3% mAP on PASCAL VOC 2007. YOLO [4] divides the image into an S×S grid and predicts boxes and class probabilities in one forward pass, giving 45 FPS at 63.4% mAP on the same benchmark.

The YOLO family has since been refined many times. YOLOv2 added anchor boxes and batch normalisation, YOLOv3 [13] added multi-scale prediction, and YOLOv4 added data-augmentation and loss improvements, reaching 43.5% AP on COCO at 65 FPS [14]. The Ultralytics releases YOLOv5 and YOLOv8 [5] provide PyTorch implementations with simple training and export tools. YOLOv8 further uses a decoupled, anchor-free detection head. Their small variants (YOLOv5s with 7.2M parameters and YOLOv8n with 3.2M) run in about 18–20 ms per frame on a mid-range CPU at 320×240, which fits our latency budget.

On the hardware side, commercial PTZ cameras from Axis, Hikvision and Dahua run the whole pipeline on a single ARM processor and show 150–500 ms delay with background-subtraction tracking. Raspberry Pi based turrets with Haar cascades or MobileNet-SSD have a similar problem, because detection and servo control compete for the same CPU. FPGA and TensorRT based systems reach below 20 ms, but their cost (INR 50,000 and above), power use (15–300 W) and design complexity make them unsuitable for student projects. Even so, a gap remains for a low-cost, open and semantic tracker, and this gap is what the proposed decoupled system addresses. Table 1 compares these approaches with the proposed system.

Table 1: Comparison of existing tracking approaches

| System | Detection Method | Latency | Approx. Cost (INR) |
|---|---|---|---|
| Commercial PTZ camera | Background subtraction | ~200 ms | 1,50,000+ |
| RPi 4 + Haar cascade | Haar cascade | ~180 ms | 15,000 |
| RPi 4 + MobileNet-SSD | MobileNet-SSD | ~220 ms | 18,000 |
| FPGA based (Xilinx DPU) | DNN (TensorRT) | < 20 ms | 80,000+ |
| **Proposed system** | **YOLOv5s / YOLOv8n** | **80–150 ms** | **1,940** |

Beyond detection, the quality of a tracker also depends on the control strategy that turns image error into motion. Many pan-tilt trackers use a proportional or PID controller driven by the pixel error between the target and the image centre, and a Kalman filter [11] is often added to smooth noisy measurements and to predict the target position when detections are lost. For hobby-grade servos, which have limited speed and no position feedback, a simple proportional law with a dead zone is usually adequate and avoids the tuning effort of a full PID loop. Work on TinyML has also shown that very small networks can run directly on microcontrollers, but such models normally work at low resolution, with few classes and lower accuracy than YOLO-class detectors. The decoupled approach used here keeps the accuracy of a YOLO-class detector while the microcontrollers remain simple.

## III. PROPOSED METHOD

The proposed turret divides the workload between two domains. The host-processing domain (PC) performs detection and decision making, and the embedded domain (ESP32 hardware) performs video capture and actuation. The ESP32-CAM streams video over Wi-Fi, the PC detects the target and computes the required angles, and an ESP32 development board converts the received commands into servo and laser signals. The complete signal flow is shown in Figure 1.

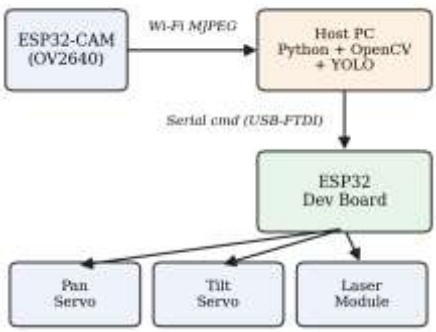



Figure 1: System Block Diagram (signal flow from camera to actuators)

**Design rationale:** The decoupled design follows from the different timing needs of the two tasks. Neural-network inference is computationally heavy but can tolerate a delay of several tens of milliseconds, whereas servo control needs only a few bytes of data and a steady 50 Hz PWM signal. Placing both on one processor makes them compete for CPU time, which is the main cause of the long control delays in monolithic trackers. In the proposed system the PC supplies the compute needed for detection, the OV2640 sensor performs JPEG compression itself so the ESP32-CAM only forwards frames, and the second ESP32 performs only short, deterministic servo updates. Because the camera and the actuator controller are connected only through the PC, either side can be replaced, for example by a different detector or a different actuator, without changing the other.

**Video capture and streaming:** The ESP32-CAM contains an OV2640 sensor [8] and an ESP32 SoC [7] on one board. It joins the local Wi-Fi network in station mode and serves an HTTP MJPEG stream configured at QVGA (320×240). MJPEG compresses every frame independently, so the PC can decode a frame in under 2 ms, unlike H.264 which adds 100–500 ms of encoding delay. The PC opens the stream with OpenCV's VideoCapture class [6] and treats it as a virtual camera. The module is powered from a regulated 5 V rail, because Wi-Fi bursts draw up to 500 mA and can cause brownout resets on a battery supply.

**Detection and centroid computation:** Each decoded frame is passed to a pre-trained YOLOv5s or YOLOv8n model [5], which is configured for the 'person' class. Detections below a confidence threshold of 0.5 are rejected, and the highest-confidence detection is chosen as the target. For a bounding box (x1, y1, x2, y2), the centroid and its displacement from the frame centre are

$$cx = (x1 + x2) / 2, \quad cy = (y1 + y2) / 2$$
$$\Delta x = cx - (W / 2), \quad \Delta y = cy - (H / 2)$$

where W and H are the frame width and height. A dead zone of ±20 pixels is kept around the centre, and no command is issued inside it, which prevents jitter when the target is already centred.

**Effect of the dead zone:** With the sensitivity of 0.28°/pixel described below, the ±20 pixel dead zone corresponds to an angular tolerance of about ±5.6°, so the turret does not react to detection noise smaller than this value. A wider dead zone gives steadier motion but a coarser lock, whereas a narrower one gives a finer lock at the cost of servo chatter. The value of 20 pixels is therefore a compromise for the SG90 servos, which have a finite deadband and gear backlash.

**Pixel-to-angle mapping:** The displacement is scaled by an empirical sensitivity factor in degrees per pixel. For a 320-pixel frame with a 90° field of view this is 90/320 ≈ 0.28°/pixel. The new servo angles are

$$pan = pan_current + (\Delta x \times sensitivity)$$
$$tilt = tilt_current + (\Delta y \times sensitivity)$$

and are clamped to the servo range of 0° to 180°. The neutral position is 90° for both axes.

**Tracking logic:** The tracking logic runs once for every decoded frame, as shown in Algorithm 1 and in the pipeline of Figure 2. When no target is detected, the servos simply hold their last position, so a missed frame does not cause a sudden movement. The lock counter prevents the laser from flickering when the target is only passing through the centre of the frame.

Algorithm 1: Per-frame tracking loop

1. Read the next frame from the MJPEG stream.
2. Run YOLO and keep the detections of class 'person' with confidence of at least 0.5.
3. If no detection remains, keep the last pan and tilt angles and wait for the next frame.
4. Select the highest-confidence detection and compute its centroid (cx, cy).
5. Compute Δx and Δy. If the centroid lies inside the dead zone, increment the lock counter; otherwise reset the counter, update pan and tilt using the sensitivity factor and clamp them to 0°–180°.
6. Set the laser flag to 1 only when the lock counter has reached the required number of consecutive frames; otherwise set it to 0.
7. Send the command P<pan>T<tilt>L<laser> over the serial link and return to step 1.

**Serial protocol and firmware:** The PC sends one ASCII command per frame through PySerial [9] at 115200 baud (8N1) in the format P<pan>T<tilt>L<laser>, for example 'P95T85L1' followed by a newline. At this baud rate a 10–12 byte command takes about 1 ms. The ESP32 firmware reads UART0 until the newline, parses the angles and laser state, and writes them to the servos through the ESP32Servo library [10], which uses the hardware LEDC timer to generate 50 Hz PWM. The whole parse-and-actuate cycle takes under 5 ms. The laser is switched on only after a target lock, meaning the centroid stays inside the dead zone for several consecutive frames.

**Actuators and power:** Two SG90 micro servos (1.8 kg·cm stall torque at 4.8 V) form the 2-DOF pan-tilt mechanism on a plastic bracket. A 5 mW, 650 nm red laser module (KY-008) is mounted on the bracket parallel to the camera axis and serves only as a visual lock indicator. A mini-360 buck converter supplies regulated 5 V from a 7.4 V Li-Po battery with 85–95% efficiency, and 100 µF decoupling capacitors protect the ESP32 from servo current spikes. The ESP32 pins used are GPIO 13 (pan servo), GPIO 12 (tilt servo), GPIO 2 (laser) and RX0 (serial input). The itemised hardware cost is given in Table 2 and totals approximately INR 1,940, which is far below that of the systems in Table 1.

Table 2: Hardware component cost estimation

| Component | Qty | Cost (INR) |
|---|---|---|
| ESP32-CAM module (AI Thinker) | 1 | 350 |
| ESP32 development board | 1 | 400 |
| SG90 servo motor | 2 | 240 |
| Pan-tilt bracket kit (plastic) | 1 | 150 |
| Laser diode module (KY-008) | 1 | 50 |
| Buck converter (mini-360) | 1 | 80 |
| ESP32-CAM-MB programmer board | 1 | 150 |
| Breadboard, jumper wires, capacitors | 1 set | 170 |
| Li-Po pack (7.4 V, 1000 mAh) | 1 | 350 |
| **Total** | | **1,940** |

**Interface summary:** The signal and power connections of the ESP32 control board are listed in Table 3. Since GPIO 2 and GPIO 12 are boot-strapping pins on the ESP32, the laser and tilt-servo lines must not hold them at an unintended level during reset. Fixing the pin map in the firmware means that the PC software needs to know only the serial command format.

Table 3: ESP32 control board pin and power assignment

| Signal | ESP32 pin | Connected to |
|---|---|---|
| Pan servo PWM | GPIO 13 | SG90 servo (pan) |
| Tilt servo PWM | GPIO 12 | SG90 servo (tilt) |
| Laser control | GPIO 2 | KY-008 laser module |
| Serial command input | RX0 (UART0) | USB–serial link to PC |
| Servo and logic supply | 5 V | Buck converter output |
| Camera supply | 5 V | Regulated rail (ESP32-CAM) |

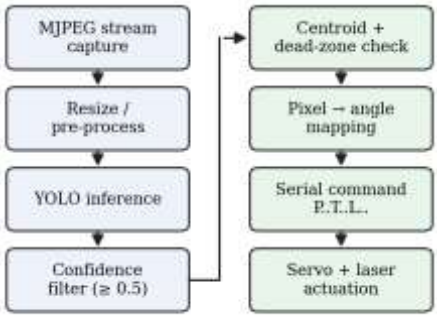



Figure 2: Software Processing Pipeline

**Software implementation:** The PC-side program is written in Python. It uses OpenCV [6] for stream capture, the Ultralytics implementation [5] for inference and PySerial [9] for the command link. A practical point is that OpenCV buffers frames from a network stream, so if inference is slower than the stream, old frames accumulate and the apparent latency grows. Reading frames in a separate thread, or discarding stale frames, keeps the turret reacting to the latest image. The loop contains no blocking call other than inference, so that the camera frame rate, and not the serial link, is the limiting factor.

## IV. EXPERIMENTAL RESULTS

The assembled turret was tested in a controlled indoor laboratory. The buck converter held 5.0 V ± 0.1 V with the servos moving and the laser active, and no brownout resets occurred. The ESP32-CAM connected to Wi-Fi within 3–5 s of power-up. Sweep tests showed smooth servo motion over the full range without binding or gear slip, and the laser dot lined up with the camera axis at the 90°/90° position.

Detection was evaluated with one person as the target, at distances of 0.5 m to 3 m and under different lighting. In well-lit rooms at 0.5–2 m, the confidence score was between 0.85 and 0.98, with stable bounding boxes and smooth servo movement. Beyond 2.5 m the target becomes small in the QVGA frame, so confidence fell to 0.55–0.75 and a few frames were missed, during which the servos simply held their last position. In very dim light, confidence dropped below the 0.5 threshold and tracking stopped, which is expected from the low-light sensitivity of the OV2640 and from using a pre-trained model without night-vision training. The results are summarised in Table 4.

Table 4: Object detection performance under different test conditions

| Test Condition | Avg. Conf. | Detection Rate (%) | Observation |
|---|---|---|---|
| Well-lit, 0.5–1.5 m | 0.92 | 98 | Stable tracking |
| Well-lit, 1.5–2.5 m | 0.79 | 93 | Minor jitter |
| Well-lit, 2.5–3.0 m | 0.61 | 78 | Occasional misses |

| | | | |
|---|---|---|---|
| Low-light, 0.5–1.5 m | 0.48 | 55 | Frequent dropout |
| Partial occlusion | 0.67 | 72 | Reduced box area |

End-to-end latency was measured from timestamps logged at each pipeline stage, averaged over 50 detection events. The breakdown is given in Table 5.

Table 5: System latency component breakdown

| Stage | Avg. Latency (ms) | Share |
|---|---|---|
| Wi-Fi MJPEG frame transfer | 30–60 | ~35% |
| YOLO inference (CPU) | 50–100 | ~45% |
| Centroid and angle mapping | < 2 | < 2% |
| Serial transmission (FTDI USB) | 2–5 | ~3% |
| ESP32 parsing and PWM update | < 5 | ~3% |
| **Total (excl. servo travel)** | **82–172** | **100%** |

Table 5 shows that Wi-Fi transfer and YOLO inference together account for about 80% of the delay, while serial communication and firmware are negligible. Servo travel adds about 150 ms for a 90° move in the worst case, but under 33 ms for corrections smaller than 20°. The typical end-to-end latency was approximately 80–150 ms. This is lower than the 200–500 ms of commercial PTZ trackers, using only a CPU. Moving inference to a CUDA-enabled GPU is expected to bring YOLO time below 10 ms and the total latency below 50 ms.

## V. DISCUSSION AND LIMITATIONS

**Detection performance:** Table 4 shows that detection performance falls steadily as the target moves away. The average confidence drops from 0.92 at 0.5–1.5 m to 0.61 at 2.5–3.0 m, and the detection rate falls from 98% to 78%. This agrees with the geometry of the system: at QVGA resolution a person at 3 m covers only a small number of pixels, and the confidence of YOLO depends strongly on the object size in pixels. The low-light case is the weakest, with a detection rate of 55%, which suggests that the sensor is a larger limitation than the network in dim scenes. Partial occlusion gives 72%, and because the visible box area is reduced, the centroid can move away from the centre of the person and cause small errors in the pointing angle.

**Latency:** The typical latency of 80–150 ms corresponds to roughly one to two frame intervals at 15 FPS (66.7 ms per frame). This is acceptable for a person walking at normal speed but not for fast-moving objects. Since Wi-Fi transfer and YOLO inference make up about 80% of the total (Table 5), improving either one gives the largest gain. A GPU, a smaller input size or a quantised model would reduce inference time, while a wired link or a lower JPEG quality setting would reduce transfer time. The serial link and the firmware together contribute less than 10 ms, which confirms that the decoupled split adds no noticeable overhead.

**Comparison with existing systems:** Compared with the systems in Table 1, the proposed turret has a lower latency than the commercial PTZ camera and the two Raspberry Pi based designs, at about one-eighth of the cost of the cheapest of them. The FPGA based system is faster, but it costs more than 40 times as much. The cost of INR 1,940 does not include the host PC, which is assumed to be already available, and this should be kept in mind when comparing the systems.

**Limitations:** The evaluation was carried out indoors with a single person as the target, so the results do not cover outdoor lighting, several people in the frame or fast motion. The system depends on a stable Wi-Fi link and a host PC, and the SG90 servos give no position feedback, so the control is open-loop. The sensitivity factor is empirical and may need adjustment for a different lens or field of view. The laser is mounted parallel to the camera axis, so a small parallax error exists at short range. The 5 mW, 650 nm module belongs to the visible Class 3R range and must never be pointed at eyes; for demonstrations with people, it can be replaced by an LED or left out.

**Applications:** The platform can be used for surveillance, camera-follow tasks such as lecture recording, human-robot interaction and teaching of embedded vision. Because the detector runs on the PC, the tracked object class can be changed by loading a different model, without any change to the firmware.

## VI. CONCLUSION

In our project, a low-cost wireless pan-tilt turret was designed and tested using an ESP32-CAM for video streaming and a host PC running YOLO for detection. Unlike monolithic embedded trackers, the decoupled architecture keeps heavy neural-network inference away from the motor-control path, so the microcontroller only streams video and drives servos. The system uses off-the-shelf parts costing about INR 1,940, and it can detect semantic object classes instead of just moving pixels. It achieved detection confidence above 0.85 and detection rates above 93% up to 2.5 m in well-lit rooms, with an end-to-end latency of about 80–150 ms. Performance degrades in low light and at long range because of the resolution and sensitivity limits of the OV2640 sensor.

Overall, the proposed turret is a flexible and affordable platform for surveillance, human-robot interaction and academic research. In the future, we can add a Kalman filter [11] or a learned motion predictor to bridge missed detections, use a thermal sensor such as MLX90640 for dark environments, track multiple targets with a priority rule, and move inference to an edge accelerator such as Jetson Nano or Coral to remove the PC and Wi-Fi delay. A web dashboard for remote monitoring and manual override would further improve usability.

## VII. REFERENCES

1. Viola, Paul, and Michael J. Jones. “Robust real-time face detection.” *International Journal of Computer Vision* 57.2 (2004): 137–154.
2. Dalal, Navneet, and Bill Triggs. “Histograms of oriented gradients for human detection.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2005.
3. Liu, Wei, et al. “SSD: Single shot multibox detector.” *European Conference on Computer Vision (ECCV)*. Springer, 2016.
4. Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
5. Ultralytics. “YOLOv8 Documentation.” docs.ultralytics.com. Accessed 2024.
6. Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O’Reilly Media, 2008.
7. Espressif Systems. *ESP32 Technical Reference Manual*, Version 5.1. Espressif Systems, Shanghai, 2023.

8. OmniVision Technologies. *OV2640 Camera Chip Datasheet*. OmniVision Technologies Inc., 2006.

9. PySerial Development Team. "PySerial Documentation." pyserial.readthedocs.io. Accessed 2024.

10. Arduino. "ESP32Servo Library Documentation." madhephaestus.github.io/ESP32Servo. Accessed 2024.

11. Kalman, Rudolf E. "A new approach to linear filtering and prediction problems." *Journal of Basic Engineering* 82.1 (1960): 35–45.

12. Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." *Advances in Neural Information Processing Systems (NeurIPS)* 28 (2015).

13. Redmon, Joseph, and Ali Farhadi. "YOLOv3: An incremental improvement." *arXiv preprint* arXiv:1804.02767 (2018).

14. Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal speed and accuracy of object detection." *arXiv preprint* arXiv:2004.10934 (2020).