Buletin Sistem Informasi dan Teknologi Islam
Vol 7, No 3 (2026), pp 289-300
ISSN: 2721-0901
DOI: https://dx.doi.org/10.33096/busiti.v7i3.3469

Implementasi YOLOv8 dan ESP32 pada Prototipe Sistem Penargetan Militer Berbasis *Real-Time* dengan Integrasi Aktuator Servo

Implementation of YOLOv8 and ESP32 in a Real-Time Military Targeting System Prototype with Servo Actuator Integration

Tedy Firmansyah[a,1,*], Bambang Agus Herlambang[a,2], Noora Qotrun Nada[a,3]

[a]*Program Studi Informatika, Universitas PGRI Semarang, Kota Semarang, Indonesia*
[1]*tedysyhh07@gmail.com;* [2]*bambangherlambang@upgris.ac.id;* [3]*noora@upgris.ac.id;*
**corresponding author*

Informasi Artikel

Diserahkan : 30 April 2026
Diterima : 19 Mei 2026
Direvisi : 8 Agustus 2026
Diterbitkan : 28 Agustus 2026

Kata Kunci:
AI
YOLO
Sistem Penargetan
Prototipe
ESP32

ABSTRAK

Penelitian ini mengembangkan prototipe sistem penargetan otomatis berbasis kecerdasan buatan untuk simulasi laboratorium dengan mengintegrasikan YOLOv8 yang dijalankan pada PC *host*, kamera computer vision, ESP32 sebagai pengendali aktuator servo, dan antarmuka web berbasis Flask. Sistem mendukung deteksi objek secara *real-time*, kontrol proporsional, serta antarmuka web dengan mode manual dan otomatis serta kontrol menggunakan perintah suara. Model utama YOLOv8 yang digunakan adalah redball.pt yang dilatih menggunakan dataset sebanyak 353 gambar, terdiri dari 247 data *train*, 71 data *valid*, dan 35 *data test*. Hasil pengujian menunjukkan inferensi YOLOv8 pada PC *host* mampu berjalan pada kecepatan rata-rata 28,4 FPS serta menghasilkan waktu *settling* rata-rata 2,12 detik, kesalahan posisi 4,8 piksel, dan *overshoot* sebesar 11,2%. Hasil penelitian menunjukkan bahwa prototipe mampu memberikan presisi penargetan yang baik dan fleksibilitas untuk pengembangan sistem penargetan berbasis *Artificial Intelligence* (AI) di lingkungan penelitian dan laboratorium terisolasi.

***Keywords*:**
AI
YOLO
Targeting System
Prototype
ESP32

ABSTRACT

This research develops an artificial intelligence-based automatic targeting system prototype for laboratory simulation by integrating YOLOv8 running on a PC host, a computer vision camera, an ESP32 as servo actuator controller, and a Flask-based web interface. The system supports real-time object detection, proportional control, and a web interface with manual and automatic modes, as well as voice command control. The primary YOLOv8 model used in this study was redball.pt, which was trained on a dataset consisting of 353 images, including 247 training images, 71 validation images, and 35 testing images. Experimental results showed that YOLOv8 inference on the PC host achieved an average speed of 28.4 FPS while achieving an average settling time of 2.12 seconds, a positioning error of 4.8 pixels, and an overshoot of 11.2%. The research results indicate that the prototype is capable of providing good targeting precision and flexibility for the development of Artificial Intelligence (AI)-based targeting systems within a research setting and controlled environment.

I. Pendahuluan

Dalam beberapa tahun terakhir, integrasi kecerdasan buatan telah mentransformasi sistem penargetan militer melalui peningkatan presisi dan kemampuan otonom [1]. Metode penargetan tradisional berbasis prosedur manual masih memiliki keterbatasan, seperti respons lambat dan kerentanan terhadap kondisi lingkungan [2]. Deteksi objek berbasis *deep learning*, terutama *Convolutional Neural Networks* (CNN), kini menjadi solusi utama untuk mengatasi kendala tersebut [3], [4].

Di antara berbagai kerangka kerja, *You Only Look Once* (YOLO) dikenal karena keseimbangan antara kecepatan dan akurasi [5]. Model YOLO memproses citra dalam satu tahap, sehingga cocok untuk sistem *embedded* dengan sumber daya terbatas [6]. Perkembangan terbaru berfokus pada varian ringan untuk perangkat edge [7]. Namun, kesenjangan penelitian terletak pada masih sedikitnya eksplorasi integrasi model YOLO versi terbaru (YOLOv8) yang berjalan pada PC *host* dengan mikrokontroler (ESP32) sebagai

pengendali aktuator servo dalam prototipe penargetan, terutama yang dilengkapi antarmuka web fleksibel untuk mode manual/otomatis [8].

Secara teoretis, YOLOv8 unggul dibanding versi pendahulunya (YOLOv3, v4, v5) dalam ekstraksi fitur spasial berkat arsitektur *anchor-free* dan CSPNet yang lebih efisien, sehingga meningkatkan akurasi lokalisasi objek (khususnya tepi dan objek kecil) serta menurunkan *latency inferensi*, faktor yang krusial bagi aplikasi penargetan *real-time*. Tabel 1 menyajikan komparasi singkat dengan penelitian sebelumnya untuk memperlihatkan posisi penelitian ini.

Tabel 1. Komparasi Penelitian Terdahulu dan Penelitian Ini

Penelitian	Metode Deteksi	Hardware	Akurasi (mAP/Prec)	Antarmuka Web	Kontrol Servo
[1]	YOLOv4/v5	UAV	~0.85 mAP	Tidak	Tidak
[3]	YOLO-G (YOLOv3 improvement)	GPU workstation	+1.2% mAP	Tidak	Tidak
[9]	FMR-CNN + YOLOv8	CCTV surveillance	Tinggi	Tidak	Tidak
[10]	SMCA-α-YOLOv5	Edge device	Efisien	Tidak	Tidak
Penelitian ini	**YOLOv8n**	**ESP32 + Flask**	**Prec 0.99, Recall 1.0**	**Ya (mode manual/otomatis)**	**Ya (3 servo)**

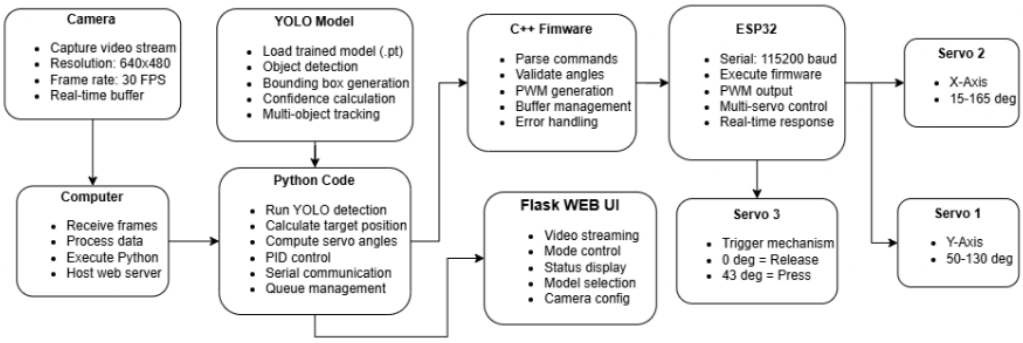
Dari tabel tersebut, terlihat bahwa penelitian ini merupakan satu-satunya yang mengintegrasikan secara utuh YOLOv8, mikrokontroler ESP32, aktuator servo, dan antarmuka web berbasis Flask untuk simulasi sistem penargetan militer. Sebagai pelengkap, penelitian sebelumnya telah menunjukkan efektivitas YOLO dalam pengawasan militer [1]. Model YOLO-G (pengembangan YOLOv3) mampu meningkatkan *mean average* precision (mAP) sebesar 1,2% tanpa mengorbankan performa *real-time* [3]. Dalam konteks pengawasan, model hibrida FMR-CNN dengan YOLOv8 mencapai akurasi tinggi pada sistem CCTV [9]. Pendekatan lain memanfaatkan berbagai versi YOLO untuk mencegah insiden penembakan massal [2], dan kerangka ringan SMCA-α-YOLOv5 dikembangkan untuk efisiensi di lingkungan kompleks [10]. Meskipun demikian, sebagian besar penelitian tersebut belum mengintegrasikan sistem deteksi dengan mikrokontroler dan antarmuka berbasis web untuk kontrol model fleksibel.

Penelitian ini menjembatani kesenjangan tersebut melalui pengembangan prototipe sistem penargetan otomatis berbasis AI sebagai bukti konsep *(proof-of-concept)* dalam lingkungan laboratorium terisolasi, yang mengintegrasikan YOLOv8 pada PC *host* dengan ESP32 berkamera *computer vision*, tiga aktuator servo (sumbu x-y dan mekanisme pemicu), serta antarmuka web Flask dengan mode manual/otomatis dan fitur integrasi model YOLO kustom. Sebagai validasi awal, bola berwarna kontras (merah dan putih) dipilih sebagai *proxy* target karena geometrinya simetris, tepiannya jelas, dan warnanya konsisten, sehingga sifat-sifat ini meminimalkan ambiguitas deteksi dan memungkinkan performa sistem kontrol diukur secara murni, terlepas dari kompleksitas visual target nyata. Pendekatan ini lazim digunakan pada tahap awal pengembangan sistem kontrol berbasis visi komputer sebelum diuji pada target domain spesifik [11].

II. Metode

Proses pengembangan dimulai dari identifikasi permasalahan pada sistem penargetan tradisional, dilanjutkan dengan perancangan arsitektur yang mengintegrasikan *computer vision* dan perangkat keras *embedded* [12]. Komponen utamanya meliputi ESP32 sebagai unit eksekutor mekanis yang menerima perintah nilai sudut dari PC *host* melalui komunikasi serial, kamera untuk akuisisi citra *real-time*, model YOLOv8 untuk deteksi objek, servo untuk pengaturan sumbu x-y dan mekanisme pemicu, serta antarmuka web Flask untuk kontrol pengguna [11], [13]. Perangkat lunak Python pada PC *host* menangani pemrosesan AI dan komunikasi serial, sementara *firmware* C++ pada ESP32 mengeksekusi perintah tersebut dengan menggerakkan servo sesuai nilai sudut yang diterima. Algoritma penargetan menggunakan Kontrol Proporsional *(P-Control)* pada lapisan Python dengan mekanisme *deadzone* untuk mencegah osilasi. Pengujian mencakup simulasi virtual untuk validasi algoritma deteksi dan pengujian fisik pada prototipe untuk mengukur performa *real-time* seperti *frame rate* dan akurasi penargetan dalam kondisi keterbatasan komputasi [14].

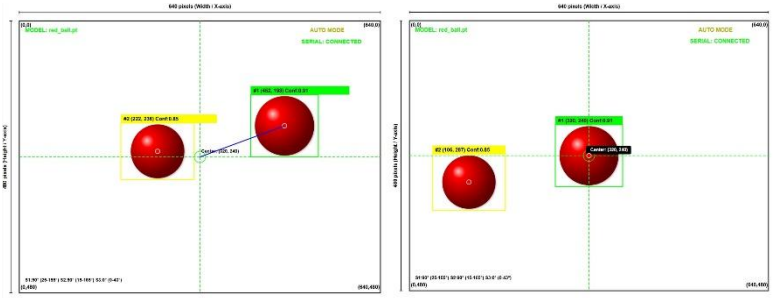
A. Diagram Alur Sistem



Gambar 1. Diagram alur sistem

Berdasarkan Gambar 1, sistem bekerja secara terintegrasi dengan mengambil video *real-time* pada resolusi 640×480 piksel (30 FPS), lalu menjalankan inferensi YOLOv8 untuk menghasilkan *bounding box*, skor kepercayaan, dan pelacakan multi-objek secara simultan.

Frame yang diambil dari buffer kemudian diproses menggunakan Python pada komputer *host*, di mana dilakukan inferensi YOLO untuk menentukan koordinat target, diikuti penerapan kontrol proporsional (*P-Control)*, dan pengiriman instruksi melalui komunikasi serial ke ESP32 [15]. Pada sisi mikrokontroler, perintah diproses melalui tahap parsing, validasi, dan pembangkitan sinyal PWM untuk menggerakkan tiga servo: servo 1 untuk pergerakan sumbu Y (50°–130°), servo 2 untuk sumbu X (15°–165°), dan servo 3 sebagai mekanisme pemicu yang bergerak antara 0° (lepas) dan 43° (tekan). Antarmuka web Flask menyediakan live video, kontrol manual, pergantian model, dan diagnostik sistem [14]. sehingga sistem dapat beroperasi secara otonom maupun dikendalikan secara manual sesuai kebutuhan.



Gambar 2. Visualisasi proses deteksi objek

Gambar 2 menunjukkan visualisasi deteksi objek pada sistem turret otonom dengan resolusi 640×480 piksel. Model YOLO (*redball.pt*) mendeteksi dua objek, yaitu target utama dengan *confidence* 0,91 pada koordinat (452, 193) dan target sekunder dengan *confidence* 0,85 pada koordinat (222, 236). Pusat bidang pandang berada pada koordinat (320, 240) dan ditandai dengan *crosshair* berwarna hijau. Garis biru menunjukkan vektor pergerakan servo menuju target utama. Servo S1 mengendalikan sumbu vertikal (Y) pada rentang 25°–155°, sedangkan Servo S2 mengendalikan sumbu horizontal (X) pada rentang 15°–165°. Algoritma kontrol proporsional digunakan untuk meminimalkan galat posisi antara target dan pusat frame sehingga penargetan berlangsung secara presisi.

B. Spesifikasi Komponen

Penelitian ini memilih setiap komponen berdasarkan kebutuhan sistem dengan mempertimbangkan efisiensi energi, performa *real-time*, kompatibilitas dengan *computer vision*, serta kemudahan integrasi dengan kerangka kerja kecerdasan buatan. Spesifikasi utama dari komponen yang digunakan dijelaskan sebagai berikut:

1. Mikrokontroler ESP32 menggunakan prosesor dual-core Xtensa LX6 240 MHz dengan 520 KB SRAM, Wi-Fi 802.11 b/g/n, Bluetooth v4.2, serta antarmuka UART, SPI, I2C, dan PWM. Konsumsi daya efisien (5V/160 mA aktif, 6 mA deep sleep) dan kemampuan komunikasi serial menjadikannya komponen utama penerima perintah sudut dari Python untuk menggerakkan servo secara real-time [12], [13].
2. Modul kamera USB beresolusi hingga 1080p pada 30 FPS, sudut pandang 75–90°, dengan sensor CMOS yang dioptimalkan untuk pencahayaan rendah. Kamera ini mendukung deteksi objek YOLO berlatensi minimal serta video streaming untuk identifikasi target secara real-time [16].
3. Tiga aktuator servo MG996R menggunakan modulasi PWM dengan torsi 9,4–11 kg·cm pada tegangan 4,8–6V dan kecepatan rotasi 0,17–0,14 detik per 60°, dilengkapi *metal gear train* dan *dual ball bearing* untuk ketahanan mekanis. Komponen ini dipilih karena mampu menghasilkan pergerakan yang halus dan presisi dalam sistem penargetan berbasis kontrol proporsional *(P-Control)* [17].
4. Laptop ThinkPad T480 (Intel Core i5-8350U, RAM 16 GB DDR4, GPU UHD Graphics 620) berfungsi sebagai platform komputasi utama. Framework Flask 3.0 menangani routing, live video feed, dan pengunggahan model YOLO format .pt, sehingga memungkinkan kontrol sistem secara adaptif melalui browser [14].

Spesifikasi tersebut memastikan performa sistem yang optimal melalui integrasi yang harmonis antara kecerdasan buatan dan perangkat keras, sehingga mendukung operasi penargetan otomatis yang andal [15].

C. *Training Model YOLOv8 (redball.pt)*

Model *redball.pt* dilatih menggunakan YOLOv8 nano (*yolov8n*) melalui Roboflow untuk manajemen dataset dan Google Colaboratory dengan GPU NVIDIA T4. Dataset terdiri dari 353 gambar bola merah yang dibagi menjadi 247 data *training* (70%), 71 *validation* (20%), dan 35 *testing* (10%). Tahap *preprocessing* hanya menerapkan *Auto-Orient* tanpa augmentasi tambahan karena variasi data telah diperoleh pada proses akuisisi. Pelatihan menggunakan ukuran citra 640×640 piksel, *batch size* 16, serta ekosistem PyTorch 2.x dengan CUDA 11.8.

Evaluasi pada subset test menghasilkan mAP@50 sebesar 0,9949, mAP@50-95 sebesar 0,8923, *Precision* 0,9901, dan *Recall* 1,0000. Nilai *Recall* sempurna menunjukkan model berhasil mendeteksi seluruh objek target tanpa ada yang terlewat, sementara mAP@50 yang mendekati satu mengkonfirmasi keandalan model untuk diimplementasikan pada sistem penargetan *real-time*.

D. *Deteksi Objek dengan YOLOv8*

Deteksi objek dalam penelitian ini menggunakan kerangka kerja YOLOv8 untuk mengidentifikasi objek pada *frame* citra secara real-time. Model menghasilkan *bounding box* dan skor kepercayaan (*confidence score*), yang kemudian digunakan untuk menghitung koordinat pusat objek serta sebagai dasar pemrosesan kontrol selanjutnya.

1. Koordinat Pusat Objek

   Deteksi objek berbasis YOLO menghasilkan *bounding box* yang didefinisikan oleh koordinat kiri atas (x_1, y_1) dan kanan bawah (x_2, y_2) untuk setiap objek yang terdeteksi. Titik pusat objek dihitung menggunakan persamaan berikut:

$$x_{center} = \frac{(x_1 + x_2)}{2} \tag{1}$$

$$y_{center} = \frac{(y_1 + y_2)}{2} \tag{2}$$

   Di mana x_{center} dan y_{center} merepresentasikan titik tengah objek dalam koordinat piksel.

2. Algoritma Prioritas Multi-Objek

   Dalam kondisi terdapat lebih dari satu objek, sistem menerapkan algoritma prioritas berdasarkan luas *bounding box*. Luas objek dihitung sebagai berikut:

$$A = (x_2 - x_1) \times (y_2 - y_1) \tag{3}$$

   Objek dengan luas terbesar diprioritaskan sebagai target utama:

$$Priority = max\,(A_1, A_2, A_3, \dots, A_n) \tag{4}$$

   Di mana $A_n$merupakan luas objek ke-n dan $n$adalah jumlah total objek yang terdeteksi dalam satu *frame*.

3. Sistem Kontrol Penargetan

   Sistem kontrol penargetan berfungsi untuk mengubah kesalahan posisi visual menjadi gerakan mekanis servo. Proses ini terdiri dari beberapa tahap yang saling terintegrasi untuk memastikan akurasi penargetan yang optimal [18].

4. Normalisasi Koordinat

   Koordinat objek dalam piksel dinormalisasi terhadap pusat *frame* untuk menghasilkan nilai error yang seragam. Pada resolusi 640×480 piksel, pusat *frame* berada pada titik (320, 240). Normalisasi dilakukan dengan persamaan berikut:

$$x_{norm} = \frac{\left(x_{obj} - x_{frame\ center}\right)}{\left(\frac{x_{frame\ center}}{15}\right)} \tag{5}$$

$$y_{norm} = \frac{\left(y_{obj} - y_{frame\ center}\right)}{\left(\frac{y_{frame\ center}}{10}\right)} \tag{6}$$

   Dengan $x_{frame\ center} = 320$piksel dan $y_{frame\ center} = 240$piksel. Nilai normalisasi dibatasi pada rentang $-15 \leq x_{norm} \leq 15$untuk sumbu X dan $-10 \leq y_{norm} \leq 10$untuk sumbu Y.

5. Perhitungan Error

   Error posisi menunjukkan jarak objek terhadap pusat *frame* [19]. Nilai positif menunjukkan objek berada di kanan (sumbu X) atau di atas (sumbu Y), sedangkan nilai negatif menunjukkan sebaliknya. Perhitungan error dinyatakan sebagai berikut:

$$e_x = x_{norm} \tag{7}$$

$$e_y = y_{norm} \tag{8}$$

E. Kontrol Servo

Sistem ini menggunakan kontrol proporsional (*P-control*) untuk menentukan penyesuaian sudut servo. Pemilihan kontrol proporsional didasarkan pada kemampuannya dalam memberikan respons yang cepat dan stabil, tanpa memerlukan proses *tuning* yang kompleks seperti pada kontrol PID [20].

1. Kontrol Proposional

Penerapan kontrol proporsional pada sistem servo telah terbukti efektif dalam berbagai aplikasi, khususnya pada sistem penentuan posisi presisi tinggi dengan beban komputasi yang rendah [21], [22]. Koreksi sudut dihitung menggunakan persamaan berikut:

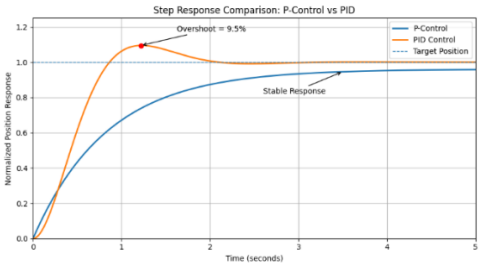
$$\Delta\Theta_x = K_{p\_x} \times e_x \quad (9)$$

$$\Delta\Theta_y = K_{p\_y} \times e_y \quad (10)$$

Di mana $K_{p_x} = 2.0$ dan $K_{p_y} = 1.5$merupakan konstanta proporsional yang diperoleh melalui proses *tuning* empiris. Nilai K_{p_x}dibuat lebih besar dibandingkan K_{p_y}karena pergerakan pada sumbu horizontal membutuhkan respons yang lebih cepat untuk mengikuti objek yang bergerak.

2. Grafik Step Response P-Control vs PID

Gambar 3 mengilustrasikan perbedaan karakteristik respons langkah *(step response)* antara *P-control* dan PID terhadap perubahan posisi target secara tiba-tiba. PID memiliki *rise time* lebih cepat namun menghasilkan *overshoot* sebelum stabil, sedangkan *P-control* memberikan respons lebih halus dengan *overshoot* minimal meski konvergensinya lebih lama. Berdasarkan karakteristik tersebut, penelitian ini memilih *P-control* karena lebih ringan secara komputasi dan cukup stabil untuk implementasi sistem penargetan *real-time* pada ESP32. Gambar ini merupakan ilustrasi konseptual, bukan hasil pengujian eksperimental.



Gambar 3. Ilustrasi konseptual perbandingan karakteristik step response antara *P-control* dan PID

3. Pembaruan Posisi Servo

Posisi servo diperbarui berdasarkan nilai koreksi sudut yang telah dihitung. Servo 2 mengontrol pergerakan horizontal, sedangkan Servo 1 mengatur pergerakan vertikal. Persamaan pembaruan posisi adalah sebagai berikut:

$$\Theta_{servo2}(t+1) = \Theta_{servo2}(t) - \Delta\Theta_x \quad (11)$$

$$\Theta_{servo1}(t+1) = \Theta_{servo1}(t) - \Delta\Theta_y \quad (12)$$

Di mana t menunjukkan waktu saat ini dan $t + 1$menunjukkan waktu berikutnya. Tanda negatif pada persamaan menunjukkan adanya perbedaan arah antara sistem koordinat citra dan arah gerakan servo.

4. *Deadzone Anti-Oscillation*

Untuk mencegah osilasi (*hunting*) ketika objek berada di dekat pusat *frame*, sistem menerapkan mekanisme *deadzone* [23]. Jika nilai error berada dalam rentang *deadzone*, maka tidak dilakukan koreksi sudut:

$$\Delta\Theta_x = 0, \text{if } |e_x| < D_x \quad (13)$$

$$\Delta\Theta_y = 0, \text{if } |e_y| < D_y \quad (14)$$

Dengan $D_x = 1.0$dan $D_y = 1.0$sebagai ambang batas *deadzone*. Nilai ini dipilih untuk menjaga keseimbangan antara akurasi penargetan dan stabilitas sistem.

5. Batasan Servo untuk Keamanan

Untuk menjaga keandalan operasional serta mencegah kerusakan mekanis, pergerakan sudut setiap servo dibatasi dalam rentang tertentu:

$$25° \leq \Theta_{servo1} \leq 155° \quad (15)$$

$$15° \leq \Theta_{servo2} \leq 165° \quad (16)$$

Batasan ini memastikan bahwa pergerakan servo tetap berada dalam zona aman, sehingga menghindari benturan dengan komponen lain serta menjaga stabilitas sistem secara keseluruhan.

F. *Mekanisme Pelacakan*

Sistem ini dilengkapi dengan mekanisme pelacakan (*tracking*) untuk menangani kondisi ketika objek tidak terdeteksi sementara, misalnya akibat *occlusion* atau keterbatasan deteksi YOLO [24]. Mekanisme ini memungkinkan sistem tetap mempertahankan estimasi posisi target dalam jangka waktu tertentu.

1. Prediksi Linear

Ketika objek tidak terdeteksi, sistem menggunakan metode prediksi linear berdasarkan data posisi sebelumnya. Kecepatan objek dihitung dari dua *frame* terakhir menggunakan persamaan berikut:

$$V_x = x(\mathrm{t}-1) - x(\mathrm{t}-2) \quad (17)$$

$$V_y = y(\mathrm{t}-1) - y(t-2) \quad (18)$$

Estimasi posisi objek pada *frame* berikutnya dihitung sebagai:

$$x_{pred} = x(t-1) + V_x \quad (19)$$

$$y_{pred} = y(t-1) + V_y \quad (20)$$

Prediksi ini hanya digunakan hingga maksimal 10 *frame* berturut-turut. Jika objek tetap tidak terdeteksi setelah batas tersebut, sistem akan menghentikan pelacakan dan kembali ke mode pemindaian (*scanning mode*).

2. Deteksi *Timeout*

Apabila objek tidak terdeteksi dalam waktu tertentu ($T_{timeout} = 5.0$ detik), sistem secara otomatis mengembalikan posisi servo ke koordinat netral (90°, 90°). Hal ini bertujuan untuk mempersiapkan sistem dalam siklus deteksi berikutnya.

G. *Parameter Tuning*

Tabel 2. Parameter sistem

Parameter	Simbol	Nilai	Unit
Frame Width	W_{frame}	640	*Pixels*
Frame Height	H_{frame}	480	*Pixels*
Proportional Gain X	K_{p_x}	2.0	-
Proportional Gain Y	K_{p_y}	1.5	-
Deadzone X	D_x	1.0	-
Deadzone Y	D_y	1.0	-
Servo Update Interval	T_{servo}	0.55	*seconds*
Detection Interval	T_{detect}	0.2	*seconds*
Detection Timeout	$T_{timeout}$	5.0	*seconds*
Maximum Lost Frames	F_{max_lost}	10	*frames*

Parameter K_{p_x}dan K_{p_y}ditentukan melalui metode *trial-and-error* dengan mempertimbangkan waktu respons dan besarnya *overshoot*. Nilai $K_{p_x} = 2.0$memberikan respons cepat pada sumbu horizontal tanpa menghasilkan *overshoot* yang berlebihan, sedangkan $K_{p_y} = 1.5$ digunakan karena sumbu vertikal lebih sensitif dan memerlukan pergerakan yang lebih halus.

Nilai *deadzone* (D_x dan D_y) diperoleh melalui analisis empiris terhadap pola osilasi sistem. Penetapan nilai 1.0 terbukti mampu mengurangi fenomena *hunting* tanpa mengorbankan akurasi penargetan. Interval pembaruan servo $T_{servo} = 0.55$detik disesuaikan dengan karakteristik perangkat keras untuk menghindari beban mekanis berlebih. Proses *tuning* ini mengikuti pendekatan *step-response* yang umum digunakan pada sistem servo komersial [25].

H. *Algoritma Sistem Secara Keseluruhan*

Sistem penargetan otomatis bekerja secara siklik dengan interval deteksi $T_{detect} = 0.2$detik. Pada setiap siklus, sistem mengambil *frame* beresolusi 640×480 piksel yang kemudian diproses oleh model YOLO untuk menghasilkan *bounding box* (x_1, y_1, x_2, y_2). Selanjutnya dihitung koordinat pusat objek (Persamaan 1–2) dan luas area (Persamaan 3), lalu objek dengan luas terbesar (Persamaan 4) dipilih sebagai target utama.

Koordinat pusat kemudian dinormalisasi (Persamaan 5–6) dan digunakan untuk menghitung error posisi (e_x, e_y) (Persamaan 7–8). Nilai error ini dievaluasi menggunakan kondisi *deadzone* (Persamaan 13–14) untuk menentukan apakah diperlukan koreksi sudut $\Delta\theta$ (Persamaan 9–10). Posisi servo kemudian diperbarui (Persamaan 11–12) dengan mempertimbangkan batas sudut aman (Persamaan 15–16), sebelum perintah dikirim ke mikrokontroler.

Jika objek tidak terdeteksi, sistem akan menggunakan prediksi posisi (Persamaan 18–21). Apabila jumlah *frame* yang hilang melebihi batas F_{max_lost}, maka pelacakan dihentikan dan sistem kembali ke posisi netral melalui mekanisme *timeout*. Proses ini berlangsung secara berulang untuk memastikan sistem mampu mempertahankan akurasi, adaptabilitas, dan respons real-time dalam berbagai kondisi operasional.

I. Evaluasi Sistem

Evaluasi sistem dilakukan untuk memvalidasi kinerja sistem penargetan otomatis melalui pengukuran akurasi penargetan secara menyeluruh.

1. Evaluasi Akurasi Penargertan

   Evaluasi ini mengukur seberapa presisi sistem dalam mengarahkan servo menuju target yang terdeteksi. Parameter yang digunakan meliputi *settling time*, *positioning error*, dan *overshoot*.

2. *Settling Time*

   Settling time didefinisikan sebagai waktu yang dibutuhkan sistem untuk mencapai dan mempertahankan posisi target dalam toleransi ±5 piksel dari pusat *frame*. Perhitungan dilakukan dengan persamaan berikut:

$$T_{settling} = t_{final} - t_{initial} \quad (21)$$

   Di mana $t_{initial}$adalah waktu saat objek pertama kali terdeteksi dan t_{final}adalah waktu ketika error posisi masuk ke dalam zona toleransi dan tetap stabil.

3. *Positioning Error*

   Positioning error mengukur jarak antara pusat objek dengan pusat *frame* setelah sistem mencapai kondisi stabil. Error dihitung pada masing-masing sumbu:

$$E_x = |x_{target} - x_{center}| \quad (22)$$

$$E_y = |y_{target} - y_{center}| \quad (23)$$

   Total error dihitung menggunakan jarak Euclidean:

$$E_{total} = \sqrt{(E_{x^2} + E_{y^2})} \quad (24)$$

4. *Overshoot*

   Overshoot mengukur seberapa jauh pergerakan servo melampaui posisi target sebelum mencapai kondisi stabil. Persentase *overshoot* dihitung dengan persamaan:

$$OS(\%) = \left[\frac{\Theta_{peak} - \Theta_{target}}{\Theta_{target}}\right] \times 100\% \quad (25)$$

   Di mana θ_{peak}adalah sudut maksimum yang dicapai servo, dan θ_{target} adalah sudut target akhir.

J. Skenario Pengujian

Pengujian dilakukan dengan menempatkan objek target (bola merah) pada delapan posisi berbeda dalam *frame* berukuran 640×480 piksel. Setiap skenario diuji sebanyak 20 kali untuk memperoleh data statistik yang representatif. Sistem selalu dimulai dari posisi servo netral (90°, 90°), dan waktu yang dibutuhkan untuk mencapai kondisi stabil dicatat sebagai *settling time*.

Tabel 3. Skenario pengujian Akurasi Penargetan

No	*Initial Object Position*	*Coordinates (x, y) Pixels*	*Number of Trials*	*Distance from Center (Pixels)*
1	*Upper Left*	(160, 120)	20	200
2	*Upper Right*	(480, 120)	20	200
3	*Lower Left*	(160, 360)	20	200
4	*Lower Right*	(480, 360)	20	200
5	*Center Left*	(160, 240)	20	160
6	*Center Right*	(480, 240)	20	160
7	*Center Top*	(320, 120)	20	120
8	*Center Bottom*	(320, 360)	20	120

Setiap skenario diulang sebanyak 20 kali untuk memperoleh nilai rata-rata (*mean*) dan standar deviasi. Kriteria keberhasilan sistem ditetapkan sebagai berikut: *settling time* < 3 detik, *total error* < 10 piksel, dan *overshoot* < 20%.

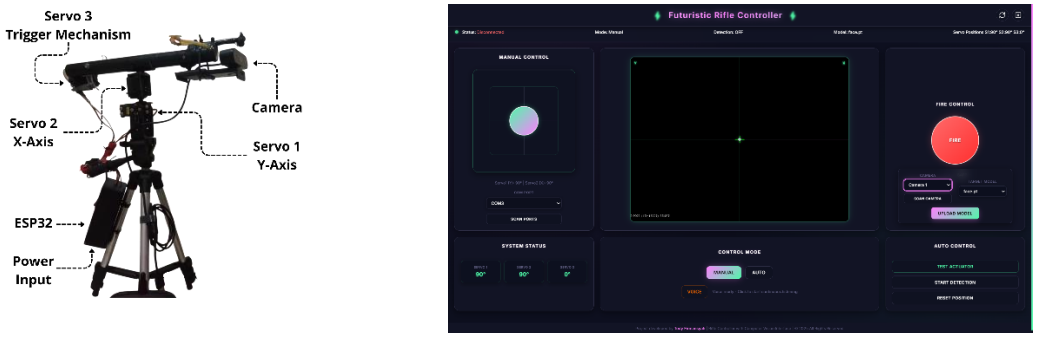
III. Hasil dan Pembahasan

A. Implementasi Sistem

Implementasi mencakup prototipe perangkat keras, antarmuka web Flask, dan integrasi *end-to-end*, disertai evaluasi akurasi penargetan untuk memastikan sistem mampu beroperasi otonom maupun manual.

B. Prototipe Perangkat Keras dan Antar Muka Flask

Gambar 4 menampilkan prototipe perangkat keras beserta antarmuka web berbasis Flask yang telah terintegrasi dalam sistem penargetan otomatis. Arsitektur mekanik, penempatan komponen utama, serta tampilan kontrol sistem divisualisasikan secara jelas untuk memudahkan pemahaman terhadap konstruksi fisik dan mekanisme operasional sistem secara keseluruhan.



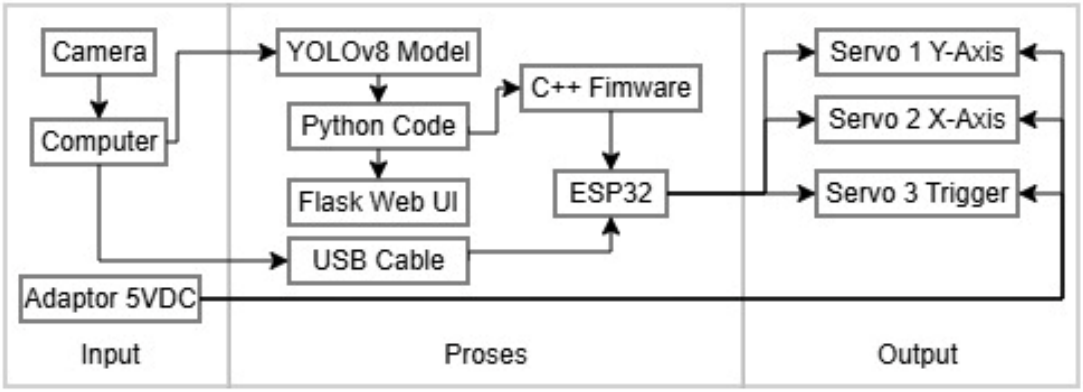
(a) (b)

Gambar 4. (a) Prototipe perangkat keras, (b) Antarmuka web Flask

Prototipe dibangun dengan tiga servo MG996R: servo 1 (vertikal, 25°–155°), servo 2 (horizontal, 15°–165°), dan servo 3 (pemicu, 0°–43°). ESP32 berfungsi sebagai unit eksekutor mekanis yang menerima perintah nilai sudut dari PC *host* melalui komunikasi serial 115200 baud, kemudian mengubahnya menjadi sinyal PWM untuk menggerakkan servo secara langsung. Kamera USB 640×480 piksel pada 30 FPS dipasang di bawah laras, dan struktur pan-tilt dua sumbu dengan mounting bracket khusus menjaga stabilitas pergerakan.

Antarmuka web berbasis Flask 3.0 menyediakan kontrol sistem secara terintegrasi dengan latensi kurang dari 100 ms. Bagian kiri atas menampilkan virtual joystick, pemilihan COM, dan *scan ports* untuk operasi manual maupun otomatis. Bagian tengah menampilkan live video feed lengkap dengan bounding box YOLO, nilai confidence, crosshair pada koordinat (320, 240), serta informasi FPS secara real-time. Di bawahnya terdapat tombol pilihan mode manual/otomatis serta button untuk mengaktifkan kontrol sistem dengan perintah suara. Sementara itu, bagian kanan menyediakan kontrol servo pemicu, pemilihan sekaligus scan kamera, pemilihan serta *upload* model YOLO format .pt, serta tombol tambahan seperti LOCK ACTIVATION, START DETECTION, dan RESET POSITION ke posisi netral (90°, 90°). Penggunaan Flask dipilih karena ringan dan mampu mendukung sistem kontrol IoT *real-time* secara stabil [16].

C. Integrasi Sistem End-to-End



Gambar 5. Diagram blok arsitektur terintegrasi *hardware-software*

Frame video dari kamera dikirim via USB ke skrip Python pada PC *host*, di mana YOLOv8 melakukan deteksi objek, lalu koreksi sudut servo dihitung menggunakan kontrol proporsional dan dikirim ke ESP32 via serial 115200 baud. ESP32 mengubah perintah tersebut menjadi sinyal PWM untuk menggerakkan servo, sementara Flask menyajikan live video dengan overlay deteksi ke antarmuka web dan mendukung peralihan antara mode manual dan otomatis.

Total latensi sistem berkisar 0,75 detik, yang terdiri dari 0,2 detik untuk proses deteksi dan 0,55 detik untuk pembaruan servo. Nilai ini menunjukkan bahwa sistem telah memenuhi kriteria respons *real-time*.

D. Evaluasi dan Pengujian Akurasi Penargetan

(a) (b)

Gambar 6. Deteksi bola merah, (b) Deteksi bola putih.

Evaluasi akurasi penargetan dilakukan untuk menguji kemampuan sistem dalam mengarahkan servo ke target yang terdeteksi YOLO menggunakan tiga metrik utama, yaitu settling time, positioning error, dan overshoot pada delapan skenario posisi objek. Pengujian menggunakan model utama YOLOv8 yang dilatih untuk mendeteksi bola merah serta model lain untuk mendeteksi bola putih agar variasi objek lebih beragam. Gambar 7 menunjukkan antarmuka kamera sistem beserta bounding box, nilai confidence, dan logika prioritas target.

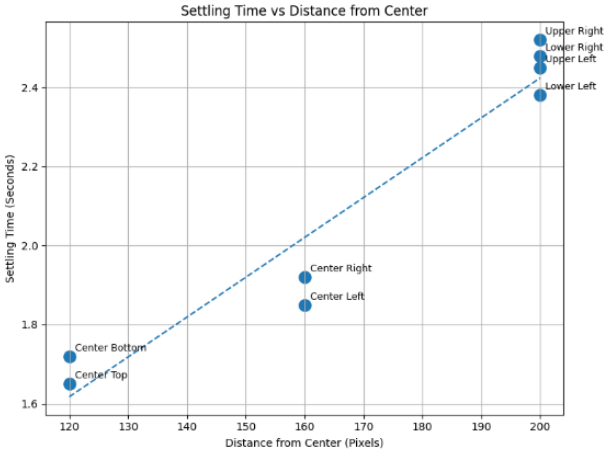
1. Hasil Pengukuran *Settling Time*

Settling time mengukur waktu yang dibutuhkan sistem untuk mencapai dan mempertahankan posisi target dalam toleransi ±5 piksel dari pusat *frame* (320, 240).

Tabel 4. Pengukuran *settling time*

No	*Object Position*	*Average Settling Time (s)*	*Standard Deviation (s)*	*Min (s)*	*Max (s)*
1	*Upper Left*	2.45	0.18	2.15	2.85
2	*Upper Right*	2.52	0.21	2.20	2.95
3	*Lower Left*	2.38	0.16	2.10	2.75
4	*Lower Right*	2.48	0.19	2.18	2.88
5	*Center Left*	1.85	0.12	1.65	2.15
6	*Center Right*	1.92	0.14	1.70	2.25
7	*Center Top*	1.65	0.10	1.50	1.90
8	*Center Bottom*	1.72	0.11	1.55	2.00
Overall Average		2.12	0.15	1.88	2.47

Hasil pengujian menunjukkan rata-rata settling time sebesar 2,12 detik, sehingga memenuhi kriteria < 3 detik. Objek yang berada dekat pusat *frame* memiliki settling time lebih cepat (±1,65–1,72 detik), sedangkan objek di area sudut membutuhkan waktu lebih lama (±2,38–2,52 detik). Kondisi ini disebabkan oleh jarak perpindahan sudut servo dan error posisi awal yang lebih besar, sehingga kontrol proporsional menghasilkan koreksi yang lebih agresif dan memerlukan lebih banyak siklus untuk mencapai kondisi stabil.



Gambar 7. Scatter plot settling time vs distance

Scatter plot pada Gambar 7 menunjukkan adanya kecenderungan peningkatan settling time seiring bertambahnya jarak objek dari pusat frame. Hal ini mengindikasikan bahwa performa sistem dipengaruhi oleh besarnya perpindahan mekanis servo dan nilai error yang diproses oleh kontrol proporsional.

2. Hasil Pengukuran *Positioning Error*

Positioning error dihitung sebagai jarak Euclidean antara pusat objek dan pusat *frame* setelah sistem stabil.

Tabel 5. Pengukuran *positioning error*

No	*Object Position*	*Average X Error (pixels)*	*Average Y Error (pixels)*	*Average Total Error (pixels)*	*Standard Deviation (pixels)*
1	*Upper Left*	4.2	3.8	5.7	1.2
2	*Upper Right*	4.5	3.5	5.7	1.3
3	*Lower Left*	3.8	4.2	5.7	1.1
4	*Lower Right*	4.3	4.0	5.9	1.4
5	*Center Left*	3.2	2.8	4.3	0.9
6	*Center Right*	3.5	2.5	4.3	0.8
7	*Center Top*	2.8	2.2	3.6	0.7
8	*Center Bottom*	2.5	2.5	3.5	0.6
Overall Average		3.6	3.2	4.8	1.0

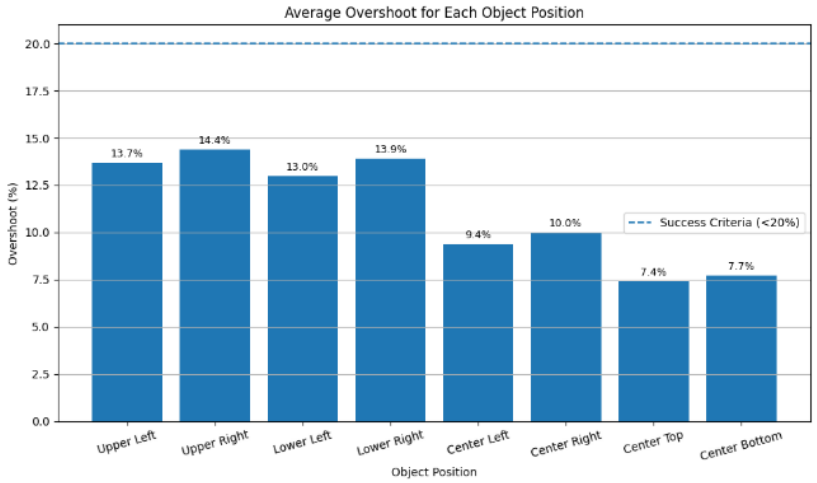
Rata-rata error sebesar 4,8 piksel, jauh di bawah batas 10 piksel, menunjukkan akurasi tinggi. Error terkecil terjadi di area tengah (±3,5 piksel), sedangkan error terbesar terjadi di sudut (±5,7–5,9 piksel).

3. Hasil Pengukuran *Overshoot*

Overshoot mengukur seberapa jauh servo melewati target sebelum stabil.

Tabel 6. Pengukuran *Overshoot*

No	*Object Position*	*X Overshoot (%)*	*Y Overshoot (%)*	*Average Overshoot (%)*	*Status*
1	*Upper Left*	12.5	14.8	13.7	✓*Pass*
2	*Upper Right*	13.2	15.5	14.4	✓*Pass*
3	*Lower Left*	11.8	14.2	13.0	✓*Pass*
4	*Lower Right*	12.8	15.0	13.9	✓*Pass*
5	*Center Left*	8.5	10.2	9.4	✓*Pass*
6	*Center Right*	9.2	10.8	10.0	✓*Pass*
7	*Center Top*	6.5	8.2	7.4	✓*Pass*
8	*Center Bottom*	6.8	8.5	7.7	✓*Pass*
Overall Average		10.2	12.2	11.2	✓*Pass*



Gambar 8. *Bar chart* perbandingan *overshoot*

Nilai rata-rata *overshoot* sebesar 11,2%, masih dalam batas aman (< 20%). *Overshoot* pada sumbu Y sedikit lebih tinggi karena nilai gain yang lebih kecil ($K_{p_y} = 1.5$) untuk menjaga kestabilan vertikal.

4. Analisis Performa Sistem

Evaluasi terhadap ketiga metrik menunjukkan bahwa sistem penargetan otomatis bekerja dengan sangat baik, dengan tingkat keberhasilan 100% dari 160 percobaan.

Tabel 7. Evaluasi performa sistem

Parameter	*Success Criteria*	*Measurement Results*	*Status*	*Success Percentage*
Settling Time	*< 3.0 seconds*	*2.12 ± 0.15 seconds*	✓*Meets*	100% (160/160)
Positioning Error	*< 10 pixels*	*4.8 ± 1.0 pixels*	✓*Meets*	100% (160/160)
Overshoot	*< 20%*	*11.2%*	✓*Meets*	100% (160/160)

Analisis menunjukkan bahwa semakin dekat posisi awal objek dengan pusat frame, semakin cepat sistem stabil dan semakin kecil error. Nilai deviasi standar yang rendah mengindikasikan konsistensi dan repeatability tinggi. Kontrol proporsional dengan parameter hasil tuning empiris terbukti stabil tanpa

osilasi berlebih, sementara deadzone efektif mengurangi hunting tanpa menurunkan responsivitas. Pada ThinkPad T480 (Core i5-8350U, RAM 16 GB), sistem mencapai kecepatan inferensi rata-rata 28,4 FPS pada resolusi 640×480 menggunakan model YOLOv8n (CPU-*only*).

E. Keterbatasan Dataset dan Validitas Pengujian

Perlu ditegaskan bahwa seluruh hasil pengujian pada penelitian ini merupakan validasi awal prototipe yang dilakukan dalam kondisi laboratorium terisolasi (*controlled environment*), bukan pengujian operasional lapangan. Dataset pelatihan yang berjumlah 353 gambar tergolong kecil dan terbatas pada objek berbentuk bola berwarna kontras (merah dan putih) dengan latar belakang yang relatif seragam. Metrik evaluasi yang mendekati sempurna (*Recall* 1,000, *mAP@50* 0,9949) sangat mungkin dipengaruhi oleh keterbatasan variasi visual dalam dataset tersebut, sehingga terdapat risiko *overfitting* terhadap karakteristik objek uji yang spesifik. Performa model berpotensi menurun secara signifikan apabila diuji pada target yang lebih beragam, kondisi pencahayaan dinamis, latar belakang kompleks, atau target yang bergerak dengan kecepatan tinggi. Oleh karena itu, penelitian mendatang disarankan untuk memperluas dataset dengan cakupan objek dan kondisi yang lebih variatif, serta melakukan pengujian pada skenario target militer dinamis di lingkungan nyata guna mengevaluasi generalisasi sistem secara menyeluruh.

F. Keterbatasan dan Tantangan

Sistem masih memiliki beberapa keterbatasan, yaitu penurunan akurasi pada pencahayaan rendah atau *occlusion*, keterbatasan pelacakan akibat interval servo 0,55 detik, potensi deviasi karena getaran mekanis, serta belum diuji pada skenario multiobjek dinamis. Dari sisi komputasi, keterbatasan perangkat keras membatasi laju akuisisi kamera pada 30 FPS dengan kecepatan inferensi rata-rata 28,4 FPS pada resolusi 640×480 (CPU-only). Meskipun integrasi sistem sudah stabil, penggunaan GPU atau *edge* AI *accelerator* berpotensi meningkatkan kecepatan, akurasi, dan kemampuan pelacakan multi-objek.

IV. Kesimpulan dan saran

Penelitian ini berhasil mengembangkan prototipe sistem penargetan otomatis berbasis AI yang mengintegrasikan YOLOv8, ESP32, kamera, dan antarmuka web Flask sebagai *proof-of-concept* dalam lingkungan laboratorium terisolasi, sehingga belum merepresentasikan implementasi operasional di dunia nyata. Bola berwarna kontras digunakan sebagai *proxy* target uji awal karena menyediakan kondisi ideal dan terkendali untuk memvalidasi arsitektur sistem kontrol dan pipeline deteksi sebelum diarahkan ke target domain yang lebih kompleks. Kontribusi utama penelitian terletak pada integrasi *end-to-end* antara YOLOv8 dan algoritma P-Control pada PC *host* dengan ESP32 sebagai unit eksekutor mekanis, dalam satu sistem yang mendukung dual mode (manual dan otomatis), kontrol perintah suara, serta fitur unggah model YOLO kustom (.pt) tanpa modifikasi sistem inti. Sistem menunjukkan kinerja baik pada kondisi uji terkendali, dengan settling time 2,12 detik, positioning error 4,8 piksel, dan overshoot 11,2%. Meskipun masih terbatas pada kondisi *low-light, occlusion,* dan pelacakan objek cepat, hasil ini mengonfirmasi bahwa arsitektur yang diusulkan layak menjadi fondasi awal pengembangan teknologi penargetan otomatis berbasis visi komputer, dengan peluang perluasan ke domain aplikasi lain.

Daftar Pustaka

[1] M. A. M. Alhassan dan E. Yılmaz, “Evaluating YOLOv4 and YOLOv5 for Enhanced Object Detection in UAV-Based Surveillance,” *Processes*, vol. 13, no. 1, hlm. 254, Jan 2025, doi: 10.3390/pr13010254.

[2] N. Zhu, F. Zhong, X. Lei, G. Niu, H. Xie, dan Y. Zhang, “Situation Awareness and Tracking Algorithm for Countering Low-Altitude Swarm Target Threats,” *Remote Sens. (Basel).*, vol. 17, no. 7, hlm. 1172, Mar 2025, doi: 10.3390/rs17071172.

[3] L. Kong, J. Wang, dan P. Zhao, “YOLO-G: A Lightweight Network Model for Improving the Performance of Military Targets Detection,” *IEEE Access*, vol. 10, hlm. 55546–55564, 2022, doi: 10.1109/ACCESS.2022.3177628.

[4] G. Alotaibi, M. Awawdeh, F. F. Farook, M. Aljohani, R. M. Aldhafiri, dan M. Aldhoayan, “Artificial intelligence (AI) diagnostic tools: utilizing a convolutional neural network (CNN) to assess periodontal bone level radiographically—a retrospective study,” *BMC Oral Health*, vol. 22, no. 1, hlm. 399, Sep 2022, doi: 10.1186/s12903-022-02436-3.

[5] J. Hsueh dan C.-T. Yang, “Using a High-Precision YOLO Surveillance System for Gun Detection to Prevent Mass Shootings,” *AI*, vol. 6, no. 9, hlm. 198, Agu 2025, doi: 10.3390/ai6090198.

[6] A. Vijayakumar dan S. Vairavasundaram, “YOLO-based Object Detection Models: A Review and its Applications,” *Multimed. Tools Appl.*, vol. 83, no. 35, hlm. 83535–83574, Mar 2024, doi: 10.1007/s11042-024-18872-y.

[7] P. Mittal, "A comprehensive survey of deep learning-based lightweight object detection models for edge devices," *Artif. Intell. Rev.*, vol. 57, no. 9, hlm. 242, Agu 2024, doi: 10.1007/s10462-024-10877-1.

[8] Y. Sun *dkk.*, "YOLO-E: a lightweight object detection algorithm for military targets," *Signal Image Video Process.*, vol. 19, no. 3, hlm. 241, Mar 2025, doi: 10.1007/s11760-024-03808-8.

[9] S. P dan M. V, "Weapon detection with FMR-CNN and YOLOv8 for enhanced crime prevention and security," *Sci. Rep.*, vol. 15, no. 1, hlm. 26766, Jul 2025, doi: 10.1038/s41598-025-07782-0.

[10] X. Du, L. Song, Y. Lv, dan S. Qiu, "A Lightweight Military Target Detection Algorithm Based on Improved YOLOv5," *Electronics (Basel).*, vol. 11, no. 20, hlm. 3263, Okt 2022, doi: 10.3390/electronics11203263.

[11] Y.-H. Chang, F.-C. Wu, dan H.-W. Lin, "Design and Implementation of ESP32-Based Edge Computing for Object Detection," *Sensors*, vol. 25, no. 6, hlm. 1656, Mar 2025, doi: 10.3390/s25061656.

[12] A. Aniobi, "Sensor Fusion for Real-Time Object Detection and Spatial Positioning in Unmanned Vehicles Using YOLOv8 and ESP32-Cam," *Preprints (Basel).*, Nov 2024, doi: 10.20944/preprints202411.0611.v1.

[13] M. Guerbaoui *dkk.*, "From Data to Decisions: A Smart IoT and Cloud Approach to Environmental Monitoring," *E3S Web of Conferences*, vol. 601, hlm. 00008, Jan 2025, doi: 10.1051/e3sconf/202560100008.

[14] H. Al-Safi, H. Ibrahim, dan P. Steenson, "Vega: LLM-Driven Intelligent Chatbot Platform for Internet of Things Control and Development," *Sensors*, vol. 25, no. 12, hlm. 3809, Jun 2025, doi: 10.3390/s25123809.

[15] D. Cahyono, A. Habibi, A. A. Fairuziya, dan W. Caesarendra, "Design and Implementation of a PID Controller for a Two-Axis Gimbal System Using ESP32," *International Journal of Artificial Intelligence & Robotics (IJAIR)*, vol. 7, no. 2, hlm. 72–82, Nov 2025, doi: 10.25139/ijair.v7i2.10771.

[16] A. Aljaafreh *dkk.*, "A Real-Time Olive Fruit Detection for Harvesting Robot Based on YOLO Algorithms," *Acta Technologica Agriculturae*, vol. 26, no. 3, hlm. 121–132, Sep 2023, doi: 10.2478/ata-2023-0017.

[17] P. Chotikunnan *dkk.*, "Enhancing MG996R Servo Motor Performance Using PSO-Tuned PID and Feedforward Control," *International Journal of Robotics and Control Systems*, vol. 5, no. 2, hlm. 1120–1138, Apr 2025, doi: 10.31763/ijrcs.v5i2.1854.

[18] H. Wang dan Y. Shi, "Design of UAV target tracking controller based on visual servo," *J. Phys. Conf. Ser.*, vol. 2246, no. 1, hlm. 012051, Apr 2022, doi: 10.1088/1742-6596/2246/1/012051.

[19] M. Labeni, C. Boufenar, dan M. Taffar, "Visual Tracking With Object Center Displacement and CenterNet," *International Journal of Computer Vision and Image Processing*, vol. 12, no. 1, hlm. 1–17, Nov 2021, doi: 10.4018/IJCVIP.290397.

[20] Z. Li, "Review of PID control design and tuning methods," *J. Phys. Conf. Ser.*, vol. 2649, no. 1, hlm. 012009, Nov 2023, doi: 10.1088/1742-6596/2649/1/012009.

[21] M. Z. B. A. Karim dan N. M. Thamrin, "Servo Motor Controller using PID and Graphical User Interface on Raspberry Pi for Robotic Arm," *J. Phys. Conf. Ser.*, vol. 2319, no. 1, hlm. 012015, Agu 2022, doi: 10.1088/1742-6596/2319/1/012015.

[22] X. Hao *dkk.*, "Deep reinforcement learning enhanced PID control for hydraulic servo systems in injection molding machines," *Sci. Rep.*, vol. 15, no. 1, hlm. 23005, Jul 2025, doi: 10.1038/s41598-025-05904-2.

[23] W. Gao dan H. Cui, "Composite control of anti-drone platform for stable tracking under disturbance," *Review of Scientific Instruments*, vol. 94, no. 9, Sep 2023, doi: 10.1063/5.0147699.

[24] Q. Liu *dkk.*, "Online multi-object tracking with unsupervised re-identification learning and occlusion estimation," *Neurocomputing*, vol. 483, hlm. 333–347, Apr 2022, doi: 10.1016/j.neucom.2022.01.008.

[25] R.-T. Hong, "Design and Implementation of Environmental Monitoring System Using Flask-Based Web Application," dalam *2024 IEEE 6th Eurasia Conference on IoT, Communication and Engineering*, Basel Switzerland: MDPI, Apr 2025, hlm. 37. doi: 10.3390/engproc2025092037.