

**IMPLEMENTASI EXERTION INTERFACE DALAM GIM
LEMPAR BOLA SEBAGAI PROOF OF CONCEPT
BERBASIS COMPUTER VISION YOLOV8**



Disusun Oleh:

N a m a : Agung Fitrah Ramadhana Ukfah

NIM : 18523178

**PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA**

2025

HALAMAN PENGESAHAN DOSEN PEMBIMBING

**IMPLEMENTASI EXERTION INTERFACE DALAM GIM
LEMPAR BOLA SEBAGAI PROOF OF CONCEPT
BERBASIS COMPUTER VISION YOLOV8**



الجامعة الإسلامية
الاندونيسية

Yogyakarta, 13 Januari 2025

Pembimbing,


(Arrie Kurnawardhani, S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

**IMPLEMENTASI EXERTION INTERFACE DALAM GIM
LEMPAR BOLA SEBAGAI PROOF OF CONCEPT
BERBASIS COMPUTER VISION YOLOV8**

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 13 Januari 2025

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom.



Anggota 1

Kholid Haryono, S.T., M.Kom.



Anggota 2

Rahadian Kurniawan, S.Kom., M.Kom.

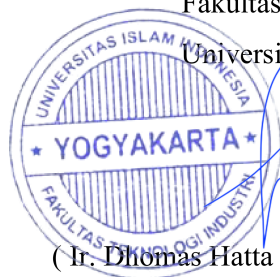
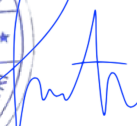


Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia



(Ir. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Agung Fitrah Ramadhana Ukfah

NIM : 18523178

Tugas akhir dengan judul:

**IMPLEMENTASI EXERTION INTERFACE DALAM GIM
LEMPAR BOLA SEBAGAI PROOF OF CONCEPT
BERBASIS COMPUTER VISION YOLOV8**

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 13 Januari 2025



(Agung Fitrah Ramadhana Ukfah)

HALAMAN PERSEMBAHAN

Alhamdulillah rabbil ‘alamin, puji syukur peneliti panjatkan kepada Allah SWT yang telah memberikan nikmat kesehatan, kemudahan, dan keberkahan selama proses penyusunan hingga penyelesaian tugas akhir ini. Dengan terselesaikannya tugas akhir ini, saya berharap hasilnya dapat memberi manfaat bagi diri saya sendiri dan bagi pembaca.

Halaman ini dengan penuh cinta saya persembahkan kepada:

1. Kedua Orang Tua Tercinta, Bapak Marzuki Shahab dan Ibu Siti Maifah yang senantiasa memberikan kasih sayang, doa, dan dukungan yang tak pernah putus. Kalian selalu percaya pada setiap langkah yang penulis lalui, dan memberikan dukungan tanpa ragu. Segala jerih payah, pengorbanan, serta kesabaran kalian dalam mendidik dan membimbing penulis menjadi kekuatan terbesar dalam perjalanan hidup dan pendidikan penulis. Terima kasih atas cinta dan perhatian yang tidak pernah surut, serta doa yang senantiasa dipanjatkan untuk keberhasilan penulis.
2. Kakak-kakak Tersayang, Nining, Nanang dan Devi, terima kasih atas motivasi dan dukungan yang telah menjadi fondasi bagi penulis. Kalian memberikan rasa aman, keberanian, dan semangat untuk terus melangkah maju. Dukungan kalian menjadi sumber kekuatan, dan keteladanan kalian menjadikan kalian sebagai role model dalam perjalanan hidup penulis.
3. Dosen Pembimbing, Ibu Arrie Kurniawardhani, yang telah meluangkan waktu, tenaga, dan pikirannya untuk membimbing serta mengarahkan saya selama proses penelitian dan penulisan laporan tugas akhir ini. Semoga kebaikan yang telah diberikan senantiasa mendapatkan balasan dari Allah SWT. Terima kasih atas segala masukan dan arahan yang sangat berarti dalam penyelesaian laporan ini.
4. Ayuningtyas Maarif, yang selalu menemani penulis di masa-masa sulit. Kepercayaan yang diberikan kepada penulis telah menjadi dorongan besar untuk terus maju. Selalu hadir di saat penulis sangat membutuhkan, serta keteguhan yang tak pernah goyah, menjadi sumber semangat bagi penulis untuk terus melangkah ke depan.
5. Teman Seperjuangan, yang selalu hadir memberi semangat, dukungan, dan kebersamaan selama masa perkuliahan dan proses pengerjaan laporan tugas akhir ini. Kehadiran kalian memberikan warna tersendiri dalam perjalanan ini. Terima kasih atas tawa, cerita, dan dukungan moral yang selalu menguatkan saya.

Semoga tugas akhir ini menjadi amal jariyah dan kontribusi yang bermanfaat bagi ilmu pengetahuan dan kehidupan. Aamiin.

HALAMAN MOTO

“Sesungguhnya, bersama kesulitan ada kemudahan”

— QS. Al-Insyirah (94:6)

“Dan sungguh, Kami akan menguji kamu dengan sedikit ketakutan, kelaparan, kekurangan harta, jiwa, dan buah-buahan. Dan sampaikanlah kabar gembira kepada orang-orang yang sabar”

— QS. Al-Baqarah (2:155)

“Present fears are less than horrible imaginings”

— William Shakespeare

“When nothing seems to help, I go and look at a stonecutter hammering away at his rock. He may hit it a hundred times without as much as a crack showing. Yet at the hundred-and-first blow, it will split in two”

— James Clear

“Courage doesn’t always roar. Sometimes courage is the quiet voice at the end of the day saying, ‘I will try again tomorrow’”

— Mary Anne Radmacher

“It always seems impossible until it’s done”

— Nelson Mandela

“Stop beating yourself up. You are a work in progress, which means you get there a little at a time, not all at once”

— Unknown

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakaatuh,

Alhamdulillah, segala puji dan syukur senantiasa penulis panjatkan ke hadirat Allah SWT, Sang Maha Pemilik ilmu pengetahuan, atas limpahan rahmat, hidayah, dan karunia-Nya yang tiada terhingga sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik. Tugas akhir ini yang berjudul *“Implementasi Exertion Interface Dalam Gim Lempar Bola Sebagai Proof Of Concept Berbasis Computer Vision YOLOv8”* merupakan salah satu syarat untuk menyelesaikan studi pada Program Studi Informatika Fakultas Teknologi Industri Universitas Islam Indonesia. Proses panjang penyelesaian laporan ini tidak hanya dipenuhi dengan berbagai tantangan, tetapi juga memberikan banyak pelajaran, pengalaman berharga, dan kesempatan untuk mengembangkan kemampuan berpikir ilmiah serta kreativitas.

Penyelesaian tugas akhir ini tidak lepas dari dukungan, bimbingan, serta bantuan berbagai pihak yang telah memberikan arahan dan motivasi kepada penulis. Oleh karena itu, pada kesempatan ini, penulis ingin menyampaikan rasa terima kasih dan penghargaan setinggi-tingginya kepada:

1. **Allah SWT**, atas segala limpahan nikmat dan pertolongan-Nya dalam setiap langkah penulis, yang telah memberikan kekuatan, kesehatan, dan kelancaran dalam penyusunan tugas akhir ini.
2. **Kedua orang tua tercinta dan kakak-kakak tersayang**, atas doa, cinta, dukungan moral dan material, serta motivasi yang tidak pernah henti.
3. **Prof. Dr. Ir. Hari Purnomo, M.T., IPU, ASEAN.Eng**, selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
4. **Dr. Ir. Raden Teduh Dirgahayu, S.T., M.Sc.**, selaku Ketua Jurusan Informatika - Fakultas Teknologi Industri Universitas Islam Indonesia.
5. **Ir. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.**, selaku Ketua Program Studi Informatika – Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia.
6. **Ibu Arrie Kurniawardhani, S.Si., M.Kom.**, selaku dosen pembimbing, yang telah dengan sabar membimbing, memberikan waktu, ilmu, arahan, dan masukan berharga selama proses penelitian hingga penulisan laporan ini.
7. **Bapak Andhik Budi Cahyono, S.T., M.T.**, selaku dosen pembimbing akademik, yang telah memberikan arahan, bimbingan, dan dukungan selama masa studi penulis.

8. **Seluruh dosen di Program Studi Informatika Fakultas Teknologi Industri Universitas Islam Indonesia**, yang telah memberikan bekal ilmu pengetahuan, wawasan, dan motivasi selama masa perkuliahan.
9. **Ayuningtyas Maarif**, yang selalu menemani penulis pada masa-masa sulit, memberikan dukungan emosional dan semangat untuk terus melangkah.
10. **Yudha Nugroho**, seorang teman yang telah membantu penulis menjadi lebih berani dalam menghadapi ketakutan lama.
11. **Teman-teman seperjuangan di Discord AVOCADO**, yang telah menjadi tempat berbagi cerita dan memberikan dukungan luar biasa selama masa-masa sulit.
12. **Rekan-rekan seperjuangan di Program Studi Informatika angkatan 2018**, yang telah menjadi teman diskusi, berbagi cerita, dan memberikan semangat selama masa kuliah hingga penyusunan tugas akhir ini.
13. **Sahabat-sahabat terbaik**, yang selalu hadir memberikan dukungan moral, semangat, dan kebersamaan dalam perjalanan akademik ini.
14. **Seluruh staf administrasi dan tenaga pendukung Fakultas Teknologi Industri Universitas Islam Indonesia**, yang selalu siap membantu dalam hal teknis dan administrasi selama masa perkuliahan.
15. **Responden dan pihak yang terlibat dalam pengujian Gim ini**, atas kesediaan waktunya untuk membantu menyempurnakan hasil penelitian ini.
16. **Seluruh pihak yang tidak dapat disebutkan satu per satu**, yang telah membantu baik secara langsung maupun tidak langsung, dalam proses penyelesaian tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih jauh dari kata sempurna. Oleh karena itu, saran dan kritik yang membangun sangat diharapkan untuk menyempurnakan karya ini. Semoga tugas akhir ini dapat memberikan manfaat bagi perkembangan ilmu pengetahuan, terutama di bidang informatika, serta menjadi inspirasi bagi para pembaca.

Wassalamu'alaikum Warahmatullahi Wabarakaatuh.

Yogyakarta, 13 Januari 2025



(Agung Fitrah Ramadhana Ukfah)

SARI

Tingkat aktivitas fisik yang rendah akibat tingginya penggunaan perangkat elektronik telah memunculkan kebutuhan akan solusi inovatif untuk mendorong gaya hidup aktif bagi semua kalangan. Penelitian ini bertujuan mengatasi tantangan tersebut dengan mengembangkan kontrol interaksi fisik yang terjangkau, mudah diakses, dan tetap akurat sebagai langkah awal. Dengan mengintegrasikan teknologi Computer Vision dan Exertion Interface dalam gim lempar bola sebagai *Proof of Concept* dengan menawarkan solusi yang lebih terjangkau dan mudah diakses. Permainan dilakukan dengan melempar bola fisik ke arah target digital pada layar, sementara sistem berbasis YOLOv8 mendeteksi lemparan dan akurasi secara *real-time*. Hasil pengujian menunjukkan deteksi objek yang andal dengan mAP50-95 sebesar 93% dalam kondisi terkontrol. Survei pengguna menunjukkan skor rata-rata 8.8 dari 10 untuk kesenangan dan interaktivitas, serta 7.6 dari 10 untuk dorongan aktivitas fisik. Teknologi ini terbukti memberikan pengalaman bermain yang menyenangkan sekaligus efektif dalam mempromosikan gaya hidup aktif. Hal ini juga dirancang untuk dapat diimplementasikan di berbagai skenario seperti pendidikan, hiburan keluarga, dan kegiatan komunitas. Dengan pendekatan berbasis teknologi modern, solusi ini memperlihatkan potensi besar dalam mendorong keterlibatan fisik yang lebih baik.

Kata kunci: interaksi fisik, *Proof of Concept*, Computer Vision, YOLOv8, deteksi *real-time*, teknologi modern, gaya hidup aktif.

GLOSARIUM

<i>Aimbot</i>	Sebuah cheat dalam permainan yang secara otomatis mengarahkan senjata ke musuh, memberikan keuntungan yang tidak adil dan melanggar aturan permainan.
AI	AI (<i>Artificial Intelligence</i>) adalah teknologi yang memungkinkan mesin meniru kecerdasan manusia, seperti belajar, berpikir, memahami bahasa, dan membuat keputusan.
AR	AR (<i>Augmented Reality</i>) adalah teknologi yang menggabungkan elemen digital (seperti gambar, teks, atau objek 3D) ke dalam dunia nyata secara real-time.
Exertion Interface	Desain antarmuka pengguna yang sengaja memerlukan upaya fisik yang signifikan dari pengguna untuk menciptakan pengalaman yang lebih <i>imersif</i> .
FPS	FPS (Frames Per Second) adalah ukuran seberapa banyak gambar (frame) yang dapat dirender oleh sistem per detik. FPS yang lebih tinggi umumnya menunjukkan gerakan yang lebih mulus dalam video atau permainan.
YOLO	YOLO (You Only Look Once) adalah algoritma deteksi objek real-time yang memproses gambar dalam satu kali proses, menawarkan kecepatan dan efisiensi tinggi, sangat penting untuk aplikasi yang memerlukan respon cepat
<i>Ultra-short-throw</i>	Proyektor yang mampu memproyeksikan gambar besar dari jarak kurang dari 0.5 meter. Tidak menimbulkan bayangan, ideal untuk ruangan kecil, papan tulis interaktif, dan home theater
<i>Short-throw</i>	Proyektor yang dapat menampilkan gambar besar dari jarak pendek, sekitar 0.5 hingga 1.5 meter dari layar. Mengurangi efek bayangan dan cocok untuk kelas, kantor
<i>Standard-throw</i>	Proyektor konvensional yang membutuhkan jarak lebih dari 2 meter untuk tampilan besar. Digunakan di ruang konferensi, bioskop, dan auditorium, tetapi rentan terhadap bayangan jika ada objek di depan cahaya proyeksi
<i>Proof of Concept</i>	POC adalah prototipe untuk membuktikan kelayakan atau potensi suatu ide atau teknologi sebelum pengembangan lebih lanjut

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGESAHAN DOSEN PEMBIMBING.....	ii
HALAMAN PENGESAHAN DOSEN PENGUJI.....	iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR	iv
HALAMAN PERSEMBAHAN	v
HALAMAN MOTO	vi
KATA PENGANTAR	vii
SARI	ix
GLOSARIUM.....	x
DAFTAR ISI.....	xi
DAFTAR TABEL.....	xiii
DAFTAR GAMBAR.....	xiv
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	3
1.3 Tujuan Penelitian	3
1.4 Manfaat Penelitian	3
1.5 Batasan Masalah	3
1.6 Metodologi Penelitian.....	3
1.6.1 Pengumpulan Data	4
1.6.2 Pelabelan Data.....	4
1.6.3 Pelatihan Model Deteksi Bola.....	4
1.6.4 Pengembangan Algoritma Deteksi <i>Hit</i>	4
1.6.5 Implementasi Algoritma Deteksi Bola dan <i>Hits</i> pada Gim.....	4
1.6.6 Pengujian dan Evaluasi Implementasi.....	4
1.7 Sistematika Penulisan	5
1.7.1 BAB I PENDAHULUAN	5
1.7.2 BAB II TINJAUAN PUSTAKA.....	5
1.7.3 BAB III METODOLOGI PENELITIAN.....	5
1.7.4 BAB IV HASIL DAN PEMBAHASAN.....	5
1.7.5 BAB V KESIMPULAN DAN SARAN.....	5
BAB II TINJAUAN PUSTAKA	6

2.1	Penelitian Terdahulu	6
2.2	Computer Vision.....	10
2.3	Deteksi Objek.....	11
2.4	OpenCV (Open Source Computer Vision Library)	12
2.5	YOLO (You Only Look Once)	13
2.6	<i>Hyperparameter</i> Pelatihan YOLOv8.....	14
2.7	Metrik Evaluasi YOLOv8.....	15
2.8	LabelImg	17
2.9	Exertion Interface.....	18
2.10	Gim Lempar Bola sebagai <i>Proof Of Concept</i>	19
BAB III METODOLOGI PENELITIAN		29
3.1	Pengumpulan Data	29
3.2	Pelabelan Data	30
3.3	Pelatihan Model Deteksi Bola	30
3.4	Pengembangan Algoritma Deteksi <i>Hit</i>	31
3.5	Implementasi Algoritma Deteksi Bola dan <i>Hit</i> pada Gim	32
3.6	Pengujian dan Evaluasi Implementasi	33
BAB IV HASIL DAN PEMBAHASAN		35
4.1	Pengumpulan Data	37
4.2	Pelabelan Data	38
4.3	Pelatihan Model Deteksi Bola	39
4.4	Pengembangan Algoritma Deteksi <i>Hit</i>	44
4.5	Implementasi Algoritma Deteksi Bola dan <i>Hit</i> pada Gim	45
4.5.1	Implementasi Deteksi Bola dan <i>Hit</i>	45
4.5.2	Feedback Real-Time dan Sistem Skor	49
4.6	Pengujian dan Evaluasi Implementasi	51
4.6.1	Pengujian Teknis	51
4.6.2	Respons Pemain dan Evaluasi Gim.....	56
4.7	Diskusi	58
BAB V KESIMPULAN DAN SARAN		60
5.1	Kesimpulan	60
5.2	Saran Pengembangan	61
DAFTAR PUSTAKA		62
LAMPIRAN.....		66

DAFTAR TABEL

Tabel 2.1 Perbandingan FPS pada pengujian berbagai versi YOLO.....	7
Tabel 3.1 <i>System Usability Scale (SUS) Adjective Score</i>	34
Tabel 4.1 Hasil Pengujian YOLOv8 di permainan.....	52
Tabel 4.2 Hasil Pengujian Model YOLOv8	56
Tabel 4.3 Hasil Survei Pemain Skala 10.....	57

DAFTAR GAMBAR

Gambar 2.1 Setup Extortion Interfaces.....	6
Gambar 2.2 Contoh <i>Sit-ups monitoring</i>	7
Gambar 2.3 Contoh <i>gameplay</i> tantangan lokasi untuk dikunjungi langkah kaki dan <i>score</i>	8
Gambar 2.4 Contoh cara menggunakan <i>controller</i> dan tampilan <i>game</i> -nya.	9
Gambar 2.5 Contoh permainan <i>Airhockey over a distance</i> dan teknologi dibalikny.....	10
Gambar 2.6 (a) <i>Hist equalization</i> , (b) <i>Compression</i> , (c) <i>Normalization</i> , (d) <i>Noise reduction</i> ,11	
Gambar 2.7 Contoh Gim lempar bola di dalam kelas.....	20
Gambar 2.8 (a) Kamera untuk di kalibrasi, (b) Kamera setelah di kalibrasi	20
Gambar 2.9 Kode program <i>calibration.py</i>	22
Gambar 2.10 Antarmuka <i>Main Menu</i> dengan inputan radius.....	23
Gambar 2.11 Antarmuka <i>countdown</i>	23
Gambar 2.12 Kode fungsi <i>main menu</i>	25
Gambar 2.13 Antarmuka permainan utama dengan pembagian area permainan	27
Gambar 2.14 Kode fungsi <i>update_targets</i>	27
Gambar 2.15 Kode kelas <i>Target</i>	28
Gambar 3.1 Alur Penelitian.	29
Gambar 3.2 Struktur <i>folder</i> persiapan pelatihan model	30
Gambar 3.3 <i>System Usability Scale (SUS) Score</i>	33
Gambar 4.1 (a) Kamera, (b) proyektor, dan (c) laptop	35
Gambar 4.2 Diagram penempatan perangkat keras	36
Gambar 4.3 (a) Dua video awal simulasi permainan, (b) Video final setelah pemotongan	37
Gambar 4.4 Hasil ekstraksi gambar (a) 1368 gambar (b) 1136 gambar	37
Gambar 4.5 Kode ekstraksi gambar dari tiap <i>frame</i> video	38
Gambar 4.6 Isi file <i>text.txt</i> hasil pelabelan	39
Gambar 4.7 Hasil evaluasi <i>train loss</i>	40
Gambar 4.8 Hasil evaluasi <i>validation loss</i>	41
Gambar 4.9 Hasil evaluasi metrik <i>precision-recall</i>	41
Gambar 4.10 Hasil evaluasi metrik <i>F1 Confidence Curve</i>	42
Gambar 4.11 Hasil evaluasi metrik <i>mAP50</i> dan <i>mAP50-95</i>	43
Gambar 4.12 Hasil evaluasi metrik <i>confusion matrix</i>	43
Gambar 4.13 Kode Perhitungan Radius.....	45
Gambar 4.14 Kode Perhitungan Euclidean.....	45

Gambar 4.15 Kode Pengecekan Apakah Target dan Bola Bersentuhan.....	45
Gambar 4.16 Inisiasi model dan kalibrasi kamera.....	46
Gambar 4.17 Kode fungsi main	49
Gambar 4.18 Antarmuka statistik di akhir permainan	50
Gambar 4.19 Perbandingan proyektor <i>ultra-short-throw</i> , <i>short-throw</i> , dan <i>standard-throw</i> ..	55

BAB I PENDAHULUAN

1.1 Latar Belakang

Pada era digital ini, perkembangan teknologi telah memberikan dampak signifikan terhadap berbagai sektor kehidupan, termasuk hiburan dan industri gim. Platform gim yang dulunya hanya berfokus pada pengalaman visual kini semakin berkembang dengan mengintegrasikan elemen-elemen interaktif yang melibatkan interaksi fisik pemain. Seiring kemajuan teknologi, terutama dalam bidang Virtual Reality (VR), Augmented reality (AR), dan Computer Vision (CV), gim-gim kini menawarkan pengalaman yang lebih *imersif* dan dinamis. Salah satu model CV yang paling populer dan banyak digunakan adalah You Only Look Once (YOLO). YOLO dikenal karena kemampuannya mendeteksi objek secara *real-time* dengan kecepatan dan akurasi tinggi, menjadikannya pilihan ideal untuk gim interaktif berbasis gerakan.

Salah satu pendekatan menarik adalah pengimplementasian teknologi Computer Vision dalam bentuk gim yang dapat mendeteksi interaksi fisik dari pemainnya. Florian Mueller, melalui penelitiannya tentang pengembangan gim tendang bola interaktif jarak jauh yang diintegrasikan dengan teknologi Computer Vision (F. Mueller et al., 2003), memperkenalkan konsep Exertion Interface, yaitu antarmuka yang secara sengaja memerlukan usaha fisik signifikan dari pengguna untuk menciptakan pengalaman interaktif yang lebih *imersif*. Konsep ini bertujuan untuk mendorong pemain bergerak lebih aktif melalui permainan yang menggabungkan teknologi Computer Vision, sehingga tidak hanya memberikan hiburan, tetapi juga manfaat kesehatan.

Meskipun gim berbasis interaksi fisik memberikan pengalaman yang lebih dinamis, teknologi yang digunakan pada platform tersebut masih memiliki keterbatasan, terutama dalam hal akurasi dan responsivitas terhadap gerakan tubuh yang lebih kompleks. Di sisi lain, meskipun teknologi terbaru seperti VR dan AR menawarkan pengalaman yang lebih *imersif* dan interaktif, teknologi-teknologi ini sering kali sulit diakses oleh khalayak umum. Hal ini disebabkan oleh tingkat kompleksitas dan harga dari perangkat yang dibutuhkan, yang jauh lebih tinggi dibandingkan dengan platform gim lainnya. Perangkat VR dan AR, misalnya, membutuhkan perangkat keras khusus yang sering kali mahal dan memerlukan ruang fisik yang cukup besar untuk digunakan secara efektif. Akibatnya, meskipun teknologi ini menawarkan

potensi besar, pengalaman tersebut belum sepenuhnya dapat dinikmati oleh sebagian besar pengguna.

Interaksi fisik yang dimaksud dalam konteks ini merujuk pada metode kontrol permainan digital yang memanfaatkan deteksi pergerakan objek di dunia nyata oleh pemain melalui teknologi Computer Vision, sedangkan *imersif* yang dimaksud dalam konteks ini mengacu pada pengalaman bermain yang melibatkan interaksi mendalam dengan lingkungan dan narasi dalam gim. Ciri utama dari gim *imersif* adalah mendorong pemain untuk berinteraksi langsung dengan dunia gim, bergerak secara aktif di dalam lingkungan yang disediakan, serta memainkan peran dalam cerita yang sedang berlangsung (Fullerton, 2024).

Menurut Global Status Report on Physical Activity 2022 (World Health Organization, 2022), lebih dari 1 dari 4 orang dewasa dan 81% remaja usia 11-17 tahun secara global tidak memenuhi tingkat aktivitas fisik yang direkomendasikan WHO untuk kesehatan yang optimal, yakni minimal 1 jam per hari aktivitas fisik dengan intensitas sedang hingga berat. Data ini menunjukkan bahwa ketidakaktifan fisik masih menjadi masalah kesehatan masyarakat yang signifikan secara global. Di Indonesia, berdasarkan laporan Survei Kesehatan Indonesia (SKI) 2023 (BPS, 2018), 58% anak usia 10–14 tahun dan 50,4% remaja usia 15–19 tahun mengalami kurangnya aktivitas fisik. Fenomena ini semakin diperburuk oleh peningkatan penggunaan perangkat elektronik. Laporan State of Mobile 2023 (Nelson, 2023) menunjukkan bahwa rata-rata pengguna di pasar utama, termasuk Indonesia, menghabiskan 5 jam 15 menit per hari untuk hiburan berbasis layar, meningkat dari 4,8 jam per hari pada tahun 2021 (Iswaratioso, 2022). Hal ini mencerminkan ketergantungan yang semakin meningkat terhadap perangkat elektronik untuk aktivitas sehari-hari.

Penelitian ini bertujuan untuk mengatasi tantangan tersebut dalam jangka panjang, dengan fokus pada pengembangan kontrol interaksi fisik yang murah dan mudah diakses tanpa mengurangi akurasi sebagai langkah awal. Teknologi YOLOv8 akan digunakan untuk memungkinkan deteksi interaksi fisik secara *real-time* dengan akurasi tinggi, serta memberikan pengalaman gim berbasis gerakan yang responsif dan terjangkau. Dengan demikian, penelitian ini diharapkan dapat membuka akses lebih luas ke gim berbasis interaksi fisik dan mendorong partisipasi aktif dalam aktivitas fisik. Penelitian ini berjudul "***Implementasi Exertion Interface Dalam Gim Lempar Bola Sebagai Proof of Concept Berbasis Computer Vision YOLOv8***". Tujuan utama penelitian ini adalah mengintegrasikan teknologi Computer Vision dan Exertion Interface dalam gim lempar bola sebagai *proof of concept* dengan menawarkan solusi yang lebih terjangkau dan mudah diakses.

1.2 Rumusan Masalah

1. Bagaimana cara mengimplementasikan teknologi Computer Vision menggunakan YOLOv8 untuk mendeteksi pergerakan bola secara *real-time* dalam gim lempar bola dan menggunakannya sebagai kontrol dalam permainan?
2. Bagaimana performa YOLOv8 dalam mendeteksi objek (bola) secara akurat dan responsif dalam gim lempar bola berbasis gerakan?

1.3 Tujuan Penelitian

1. Mengimplementasikan teknologi Computer Vision menggunakan YOLOv8 untuk mendeteksi pergerakan bola secara *real-time* dalam gim lempar bola dan memanfaatkannya sebagai kontrol dalam permainan untuk menciptakan pengalaman interaktif dan responsif bagi pemain.
2. Mengevaluasi kinerja YOLOv8 dalam mendeteksi pergerakan bola secara *real-time* untuk memastikan keakuratan dan responsivitas dalam gim lempar bola berbasis gerakan.

1.4 Manfaat Penelitian

1. **Teoretis:** Memberikan kontribusi pada pengembangan teknologi Computer Vision dalam aplikasi gim interaksi fisik, khususnya dalam gim lempar bola.
2. **Praktis:** Menyediakan solusi gim lempar bola yang lebih terjangkau dan mudah diakses, meningkatkan kualitas interaksi fisik pemain.
3. **Sosial:** Mendorong partisipasi aktif dalam aktivitas fisik dengan menyediakan gim lempar bola yang menggabungkan hiburan dan manfaat kesehatan, serta mengurangi ketergantungan pada perangkat elektronik pasif.

1.5 Batasan Masalah

1. Penelitian ini hanya berfokus pada implementasi Computer Vision YOLOv8 dan Exertion Interface pada Gim Lempar Bola sebagai *Proof of Concept*.
2. Data yang digunakan untuk pelatihan model deteksi objek berbasis YOLOv8 dikumpulkan dalam lingkungan terkontrol dengan pencahayaan stabil.
3. Penelitian ini tidak mencakup pengujian Gim dalam skala besar di lingkungan nyata seperti sekolah atau ruang publik.
4. Studi ini terbatas pada penggunaan perangkat keras standar, yaitu satu kamera dengan kecepatan minimum 30 FPS dan satu proyektor *standard-throw* 3000+ lumens bukan proyektor *short-throw* ataupun proyektor *ultra-short-throw*.

1.6 Metodologi Penelitian

1.6.1 Pengumpulan Data

Pada tahap ini, dilakukan pengumpulan data berupa gambar atau video yang relevan dengan deteksi bola dalam permainan. Data ini digunakan untuk melatih model deteksi objek yang akan mengidentifikasi bola dalam gambar atau video yang diambil selama permainan. Pengumpulan data ini penting untuk memastikan model yang dibangun dapat mengenali bola secara efektif di berbagai kondisi lingkungan dan situasi permainan.

1.6.2 Pelabelan Data

Setelah data dikumpulkan, langkah berikutnya adalah proses pelabelan data. Pada tahap ini, setiap objek dalam gambar atau video yang berkaitan dengan bola akan diberi label sesuai dengan kategori yang relevan. Pelabelan data ini sangat penting untuk memastikan model deteksi dapat belajar mengenali objek yang dimaksud, yaitu bola, secara akurat.

1.6.3 Pelatihan Model Deteksi Bola

Pelatihan model deteksi bola dilakukan dengan menggunakan data yang telah dilabeli sebelumnya. Pada tahap ini, algoritma YOLOv8 (You Only Look Once version 8) akan dilatih untuk mengenali bola dalam gambar dengan menggunakan metode pembelajaran mesin. Model ini akan dipelajari untuk mendeteksi objek bola secara *real-time* berdasarkan data yang telah diproses, mengoptimalkan parameter model untuk mencapai akurasi deteksi yang tinggi.

1.6.4 Pengembangan Algoritma Deteksi Hit

Pada tahap ini, algoritma dikembangkan untuk mendeteksi apakah bola mengenai target yang ada dalam permainan, yang disebut sebagai deteksi *hit*. Sistem ini akan memproses *input* dari kamera untuk memverifikasi apakah bola berada di dalam area target yang telah ditentukan, serta memastikan penilaian yang akurat terhadap interaksi antara bola dan target. Pengembangan algoritma ini memastikan bahwa sistem dapat memberikan umpan balik langsung kepada pemain.

1.6.5 Implementasi Algoritma Deteksi Bola dan Hits pada Gim

Setelah algoritma deteksi bola dan *hit* berhasil dikembangkan, tahap selanjutnya adalah implementasinya dalam gim. Pada tahap ini, algoritma yang telah dilatih akan diterapkan dalam konteks gim yang dirancang, dengan tujuan untuk memantau pergerakan bola dan interaksi pemain dengan objek dalam gim. Implementasi ini mencakup integrasi algoritma dengan sistem permainan untuk menciptakan pengalaman bermain yang responsif dan interaktif.

1.6.6 Pengujian dan Evaluasi Implementasi

Setelah implementasi algoritma selesai, tahap selanjutnya adalah pengujian dan evaluasi. Pengujian dilakukan untuk memastikan bahwa algoritma dapat mendeteksi bola dan *hit* dengan

akurasi tinggi dalam berbagai kondisi permainan. Evaluasi dilakukan untuk menilai efektivitas sistem dalam memberikan umpan balik *real-time* dan pengalaman bermain yang menyenangkan. Hasil pengujian ini digunakan untuk memperbaiki dan mengoptimalkan algoritma lebih lanjut.

1.7 Sistematika Penulisan

1.7.1 BAB I PENDAHULUAN

Pada bab ini, akan dijelaskan latar belakang penelitian, rumusan masalah, tujuan, manfaat, serta ruang lingkup penelitian. Bab ini memberikan gambaran umum mengenai topik penelitian yang dibahas, pentingnya penelitian ini, serta fokus utama yang akan dijelajahi dalam penelitian.

1.7.2 BAB II TINJAUAN PUSTAKA

Bab ini mengulas teori-teori, konsep, dan penelitian sebelumnya yang relevan dengan topik penelitian. Penulis akan mengidentifikasi kekurangan dan kesenjangan dalam penelitian terdahulu yang akan dibahas dalam penelitian ini. Tujuan dari bab ini adalah untuk membangun landasan teoritis dan memperjelas posisi penelitian dalam konteks literatur yang ada.

1.7.3 BAB III METODOLOGI PENELITIAN

Bab ini menjelaskan tahapan-tahapan yang dilakukan dalam penelitian, mulai dari cara pengumpulan data, teknik yang digunakan untuk analisis, serta prosedur yang diterapkan dalam eksperimen atau implementasi teknologi yang diteliti. Tahapan-tahapan yang dilakukan dalam penelitian dijelaskan secara sistematis untuk memberikan gambaran bagaimana penelitian ini akan dilaksanakan.

1.7.4 BAB IV HASIL DAN PEMBAHASAN

Bab ini menyajikan hasil yang diperoleh dari penerapan metodologi penelitian, dilanjutkan dengan pembahasan mengenai temuan-temuan yang ada. Penulis menganalisis hasil penelitian, membandingkan dengan teori atau penelitian sebelumnya, serta menyajikan interpretasi dari data yang diperoleh.

1.7.5 BAB V KESIMPULAN DAN SARAN

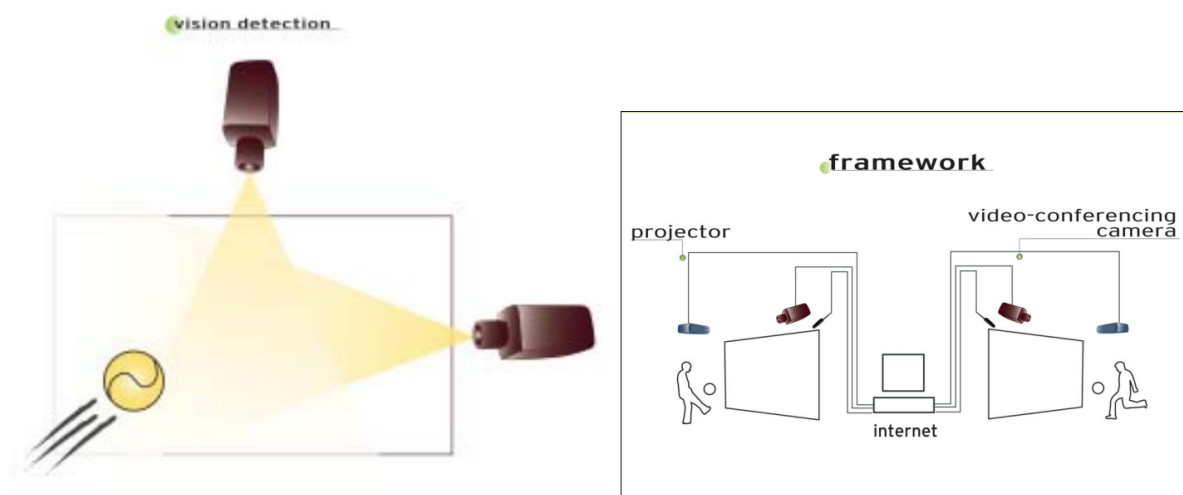
Bab ini berisi kesimpulan dari hasil penelitian yang diperoleh, dengan menyarankan langkah-langkah tindak lanjut atau penelitian lebih lanjut jika diperlukan. Penulis memberikan rekomendasi berdasarkan temuan-temuan yang ada serta saran untuk pengembangan lebih lanjut terkait topik penelitian.

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian tentang Gim interaktif berbasis Computer Vision telah berkembang secara signifikan dalam beberapa tahun terakhir. (F. Mueller et al., 2003) mengembangkan Exertion Interface, sebuah Gim interaktif yang memungkinkan dua pemain bermain sepak bola dari lokasi yang berbeda. Sistem ini menggunakan beberapa kamera, satu untuk menampilkan pemain pada layar lawan, lalu 2 digunakan dengan perpaduan satu *microphone* untuk mendeteksi tumbukan bola dengan target seperti pada Gambar 2.1. Meskipun inovatif, sistem tersebut membutuhkan setup *hardware* yang kompleks dan terbatas pada interaksi dua pemain jarak jauh.

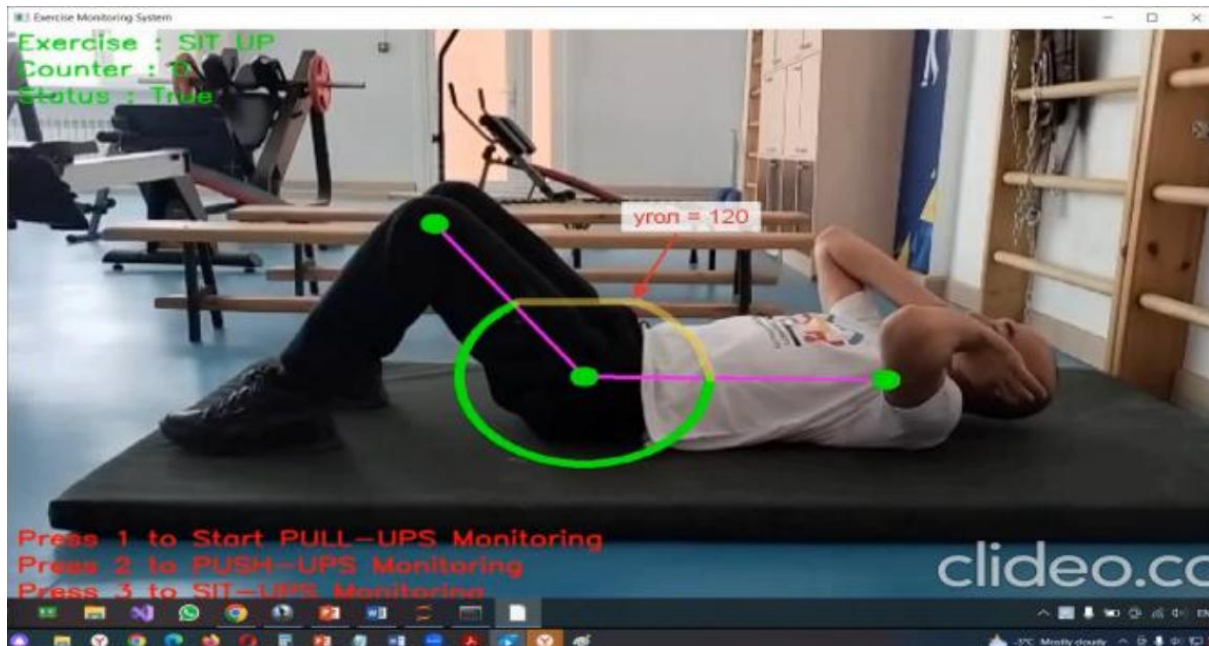


Gambar 2.1 Setup Exertion Interfaces.

Sumber: (F. Mueller et al., 2003)

Pada penelitian (Doskarayev et al., 2023) juga menunjukkan bahwa pengembangan Gim *Artificial Intelligence (AI)-enabled Augmented Reality (AR)* berbasis deteksi aksi dengan teknologi Computer Vision dapat meningkatkan motivasi olahraga. Teknologi ini digunakan untuk mendeteksi gerakan pemain dan memberikan umpan balik secara *real-time* untuk mendukung performa dan motivasi dalam beraktivitas fisik. Contohnya pada *monitoring* salah satu bentuk aktivitas fisik yaitu *sit-ups* dapat dilihat pada Gambar 2.2. Hasilnya pada kelompok

uji dilaporkan merasa lebih termotivasi untuk berpartisipasi dalam aktivitas fisik dan memiliki minat yang lebih besar terhadap olahraga saat menggunakan teknologi tersebut.



Gambar 2.2 Contoh *Sit-ups monitoring*

Sumber: (Doskarayev et al., 2023)

(Asmara et al., 2024) mengevaluasi kinerja YOLO sebagai algoritma deteksi objek untuk aplikasi *aimbot* dalam *First Person Shooter (FPS) game*. Penelitian ini membandingkan berbagai versi dari YOLO, dari YOLOv3 Tiny, YOLOv3 320x320, YOLOv4 Tiny, YOLOv4 320x320, YOLOv5s, dan YOLOv6s pada performa deteksi objek *real-time* nya. Walau YOLOv8 tidak dilakukan pengujian dikarenakan YOLOv8 baru saja diluncurkan pada saat studi tersebut diselesaikan. Namun, dari hasil yang didapatkan pada tiap perkembangan versi YOLO didapatkan perbedaan hasil yang tinggi, bukan hanya dari *Frame Per Second (FPS)* yang dapat dilihat pada Tabel 2.1, tetapi juga pada kemampuan deteksi *real-time* nya. Hasil tersebut membuat peneliti yakin bahwa dengan versi YOLO yang lebih baru, biaya komputasi akan lebih rendah, dan akurasi deteksi akan lebih tinggi. Hal ini sangat penting karena bukan hanya dari FPS yang dihasilkan akan lebih tinggi, tetapi akurasi deteksinya tidak akan dikorbankan bahkan jika dilakukan secara *real-time*, sehingga dalam konteks Gim interaktif hal ini dapat meningkatkan pengalaman pemain.

Tabel 2.1 Perbandingan FPS pada pengujian berbagai versi YOLO

Algorithm	CPU	GPU (single precision)	GPU (half precision)
YOLOv3 tiny	6.5 FPS	15.4 FPS	32 FPS
YOLOv3 320x320	8.5 FPS	12.6 FPS	18.13 FPS
YOLOv4 tiny	6.5 FPS	15.2 FPS	35 FPS
YOLOv4 320x320	8.5 FPS	12.4 FPS	18.32 FPS
YOLOv5s	10 FPS	29 FPS	34 FPS
YOLOv6s	12 FPS	30 FPS	36.5 FPS

Sumber: (Asmara et al., 2024)

(Woodward et al., 2024) membuktikan bahwa Gim berbasis *wearable proximity* tidak hanya dapat memantau aktivitas fisik secara *real-time* tetapi juga mampu meningkatkan partisipasi pengguna dalam aktivitas fisik melalui gamifikasi tantangan berbasis lokasi dengan *gameplay*-nya dapat dilihat pada Gambar 2.3. Hasil dari studi ini menekankan bahwa teknik gamifikasi seperti poin, insentif, dan tantangan berbasis lokasi secara efektif memotivasi aktivitas fisik selama pengujian dan memiliki potensi untuk mendorong perubahan perilaku kesehatan positif.



Gambar 2.3 Contoh *gameplay* memperlihatkan beberapa tantangan lokasi untuk dikunjungi dengan jumlah langkah kaki dan *score*.

Sumber: (Woodward et al., 2024)

Sementara (Giannios & Margounakis, 2022) menyoroti potensi *Mixed Reality*, di mana *controller multisensor* memungkinkan interaksi fisik-virtual melalui aksi seperti mengetuk atau memiringkan perangkat, menciptakan pengalaman bermain yang lebih *imersif* dan responsif. Contoh cara menggunakan *controller* dan tampilan Gim dapat dilihat pada Gambar 2.4. Pendekatan ini berhasil meningkatkan keterlibatan pemain melalui umpan balik langsung yang mendalam.



Gambar 2.4 Contoh cara menggunakan *controller* dan tampilan *game*-nya.

Sumber: (Giannios & Margounakis, 2022)

Pada studi lainnya (F. F. Mueller et al., 2006) mengembangkan kelanjutan dari Exertion Interfaces yang ia kembangkan sebelumnya. Ia mengembangkan sistem yang memungkinkan dua pemain di lokasi berbeda untuk bermain *airhockey* fisik melalui meja yang terhubung secara digital dengan dukungan video konferensi, menciptakan pengalaman sosial yang *imersif* dan interaktif. Contoh permainan dan teknologi yang digunakannya dapat dilihat pada Gambar 2.5. Pendekatan yang sama dilakukan dengan basis Gim yang berbeda dan membutuhkan lebih banyak teknologi *hardware* yang harus di *custom*, seperti meja khusus yang menampilkan Gim *airhockey* yang dapat berinteraksi dengan pemain di dunia nyata, serta layar permainan dan proyektor untuk menampilkan daerah lawan. Walau hal ini adalah perkembangan dari penelitian sebelumnya, tetapi penelitian ini lebih sulit untuk di replikasi karena keterbatasan *hardware* nya. Itulah sebabnya penulis akan menggunakan (F. Mueller et al., 2003) sebagai bahan referensi utama karena kesederhanaannya dan lebih mudah untuk replikasi.



Gambar 2.5 Contoh permainan *Airhockey over a distance* dan teknologi dibalikinya.

Sumber: (F. F. Mueller et al., 2006)

2.2 Computer Vision

Computer Vision adalah bidang ilmu yang berfokus pada pemahaman dan pemrosesan informasi visual oleh komputer, yang semakin relevan dalam pengembangan Gim interaktif. Teknologi ini memungkinkan sistem untuk mentransformasi data visual dari dunia nyata menjadi informasi yang dapat diolah untuk menciptakan pengalaman bermain yang lebih mendalam dan interaktif. Dalam prosesnya, Computer Vision melibatkan serangkaian langkah, mulai dari *image acquisition* untuk menangkap data visual dapat berupa video atau kumpulan gambar, *preprocessing* selain untuk membersihkan dataset juga dapat digunakan untuk meningkatkan kualitas data yang dapat dilihat pada Gambar 2.6 melalui teknik seperti *noise reduction*, *color space conversion*, *histogram equalization*, *compression*, *normalization*, dan *image enhancement*, hingga tahap *feature extraction* dan *interpretation* untuk memahami pola atau objek dalam data tersebut.



Gambar 2.6 (a) *Histogram equalization*, (b) *Compression*, (c) *Normalization*, (d) *Noise reduction*,

Sumber: (Hedenfalk & Rehnström, 2024)

Dalam konteks Gim interaktif, Computer Vision menjadi sangat penting, terutama dalam mendukung *real-time processing*, yang memungkinkan interaksi lebih responsif dan natural antara pemain dan *game*. Hal ini mencakup penggunaan teknologi seperti pelacakan gerak, deteksi wajah, hingga analisis ekspresi pemain untuk menciptakan interaksi yang *imersif*. Evolusi Computer Vision dalam pengembangan Gim tidak hanya meningkatkan kualitas visual tetapi juga membuka peluang untuk menciptakan *gameplay* yang adaptif dan lebih personal. Dengan terus berkembangnya algoritma dan kemampuan *hardware*, potensi penerapan Computer Vision dalam pengembangan Gim semakin luas, mencakup area seperti AR, VR, dan pengalaman bermain berbasis *gesture*.

2.3 Deteksi Objek

Deteksi objek adalah salah satu bidang penting dalam computer vision yang berfokus pada identifikasi dan lokalisasi objek dalam gambar atau video. Kemajuan dalam deep learning telah membawa perkembangan signifikan dalam bidang ini, yang menghasilkan berbagai arsitektur seperti Region-based Convolutional Neural Network (R-CNN), You Only Look Once (YOLO), dan Single Shot Detector (SSD). Masing-masing arsitektur ini menggunakan metode yang berbeda untuk mengatasi tantangan dalam deteksi objek, terutama dalam aspek kecepatan dan akurasi.

. R-CNN, yang diperkenalkan oleh (Girshick, 2015), merevolusi deteksi objek dengan menggabungkan proposal wilayah dengan *deep learning*, sehingga meningkatkan akurasi dalam mengenali objek dalam gambar. Pendekatan ini kemudian disempurnakan lebih lanjut dengan Faster R-CNN, yang mengintegrasikan Region Proposal Network (RPN) untuk mempercepat proses pembuatan proposal wilayah, sehingga mengurangi waktu komputasi

tanpa mengorbankan tingkat akurasi deteksi (Ren et al., 2017). Model-model dalam keluarga R-CNN telah menjadi landasan bagi perkembangan deteksi objek berikutnya, menunjukkan efektivitas jaringan konvolusi dalam tugas deteksi yang kompleks (Girshick, 2015; Ren et al., 2017).

Berbeda dengan R-CNN, YOLO menawarkan pendekatan yang lebih terpadu dengan merancang deteksi objek sebagai masalah regresi. Metode ini memungkinkan deteksi *real-time* dengan memprediksi *bounding boxes* dan probabilitas kelas langsung dari seluruh gambar dalam satu proses evaluasi (Redmon et al., 2016). Desain arsitektur YOLO menekankan pada kecepatan, menjadikannya sangat cocok untuk aplikasi yang membutuhkan umpan balik instan, seperti kendaraan otonom dan sistem pengawasan. Kemampuannya untuk memproses gambar dengan cepat tanpa mengorbankan akurasi membuat YOLO menjadi pilihan populer untuk berbagai aplikasi *real-time* (Redmon et al., 2016).

SSD juga menawarkan *one-stage detection* yang menyeimbangkan kecepatan dan akurasi dengan mendiskretkan ruang keluaran *bounding box* ke dalam sekumpulan *default boxes* dengan berbagai rasio aspek dan skala (Liu et al., 2016). Arsitektur ini sangat efektif dalam mendeteksi objek dengan ukuran berbeda dalam satu pemrosesan, yang membuatnya ideal untuk aplikasi *real-time*. Model SSD terbukti mampu mengungguli metode tradisional dalam hal kecepatan pemrosesan, sambil tetap mempertahankan tingkat akurasi yang kompetitif (Liu et al., 2016).

Kemajuan dalam arsitektur deteksi objek ini menyoroti evolusi teknologi yang terus berkembang, didorong oleh perkembangan dalam *deep learning*. Seiring dengan peningkatan dalam model-model ini, deteksi objek semakin banyak diterapkan dalam berbagai bidang, mulai dari sistem otomatisasi di sektor pertanian, gim hingga keamanan dan pengawasan. Teknologi ini semakin membuktikan fleksibilitas dan efektivitasnya dalam berbagai skenario di dunia nyata (Kamilaris & Prenafeta-Boldú, 2018; Lecun et al., 2015).

2.4 OpenCV (Open Source Computer Vision Library)

OpenCV (Open Source Computer Vision Library) adalah pustaka *open-source* yang dirancang untuk memfasilitasi pengembangan aplikasi dalam bidang Computer Vision dan Machine Learning. Sejak pertama kali dirilis oleh Intel pada tahun 2000, OpenCV telah menjadi alat utama dalam penelitian dan pengembangan berbasis Computer Vision karena fleksibilitasnya serta dukungannya terhadap berbagai platform, termasuk Windows, macOS, Linux, dan perangkat *mobile* (Uke, 2012). Pustaka ini menyediakan berbagai fungsi untuk menangani tugas-tugas seperti pengolahan citra, analisis video, deteksi objek, pelacakan gerak,

segmentasi gambar, dan pengenalan wajah (Uke, 2012). Selain itu, OpenCV mendukung berbagai bahasa pemrograman seperti Python, C++, dan Java, yang membuatnya dapat digunakan oleh berbagai kalangan pengembang.

Salah satu fitur utama OpenCV adalah kemampuannya dalam deteksi objek dan pengenalan wajah. Berbagai algoritma telah dikembangkan untuk memanfaatkan pustaka ini dalam aplikasi nyata, seperti deteksi wajah menggunakan metode Haar cascade, yang telah menjadi salah satu teknik populer dalam pengenalan wajah (Guenouni et al., 2015). Penelitian menunjukkan bahwa OpenCV dapat digunakan dalam berbagai aplikasi, mulai dari Human-Computer Interaction (HCI), pengenalan objek, hingga pelacakan gerakan (Erwin Syahrudin et al., 2024). Dalam konteks ini, penggunaan Python dengan OpenCV telah terbukti lebih efisien dalam hal waktu eksekusi dan kemudahan integrasi dalam program (Jalled & Voronkov, 2016).

OpenCV telah digunakan dalam berbagai bidang, termasuk pertanian, di mana algoritma pengolahan citra digunakan untuk estimasi parameter morfologis tanaman. Selain itu, pustaka ini juga banyak diterapkan dalam gim interaktif karena efisiensi dan kemampuannya untuk menangani pemrosesan data visual secara *real-time* dan dalam sistem keamanan, seperti deteksi kendaraan darurat dan pengendalian lalu lintas (Reddy, 2023). Dengan banyaknya aplikasi yang dapat diimplementasikan menggunakan OpenCV, pustaka ini menjadi alat yang sangat berharga bagi para peneliti dan pengembang dalam menciptakan solusi berbasis Computer Vision yang inovatif dan efisien (Guenouni et al., 2015). Dengan fleksibilitas, dukungan multi-platform, dan kompatibilitas dengan berbagai bahasa pemrograman, OpenCV terus menjadi pilihan utama dalam pengembangan aplikasi berbasis Computer Vision dan Machine Learning di berbagai industri.

2.5 YOLO (You Only Look Once)

YOLO (You Only Look Once) adalah metode deteksi objek yang sangat efisien dan cepat, yang telah mengalami perkembangan signifikan sejak diperkenalkan oleh Joseph Redmon pada tahun 2016. Metode ini membagi gambar menjadi *grid* dan memungkinkan deteksi objek dilakukan dalam satu langkah dengan memprediksi lokasi objek melalui *bounding box* dan mengklasifikasikan jenis objek secara simultan. Keunggulan utama dari YOLO adalah kecepatannya yang tinggi dan efisiensi komputasi, menjadikannya sangat cocok untuk aplikasi *real-time* seperti pengawasan CCTV, kendaraan otonom, dan sistem parkir pintar (Hussain, 2023; Sholahuddin et al., 2023).

Versi terbaru dari YOLO dari Ultralytics, yaitu YOLOv8, menawarkan peningkatan signifikan dalam akurasi dan efisiensi dibandingkan dengan pendahulunya. Arsitektur YOLOv8 menggunakan pendekatan *single-stage detection* yang memungkinkan pemrosesan seluruh gambar dalam satu *forward pass*, yang menghasilkan prediksi lebih cepat dibandingkan dengan metode *two-stage detection*. Selain itu, YOLOv8 mengimplementasikan berbagai peningkatan seperti CSPDarknet *backbone* dan PANet *neck*, serta teknik *augmentasi* yang meningkatkan ketahanan model terhadap berbagai kondisi (Gao et al., 2024; Terven et al., 2023). Penelitian menunjukkan bahwa YOLOv8 memiliki kinerja yang sangat baik dalam hal *mean average precision* (mAP), *recall*, dan *precision*, menjadikannya alat yang sangat efektif untuk berbagai aplikasi praktis (Abdullah et al., 2024; H. Huang et al., 2024).

Dalam konteks aplikasi *real-time*, YOLOv8, khususnya model YOLOv8n.pt, telah terbukti sebagai pilihan terbaik. Model ini dirancang untuk mencapai keseimbangan optimal antara kecepatan dan akurasi, yang sangat penting dalam aplikasi seperti deteksi kendaraan dalam sistem parkir dan pengawasan lalu lintas. Penelitian menunjukkan bahwa YOLOv8n.pt mampu memberikan hasil yang lebih baik dalam hal kecepatan deteksi dan akurasi dibandingkan dengan model lain, menjadikannya sangat cocok untuk aplikasi yang memerlukan respons cepat dan akurasi tinggi (Khude et al., 2024; Patel et al., 2024)

2.6 Hyperparameter Pelatihan YOLOv8

Dalam deteksi objek menggunakan YOLO (You Only Look Once), khususnya pada versi Ultralytics YOLO, pengaturan *hyperparameter* merupakan langkah krusial yang harus dilakukan sebelum pelatihan dan tetap konstan selama proses pelatihan. Salah satu *hyperparameter* yang paling penting adalah *learning rate* (lr0), yang menentukan ukuran langkah dalam pembaruan bobot model untuk meminimalkan fungsi kerugian. Pengaturan *learning rate* yang optimal sangat diperlukan, karena jika terlalu tinggi, model dapat melewati nilai minimum optimal, sedangkan jika terlalu rendah, pelatihan dapat menjadi sangat lambat. Studi menunjukkan bahwa pemilihan *learning rate* yang tepat dapat mempercepat konvergensi model dan mengurangi waktu pelatihan secara signifikan (Y.-Q. Huang et al., 2020).

Selain *learning rate*, *batch size* juga memiliki pengaruh besar dalam pelatihan model. *Batch size* menentukan jumlah gambar yang diproses dalam satu iterasi pelatihan. Ukuran *batch* yang besar dapat meningkatkan stabilitas estimasi gradien, tetapi juga membutuhkan memori yang lebih besar serta waktu komputasi yang lebih lama. Menurut penelitian oleh (Y.-Q. Huang et al., 2020), pengaturan *batch size* yang tepat dapat mempengaruhi kecepatan pelatihan dan penggunaan memori, serta berdampak langsung pada akurasi model. Oleh karena

itu, pemilihan *batch size* yang sesuai sangat penting untuk mencapai keseimbangan antara efisiensi pelatihan dan penggunaan sumber daya komputasi.

Selain itu, jumlah *epoch* juga memainkan peran penting dalam efektivitas pelatihan. *Epoch* merujuk pada satu siklus penuh di mana seluruh dataset pelatihan digunakan untuk memperbarui bobot model. Jumlah *epoch* yang terlalu sedikit dapat menyebabkan *underfitting*, sedangkan *epoch* yang terlalu banyak dapat menyebabkan *overfitting* (Sharafaldein et al., 2022). Studi menunjukkan bahwa pelatihan selama 200 epoch dengan penyimpanan bobot setiap 10 *epoch* dapat menghasilkan model yang lebih baik dalam hal akurasi. Selain itu, spesifikasi arsitektur model, seperti jumlah saluran, jumlah lapisan, dan jenis fungsi aktivasi yang digunakan, juga sangat berpengaruh terhadap kemampuan model dalam mempelajari pola dari data pelatihan.

Dengan demikian, pengaturan *hyperparameter* yang cermat, termasuk *learning rate*, *batch size*, dan jumlah *epoch*, serta pemilihan arsitektur model yang sesuai, sangat penting dalam meningkatkan performa YOLO dalam tugas deteksi objek. Optimalisasi *hyperparameter* tidak hanya membantu dalam mencapai keseimbangan antara kecepatan dan akurasi, tetapi juga memastikan model mampu bekerja dengan efisien dalam berbagai skenario deteksi objek.

2.7 Metrik Evaluasi YOLOv8

Metrik evaluasi digunakan untuk menilai kinerja model deteksi objek, dan beberapa metrik utama yang relevan dalam konteks deteksi objek seperti YOLO antara lain *Intersection over Union (IoU)*, *Average Precision (AP)*, *Mean Average Precision (mAP)*, *precision*, *recall*, dan *F1 score*. IoU adalah metrik yang mengukur tumpang tindih antara kotak pembatas yang diprediksi oleh model dan kotak pembatas kebenaran dasar (*ground truth*). IoU dihitung dengan cara membandingkan luas irisan (*intersection*) antara kedua kotak pembatas dengan luas gabungan (*union*) dari kedua kotak tersebut. Semakin tinggi nilai IoU, semakin baik model dalam mendeteksi objek dengan akurat, yang dihitung dengan rumus (2.1):

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} \quad (2.2)$$

- a. *Area of Intersection* adalah luas area yang tumpang tindih antara kotak pembatas yang diprediksi dan kotak pembatas kebenaran dasar.
- b. *Area of Union* adalah luas area yang mencakup kedua kotak pembatas tersebut (tumpang tindih dan non-tumpang tindih).

Nilai IoU berkisar antara 0 hingga 1, di mana nilai 1 menunjukkan bahwa kotak pembatas yang diprediksi dan kotak pembatas kebenaran dasar sepenuhnya tumpang tindih dan sangat

akurat, sedangkan nilai 0 berarti tidak ada tumpang tindih sama sekali. IoU digunakan sebagai dasar untuk menilai seberapa baik model dalam lokalisasi objek, yaitu mendeteksi objek di tempat yang tepat dalam gambar (Simon et al., 2019).

AP mengukur kinerja model dengan menggabungkan *precision* dan *recall* dalam satu nilai tunggal. *Precision* mengukur seberapa banyak dari prediksi yang dilakukan oleh model yang benar-benar relevan, sementara *recall* mengukur seberapa banyak dari total objek yang relevan yang berhasil ditemukan oleh model. AP dihitung dengan area di bawah kurva *precision-recall*, memberikan gambaran tentang kemampuan model dalam menyeimbangkan *false positives* (FP) dan *false negatives* (FN). Rumus (2.3) untuk AP adalah sebagai berikut:

$$AP = \int_0^1 Precision(r) dr \quad (2.4)$$

- a. *Precision(r)* adalah *precision* yang dihitung pada berbagai tingkat *recall* r , dan integral ini mengukur area di bawah kurva *precision-recall* yang menunjukkan bagaimana *precision* berubah saat *recall* bervariasi.

AP memberikan nilai 0 hingga 1, di mana 1 menunjukkan kinerja model yang sangat baik, di mana model mampu menemukan objek dengan sangat sedikit *false positives* dan *false negatives* (Wang et al., 2023). AP ini penting untuk menilai kinerja model pada tugas deteksi objek, terutama untuk aplikasi yang mengharuskan keseimbangan antara *precision* dan *recall*.

mAP adalah perpanjangan dari AP, yang mengukur rata-rata nilai AP di seluruh kelas objek dalam dataset. mAP memberikan evaluasi yang lebih komprehensif dari kinerja model dalam situasi deteksi multi-kelas. Ini menggabungkan kinerja deteksi objek dalam berbagai kategori dan memberikan gambaran keseluruhan dari kemampuan model untuk mendeteksi berbagai jenis objek dalam satu pengujian. Rumus (2.5) untuk mAP adalah sebagai berikut:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (2.6)$$

- a. N adalah jumlah total kelas objek yang ada dalam dataset,
- b. AP_i adalah nilai *Average Precision* untuk kelas i

mAP digunakan untuk mengevaluasi deteksi objek multi-kelas, di mana setiap kelas objek memiliki kesulitan deteksi yang berbeda. Nilai mAP memberikan gambaran yang lebih adil mengenai kinerja model dalam menangani berbagai kelas objek secara bersamaan.

Precision dan *Recall* adalah dua metrik dasar yang digunakan untuk mengukur kinerja model deteksi objek. *Precision* mengukur seberapa banyak prediksi positif yang benar di antara semua prediksi positif yang dibuat oleh model, sementara *recall* mengukur seberapa banyak

dari objek yang relevan (positif sebenarnya) yang berhasil ditemukan oleh model. *Precision* dihitung dengan rumus (2.7):

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (2.8)$$

Di mana:

- a. *True Positives* (TP) adalah jumlah objek yang benar-benar terdeteksi oleh model,
- b. *False Positives* (FP) adalah jumlah objek yang salah terdeteksi oleh model sebagai positif.

Recall dihitung dengan rumus (2.9):

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (2.10)$$

Di mana:

- a. *False Negatives* (FN) adalah jumlah objek yang tidak terdeteksi oleh model meskipun objek tersebut ada.

Kedua metrik ini sangat penting karena *precision* mengukur seberapa tepat model dalam melakukan prediksi, sedangkan *recall* mengukur seberapa baik model dalam menemukan semua objek yang relevan (Diwan et al., 2023). *Precision* yang tinggi mengindikasikan bahwa sebagian besar prediksi positif yang dilakukan oleh model adalah benar, sementara *recall* yang tinggi menunjukkan bahwa model berhasil mendeteksi sebagian besar objek yang relevan.

F1 Score adalah rata-rata harmonis antara *precision* dan *recall*, yang memberikan gambaran yang seimbang tentang kinerja model dalam mendeteksi objek. F1 score sangat berguna ketika terdapat ketidakseimbangan antara kelas positif dan negatif dalam dataset, karena memberikan bobot yang sama untuk *precision* dan *recall*, menghindari fokus berlebihan pada salah satu metrik. Rumus F1 Score (2.11) adalah:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.12)$$

F1 score memberikan nilai antara 0 dan 1, di mana nilai 1 menunjukkan kinerja model yang sangat baik dalam mendeteksi objek, dengan keseimbangan yang optimal antara *precision* dan *recall* (Gündüz & Işık, 2023). F1 score ini berguna untuk menilai model dalam situasi di mana ada ketidakseimbangan kelas dalam dataset, karena memperhitungkan kedua metrik secara seimbang.

2.8 LabelImg

LabelImg adalah perangkat lunak *open-source* yang dirancang untuk pelabelan data, khususnya dalam proses anotasi dataset untuk tugas-tugas Computer Vision seperti deteksi objek (Tzutalin, 2015). Perangkat lunak ini banyak digunakan karena sifatnya yang *open-*

source, kemudahan penggunaannya, dan dukungan dari komunitas yang besar, sehingga tutorial atau panduan penggunaannya mudah ditemukan (Sun & Loparo, 2019). Dengan antarmuka yang intuitif, LabelImg memungkinkan pengguna untuk menandai area objek menggunakan *bounding box* pada setiap *frame* video yang telah diekstraksi menjadi gambar-gambar individu.

Selain itu, LabelImg mendukung berbagai format anotasi, seperti PASCAL VOC XML dan YOLO TXT, yang mempermudah integrasi dengan *framework* pembelajaran mesin populer. Kemampuan ini menjadikannya alat penting dalam persiapan dataset berkualitas tinggi, yang merupakan fondasi utama untuk melatih model deteksi objek yang andal.

2.9 Exertion Interface

Exertion Interface adalah konsep yang mengintegrasikan aktivitas fisik sebagai elemen interaktif dalam gim, di mana pemain menggunakan tubuh mereka, seperti berlari, melompat, atau menggerakkan tangan, untuk memengaruhi jalannya permainan. Konsep ini bertujuan menciptakan pengalaman imersif, di mana tubuh pemain menjadi bagian utama dalam kontrol gim, menggantikan perangkat eksternal seperti joystick atau mouse (F. Mueller et al., 2003). Aktivitas fisik ini bertujuan untuk meningkatkan keterlibatan pemain, dengan harapan dapat mendorong mereka bergerak lebih banyak, yang juga memberikan manfaat bagi kesehatan.

Berbeda dengan gim tradisional, Exertion Interface mendorong interaksi langsung antara pemain dan dunia virtual, sehingga menciptakan pengalaman yang lebih alami dan responsif. Sebagai contoh, dalam gim berbasis Exertion Interface, pemain bisa diminta untuk bergerak, menghindari rintangan, melempar bola, atau berlari untuk mengejar objek. Pendekatan ini menggabungkan hiburan dengan aktivitas fisik, tanpa menjadikan kebugaran sebagai tujuan utama, tetapi tetap memberikan dampak positif bagi kesehatan tubuh.

Di sisi lain, Exergame lebih terfokus pada aktivitas fisik sebagai tujuan utama untuk meningkatkan kebugaran atau mendukung rehabilitasi fisik. Platform seperti Lu Interactive Playground bertujuan memberikan pengalaman fisik yang menyenangkan, sambil mendorong gerakan yang bermanfaat untuk kebugaran tubuh (Lü Interactive Playground, 2024). Berbeda dengan Exertion Interface, yang mengutamakan hiburan, Exergame dirancang untuk kebugaran yang terukur.

Perbedaan utama antara keduanya terletak pada tujuan dan fokusnya. Exertion Interface menggabungkan aktivitas fisik untuk menciptakan pengalaman hiburan yang menyenangkan, sementara Exergame bertujuan meningkatkan kesehatan fisik secara terstruktur, seringkali digunakan untuk terapi atau program kebugaran.

2.10 Gim Lempar Bola sebagai *Proof Of Concept*

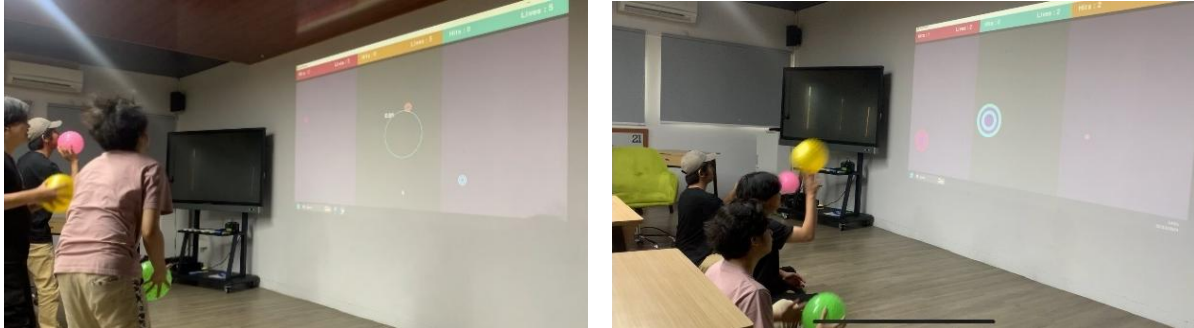
Gim lempar bola yang digunakan sebagai *proof of concept* dalam penelitian ini merupakan gim interaktif yang berakar pada konsep Exertion Interface, yaitu jenis gim yang menggabungkan elemen aktivitas fisik dengan interaksi digital. Konsep ini diadaptasi dari penelitian yang dilakukan oleh platform seperti (Lü Interactive Playground, 2024) yang secara khusus mengembangkan Exertion Interface dalam berbagai bentuk gim interaktif. Exertion Interface memiliki potensi besar untuk meningkatkan keterlibatan pemain melalui aktivitas fisik yang langsung terintegrasi dalam alur permainan. Dalam konteks ini, pemain tidak hanya berinteraksi dengan objek atau karakter di dunia virtual melalui kontrol eksternal, tetapi juga dengan gerakan tubuh mereka. Misalnya, pemain dapat menggunakan tubuh mereka untuk memanipulasi objek dalam gim, seperti melempar bola atau menghindari rintangan, yang memberikan pengalaman bermain yang lebih imersif dan responsif.

Salah satu contoh Exertion Interface yang dikembangkan adalah gim Target, di mana pemain berinteraksi dengan objek virtual, seperti bola, yang harus diarahkan atau dilemparkan menuju sasaran tertentu untuk mencapai tujuan atau skor dalam gim. Dalam gim ini, aktivitas fisik pemain, seperti melempar bola atau menggerakkan tubuh untuk mengarahkan bola, menjadi bagian integral dari kontrol gim. Konsep ini mengharuskan pemain untuk bergerak secara fisik, yang tidak hanya meningkatkan interaktivitas tetapi juga memberikan pengalaman bermain yang lebih imersif. Pemain merasakan keterlibatan yang lebih langsung dalam permainan, di mana tubuh mereka bukan hanya pengarah, tetapi juga faktor penentu dalam hasil permainan. Exertion Interface dalam bentuk gim Target ini mendorong pemain untuk lebih aktif bergerak, yang pada gilirannya dapat berkontribusi pada peningkatan kebugaran fisik, tanpa mengorbankan elemen hiburan dalam gim.

Meskipun gim yang dikembangkan dalam Lu Interactive Playground tidak tersedia untuk publik, penting untuk melanjutkan pengembangan dengan membangun prototipe serupa, khususnya gim Target, sebagai *proof of concept*. Prototipe ini akan mengintegrasikan elemen Exertion Interface, di mana pemain berinteraksi dengan objek virtual melalui gerakan fisik untuk mencapai sasaran, guna menguji potensi teknologi dalam menciptakan pengalaman bermain yang lebih imersif dan interaktif. Dalam konteks ini, penggunaan platform PyGame untuk mengembangkan prototipe gim lempar bola menjadi pilihan yang strategis. PyGame adalah pustaka Python yang banyak digunakan untuk pengembangan prototipe gim sederhana hingga gim 2D yang kompleks. PyGame dirancang untuk bersifat cross-platform, sehingga gim yang dikembangkan dapat dijalankan di berbagai sistem operasi tanpa banyak modifikasi

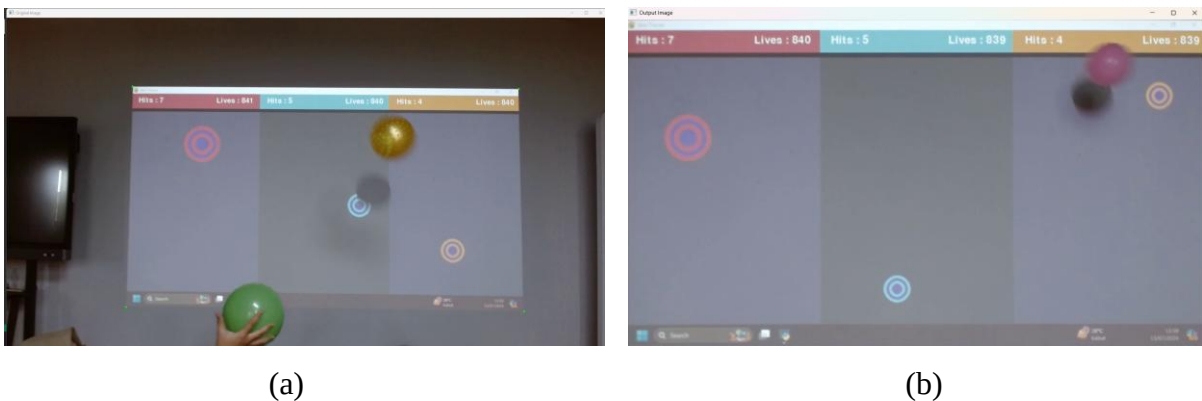
(Mahardhika, 2019). Berikut adalah tahapan pengembangan prototipe gim lempar bola sederhana sebagai PoC:

a. Algoritma Kalibrasi Kamera



Gambar 2.7 Contoh Gim lempar bola di dalam kelas

Meskipun permainan ini dirancang dalam lingkungan yang terkontrol, penulis tetap menyertakan fitur kalibrasi untuk mengakomodasi lokasi dan posisi kamera yang berbeda. Fitur ini dirancang agar hasil penelitian dalam laporan Tugas Akhir (TA) ini dapat direplikasi dengan mudah untuk pengembangan lebih lanjut. Gambar 2.7 diatas memperlihatkan contoh penggunaan permainan di dalam kelas. Oleh karena itu, proses kalibrasi kamera harus dilakukan sebelum permainan dimulai seperti pada Gambar 2.8.



Gambar 2.8 (a) Kamera dengan titik hijau untuk di kalibrasi, (b) Kamera setelah di kalibrasi

Algoritma kalibrasi kamera bertujuan untuk memastikan area tangkapan kamera sesuai dengan tampilan yang diinginkan dalam aplikasi atau permainan berbasis Computer Vision. Proses ini dilakukan dengan memilih empat titik pada gambar *input* dari kamera, yang kemudian digunakan untuk membuat transformasi perspektif. Transformasi ini memungkinkan area gambar yang diinginkan untuk diratakan atau diubah perspektifnya sehingga sesuai dengan koordinat virtual yang dibutuhkan aplikasi.

Proses dimulai dengan membaca gambar dari kamera menggunakan pustaka OpenCV. Empat titik referensi dipilih secara manual oleh pengguna dengan cara mengklik pada area gambar yang ditampilkan. Koordinat titik-titik ini disimpan dalam sebuah array bernama `circles`. Ketika pengguna telah memilih empat titik, sistem membuat matriks transformasi perspektif dengan fungsi `cv2.getPerspectiveTransform`, yang memetakan titik-titik *input* pada gambar asli (`pts1`) ke titik-titik keluaran yang diinginkan (`pts2`). Dimensi keluaran gambar ditentukan sebelumnya, misalnya 1280x720 piksel. Hal ini lebih jelasnya dapat dilihat pada Gambar 2.9.

```

1 import cv2
2 import numpy as np
3 import pickle
4
5 width, height = 1280, 720
6 cap = cv2.VideoCapture(0)
7 fileName = 'calibration.p'
8 cap.set(3, width)
9 cap.set(4, height)
10 circles = np.zeros((4, 2), int)
11 counter = 0
12
13 def mousePoints(event, x, y, flags, params):
14     global counter
15     if event == cv2.EVENT_LBUTTONDOWN:
16         circles[counter] = x, y
17         counter += 1
18         print(circles)
19 while True:
20     success, img = cap.read()
21
22     if counter == 4:
23         fileObj = open(fileName, 'wb')
24         pickle.dump(circles, fileObj)
25         fileObj.close()
26
27         pts1 = np.float32([circles[0], circles[1], circles[2],
28 circles[3]])
29         pts2 = np.float32([[0, 0], [width, 0], [0, height], [width,
30 height]])
31         matrix = cv2.getPerspectiveTransform(pts1, pts2)
32         imgOutput = cv2.warpPerspective(img, matrix, (width, height))
33         cv2.imshow("Output Image", imgOutput)
34
35         for x in range(0, 4):
36             cv2.circle(img, (circles[x][0], circles[x][1]), 3, (0, 255, 0),
37 cv2.FILLED)
38
39         cv2.imshow("Original Image", img)
40         cv2.setMouseCallback("Original Image", mousePoints)
41
42         key = cv2.waitKey(1) & 0xFF
43         if key == ord('q'):
44             break
45         elif key == ord('r'):

```

```

43         circles = np.zeros((4, 2), int)
44         counter = 0
45 cap.release()
46 cv2.destroyAllWindows()

```

Gambar 2.9 Kode program calibration.py

Setelah matriks transformasi dibuat, sistem menerapkannya pada gambar menggunakan fungsi `cv2.warpPerspective`. Hasil transformasi ini adalah gambar yang terkalibrasi, di mana area yang dipilih sebelumnya diubah perspektifnya sehingga sesuai dengan koordinat yang diinginkan. Gambar hasil transformasi ditampilkan kepada pengguna untuk memastikan hasilnya sesuai.

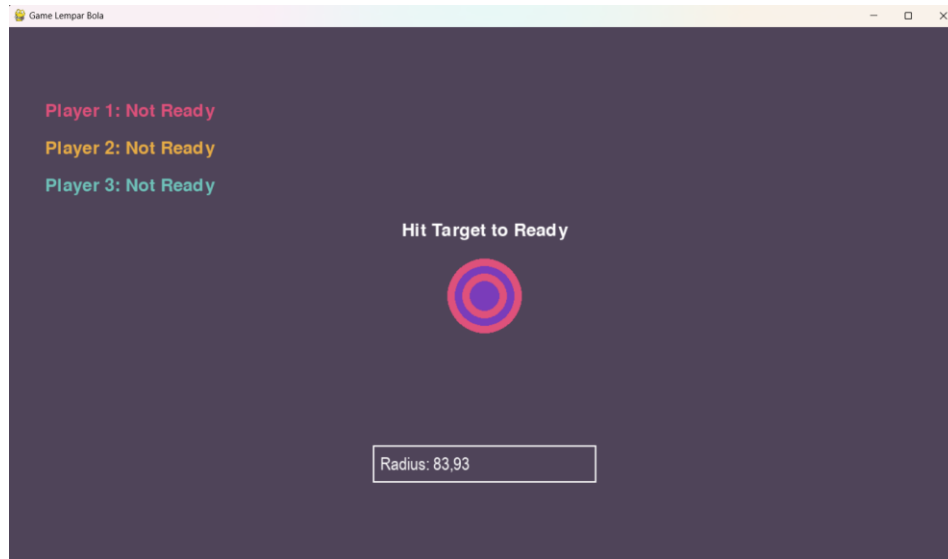
Selama proses kalibrasi, sistem juga menampilkan gambar asli dengan indikator visual berupa lingkaran kecil pada titik-titik yang telah dipilih, membantu pengguna untuk memverifikasi posisi titik-titik tersebut. Pengguna dapat menekan tombol tertentu, seperti `q` untuk keluar dari aplikasi atau `r` untuk mereset titik jika terjadi kesalahan dalam memilih.

Hasil dari proses kalibrasi ini disimpan dalam sebuah *file* bernama `calibration.p` menggunakan pustaka `pickle`. *File* ini berisi koordinat titik-titik yang dipilih, yang nantinya dapat digunakan kembali tanpa perlu melakukan proses kalibrasi ulang setiap kali aplikasi dijalankan. Dengan cara ini, kalibrasi menjadi lebih efisien dan memungkinkan transformasi perspektif dilakukan secara otomatis dalam aplikasi.

Keseluruhan algoritma memastikan bahwa sistem dapat mendeteksi objek dengan akurat dalam area yang telah disesuaikan perspektifnya, menjadikannya langkah penting dalam pengembangan aplikasi seperti permainan yang memanfaatkan deteksi target atau objek secara *real-time*.

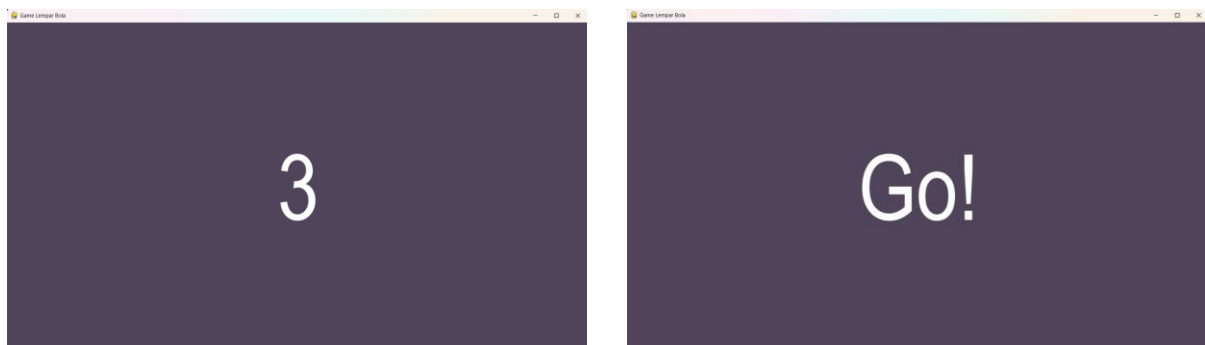
b. Algoritma Main Menu

Proses kalibrasi berikutnya adalah menentukan ukuran radius yang digunakan untuk mendeteksi *hit*. Karena ukuran radius bergantung pada jarak antara kamera dan layar permainan, kalibrasi radius dapat dilakukan pada *main menu* sebelum permainan dimulai. Proses ini dilakukan secara manual dengan menggerakkan bola sambil menyentuh setiap sudut dan sisi layar permainan untuk menentukan radius minimum dan maksimum yang terdeteksi saat bola menyentuh layar. Nilai minimum dan maksimum radius tersebut dapat dimasukkan secara manual ke dalam kolom input di *main menu* dengan format penulisan "*min, max*" seperti pada Gambar 2.10.



Gambar 2.10 Antarmuka *Main Menu* dengan inputan radius

Setelah semua pemain menekan tombol *ready*, permainan akan dimulai dengan hitungan mundur (*countdown*), seperti yang ditunjukkan pada Gambar 2.11. Fitur *countdown* ini dirancang untuk memberikan waktu kepada pemain agar dapat mempersiapkan diri sebelum permainan dimulai. Setiap pemain akan memiliki area permainan yang ditentukan berdasarkan warna bola yang dipegangnya, sehingga pemain berbagi jarak secara adil sesuai dengan area masing-masing.



Gambar 2.11 Antarmuka *countdown*

Fungsi `main_menu` pada Gambar 2.12 dalam permainan "Gim Lempar Bola" dirancang untuk memberikan pengalaman interaktif sekaligus memastikan sistem terkalibrasi dengan baik sebelum permainan dimulai. Menu utama ini menampilkan target lingkaran di tengah layar yang harus dipukul pemain sebagai tanda kesiapan. Status kesiapan setiap pemain, yang diwakili dengan label "*Ready*" atau "*Not Ready*," ditampilkan secara *real-time* di layar dengan warna unik untuk masing-masing pemain, memberikan informasi visual yang jelas.

```

1  def main_menu():
2      global MIN_RADIUS, MAX_RADIUS
3      run = True
4      ready_players = {0: False, 1: False, 2: False}
5      target_center = Target(WIDTH // 2, HEIGHT // 2, size=50)
6      clock = pygame.time.Clock()
7      input_active = False
8      input_box = pygame.Rect(WIDTH // 2 - 150, HEIGHT // 2 + 200, 300,
50)
9      user_input = f"{MIN_RADIUS},{MAX_RADIUS}"
10     font = pygame.font.SysFont("Arial", 24)
11
12     while run:
13         clock.tick(60)
14         WIN.fill(BG1_COLOR)
15         target_center.draw(WIN, P1_COLOR)
16         title_text = LABEL_FONT.render("Hit Target to Ready", 1,
"white")
17         WIN.blit(title_text, (WIDTH // 2 - title_text.get_width() //
2, HEIGHT // 2 - 100))
18
19         for i, color in enumerate([P1_COLOR, P2_COLOR, P3_COLOR]):
20             status = "Ready" if ready_players[i] else "Not Ready"
21             status_text = LABEL_FONT.render(f"Player {i + 1}:
{status}", 1, color)
22             WIN.blit(status_text, (50, 100 + i * 50))
23
24         pygame.draw.rect(WIN, (255, 255, 255), input_box, 2)
25         input_text = font.render(f"Radius: {user_input}", True, (255,
255, 255))
26         WIN.blit(input_text, (input_box.x + 10, input_box.y + 10))
27
28         success, img = cap.read()
29         if not success:
30             pygame.display.update()
31             continue
32
33         scale_factor = 3
34         output_width = WIDTH // scale_factor
35         output_height = HEIGHT // scale_factor
36
37         img = warpImage(img, circles, output_width, output_height)
38         img = cv2.resize(img, (WIDTH, HEIGHT))
39         results = model(img, stream=True)
40
41         for r in results:
42             boxes = r.boxes
43             for box in boxes:
44                 x1, y1, x2, y2 = box.xyxy[0]
45                 x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
46                 center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
47                 cls = int(box.cls[0])
48                 color = P1_COLOR if cls == 0 else P2_COLOR if cls == 1
else P3_COLOR
49
50                 width = x2 - x1
51                 height = y2 - y1
52
53                 radius = math.sqrt((width**2 + height**2) / 2)

```

```

54         pygame.draw.circle(WIN, color, (center_x, center_y),
radius, 2)
55
56         label = f"{radius:.1f}"
57         label_surface = LABEL_FONT.render(label, 1, "white")
58         WIN.blit(label_surface, (center_x - radius, center_y -
radius))
59         print(f"Detected circle at ({center_x}, {center_y})
with radius {radius}")
60
61         if target_center.collide(center_x, center_y):
62             ready_players[cls] = True
63             print(f"Player {cls + 1} ready!")
64             cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
65             label = f"{model.names[cls]} {radius:.1f}"
66             cv2.putText(img, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
67
68         fps = int(clock.get_fps())
69         cv2.putText(img, f"FPS: {fps}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
70
71         cv2.imshow("Camera Output", img)
72
73         pygame.display.update()
74
75         for event in pygame.event.get():
76             if event.type == pygame.QUIT:
77                 pygame.quit()
78                 quit()
79
80             if event.type == pygame.MOUSEBUTTONDOWN:
81                 if input_box.collidepoint(event.pos):
82                     input_active = True
83                 else:
84                     input_active = False
85
86             if event.type == pygame.KEYDOWN and input_active:
87                 if event.key == pygame.K_RETURN:
88                     try:
89                         min_r, max_r = map(int, user_input.split(","))
90                         MIN_RADIUS, MAX_RADIUS = min_r, max_r
91                     except ValueError:
92                         pass
93                     user_input = f"{MIN_RADIUS},{MAX_RADIUS}"
94                 elif event.key == pygame.K_BACKSPACE:
95                     user_input = user_input[:-1]
96                 else:
97                     user_input += event.unicode
98
99             if all(ready_players.values()):
100                 countdown()
101                 return

```

Gambar 2.12 Kode fungsi *main menu*

Selain itu, fungsi ini menyediakan fitur kalibrasi radius bola, yang sangat penting untuk memastikan deteksi bola berjalan akurat selama permainan. Pemain dapat menyesuaikan nilai

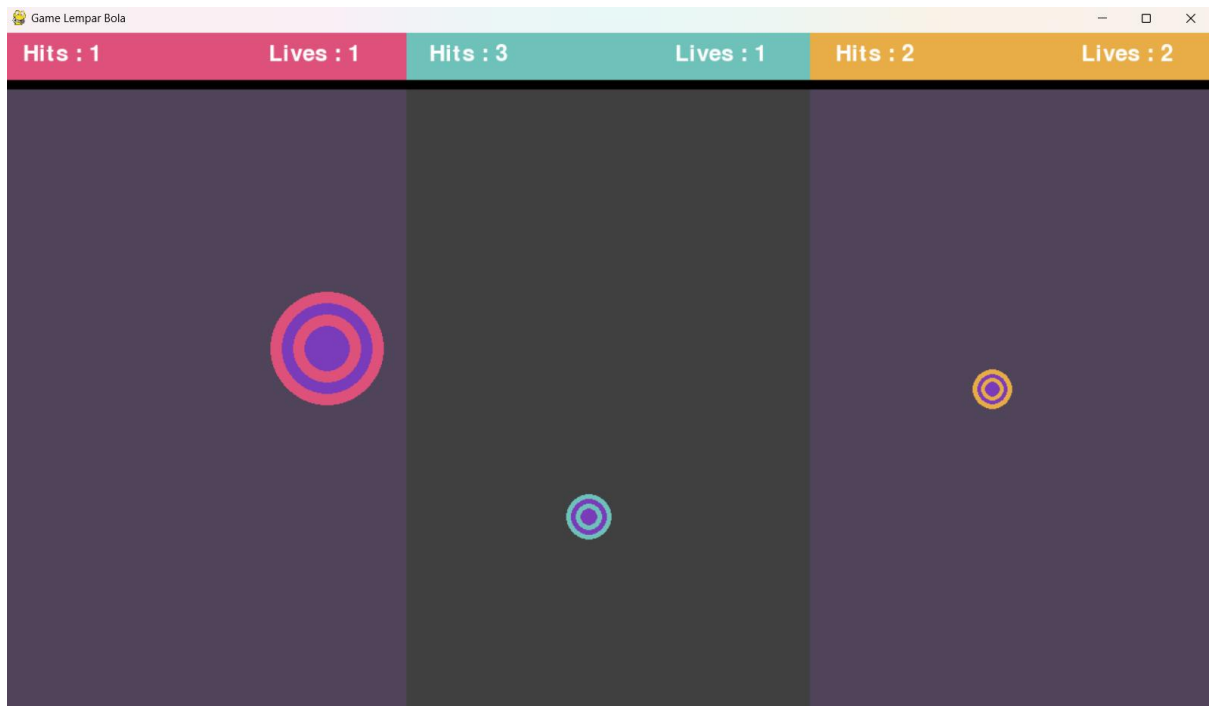
radius minimum dan maksimum bola yang akan dikenali oleh sistem melalui kotak *input* yang ditampilkan di layar. Ketika radius diubah, nilai baru akan disimpan, dan deteksi bola akan diperbarui sesuai dengan parameter ini. Untuk membantu pemain memahami bagaimana radius ini diterapkan, sistem menampilkan lingkaran deteksi secara visual di layar, sehingga pemain dapat memverifikasi apakah bola dengan ukuran tertentu dapat dikenali dengan benar.

Fungsi ini juga terintegrasi dengan kamera yang membaca gambar secara *real-time*. Sistem menggunakan algoritma YOLO untuk mendeteksi keberadaan bola, menghitung radiusnya, dan memeriksa apakah bola berhasil mengenai target lingkaran. Jika deteksi bola menunjukkan bahwa target berhasil dipukul, status kesiapan pemain yang bersangkutan diubah menjadi "Ready." Semua interaksi ini divisualisasikan dengan jelas di layar, memberikan umpan balik langsung kepada pemain.

Transisi ke permainan dilakukan hanya jika semua pemain telah menyatakan kesiapan mereka dengan memukul target. Dengan cara ini, fungsi `main_menu` tidak hanya berfungsi sebagai layar persiapan, tetapi juga sebagai mekanisme penting untuk kalibrasi sistem, memastikan bahwa deteksi bola dan parameter permainan telah diatur dengan benar sebelum permainan dimulai. Pendekatan ini memberikan pengalaman yang menyenangkan dan memastikan keandalan sistem selama permainan berlangsung.

c. Algoritma Pengelolaan Target

Area permainan akan dibagi menjadi tiga zona berdasarkan warna bola pemain: kiri (pink), tengah (hijau), dan kanan (kuning) yang terlihat pada Gambar 2.13. Target akan muncul di setiap zona dengan ukuran yang bertahap membesar hingga mencapai batas maksimal, kemudian mengecil kembali sebelum akhirnya menghilang jika tidak terkena bola. Ketika bola berhasil mengenai target, target tersebut akan digantikan dengan target baru di zona lain yang sesuai dengan warna pemain. Pemain tidak dapat menghancurkan target milik pemain lain; jika bola mengenai target di zona yang salah, sistem akan mencatatnya sebagai *miss* yang akan mengurangi nilai akurasi di akhir.



Gambar 2.13 Antarmuka permainan utama dengan pembagian area permainan

Pada permainan ini, pengelolaan target dilakukan melalui serangkaian fungsi yang bertanggung jawab atas kemunculan, penggambaran, dan siklus hidup target di layar. Fungsi `update_targets` di Gambar 2.14 berperan penting dalam mengatur siklus hidup target, yaitu memantau apakah target masih aktif atau perlu dihapus. Fungsi ini memperbarui ukuran setiap target dengan memanggil metode `update` dari objek `Target`. Jika ukuran target menjadi nol atau lebih kecil, target tersebut dihapus dari daftar target aktif, dan jumlah kesalahan (`misses`) pemain bertambah, menunjukkan bahwa target tidak berhasil dipukul sebelum menghilang.

```
1 def update_targets(targets, misses):
2     for target in targets:
3         target.update()
4         if target.size <= 0:
5             targets.remove(target)
6             misses += 1
7     return misses
```

Gambar 2.14 Kode fungsi `update_targets`

Metode `update` dalam kelas `Target` pada Gambar 2.15 baris ke 12 bertanggung jawab atas animasi target, yaitu perubahan ukuran target secara dinamis. Ketika target mencapai ukuran maksimum yang telah ditentukan (`MAX_SIZE`), status pertumbuhan (`grow`) diubah dari `True` menjadi `False`, sehingga target mulai mengecil. Sebaliknya, selama `grow` bernilai `True`, ukuran target bertambah dengan nilai tertentu yang ditentukan oleh

GROWTH_RATE. Proses ini menciptakan efek visual di mana target tampak hidup, memberikan tantangan bagi pemain untuk memukul target sebelum ukurannya menghilang.

```

1 class Target:
2     MAX_SIZE = 70
3     GROWTH_RATE = 0.3
4     SECOND_COLOR = 122, 60, 186
5
6     def __init__(self, x, y, size=0):
7         self.x = x
8         self.y = y
9         self.size = size
10        self.grow = True
11
12    def update(self):
13        if self.size + self.GROWTH_RATE >= self.MAX_SIZE:
14            self.grow = False
15
16        if self.grow:
17            self.size += self.GROWTH_RATE
18        else:
19            self.size -= self.GROWTH_RATE
20
21    def draw(self, win, color):
22        pygame.draw.circle(win, color, (self.x, self.y), self.size)
23        pygame.draw.circle(win, self.SECOND_COLOR, (self.x, self.y),
self.size * 0.8)
24        pygame.draw.circle(win, color, (self.x, self.y), self.size *
0.6)
25        pygame.draw.circle(win, self.SECOND_COLOR, (self.x, self.y),
self.size * 0.4)
26
27    def collide(self, x, y):
28        dis = math.sqrt((self.x - x)**2 + (self.y - y)**2)
29        return dis <= self.size

```

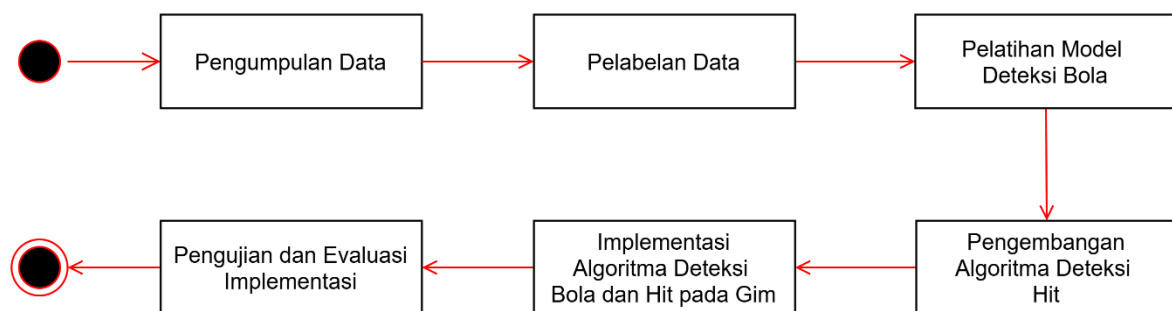
Gambar 2.15 Kode kelas Target

Penggambaran target di layar diatur oleh metode `draw` dalam kelas `Target` pada Gambar 2.15 pada baris ke 21. Metode ini menggambar target menggunakan lingkaran-lingkaran bertingkat dengan ukuran dan warna yang bervariasi. Lingkaran terluar memiliki ukuran penuh, sedangkan lingkaran-lingkaran di dalamnya memiliki ukuran yang lebih kecil secara bertahap. Kombinasi warna utama (`color`) dan warna sekunder (`SECOND_COLOR`) menciptakan efek visual yang menarik, membuat target lebih dinamis dan menonjol di layar.

BAB III

METODOLOGI PENELITIAN

Penelitian ini berfokus pada implementasi Computer Vision berbasis YOLOv8 untuk deteksi bola dalam pengembangan gim interaktif sebagai PoC. Untuk itu, metodologi penelitian ini dirancang dengan urutan yang terstruktur untuk memastikan keberhasilan dari setiap tahapan, mulai dari pengembangan ide hingga evaluasi akhir. Alur metodologi penelitian ini meliputi 6 tahapan utama, seperti yang terlihat pada Gambar 3.1.



Gambar 3.1 Alur Penelitian.

3.1 Pengumpulan Data

Tahap pengumpulan data dimulai dari melakukan perekaman video simulasi permainan dalam lingkungan terkontrol yang akan dijelaskan lebih jelasnya pada Bab IV. Hal ini dilakukan untuk memastikan data yang konsisten dan berkualitas tinggi, serta mencerminkan bagaimana Gim ini akan dimainkan. Dari video yang dihasilkan akan di ekstrak gambar dari tiap *frame* yang ada, sehingga menghasilkan dataset yang lengkap. Dataset tersebut akan digunakan sebagai data mentah untuk pelatihan model deteksi objek YOLOv8. Tiap gambar akan memuat tiga bola berbeda warna yang ditempatkan dalam berbagai posisi di depan layar. Variasi ini dirancang untuk memastikan bahwa model dapat mengenali bola secara akurat dari semua sisi dan kondisi permainan yang mungkin terjadi. Pendekatan ini bertujuan untuk menciptakan basis pelatihan yang memadai sehingga model dapat mendeteksi bola dengan presisi tinggi dalam skenario permainan nyata.

Tahap ini juga mencakup pembersihan data gambar yang telah di ekstrak dari tiap *frame* yang ada pada video simulasi permainan. Gambar yang tidak memenuhi kriteria, seperti tidak menampilkan tiga bola secara bersamaan akan dihapus, peneliti hanya memilih gambar dengan kondisi bola yang terlihat jelas dan tidak terhalang atau terpotong. Hal tersebut dilakukan untuk

memastikan bahwa model YOLOv8 lebih mudah mengenali dan membedakan tiap bola berdasarkan warnanya saat muncul bersamaan.

3.2 Pelabelan Data

Proses pelabelan dilakukan secara manual menggunakan perangkat lunak LabelImg dengan tiga label, yaitu pinkball, yellowball, dan greenball, yang merepresentasikan bola merah muda, kuning, dan hijau seperti pada **Error! Reference source not found.** Label ini ditentukan untuk menandai lokasi dan warna setiap bola pada setiap gambar. Pendekatan manual dalam pelabelan memastikan akurasi tinggi. Dataset ini dirancang agar model YOLOv8 dapat mengenali dan membedakan bola berdasarkan warna secara *real-time*.

Setelah proses pelabelan selesai, dataset dibagi menjadi tiga bagian: data train (70%), validasi (20%), dan test (10%). Setiap *folder* utama ("train", "val", dan "test") memiliki *subfolder* "images" yang berisi *file* gambar dan "labels" yang berisi *file* anotasi yang sesuai seperti pada **Error! Reference source not found.** Struktur ini memastikan setiap gambar memiliki label yang terhubung langsung, sehingga dataset siap digunakan untuk proses pelatihan model YOLOv8 tanpa memerlukan langkah tambahan dalam pengorganisasian data.

```

  ▼ DATASETBOLA3WARNA
    ▼ test
      > images
      > labels
    ▼ train
      > images
      > labels
    ▼ val
      > images
      > labels
    ! data.yaml

```

Gambar 3.2 Struktur *folder* persiapan pelatihan model

3.3 Pelatihan Model Deteksi Bola

Dataset yang telah diproses dan diberi label digunakan untuk melatih model YOLOv8 guna mendeteksi bola secara *real-time*. Model ini menggunakan basis pretrained untuk meningkatkan kecepatan dan efisiensi dalam mendeteksi objek dalam gim. Pelatihan dilakukan dengan menyetel parameter seperti *learning rate*, *momentum*, dan *weight decay* untuk mengoptimalkan performa model. Selain itu, teknik *data augmentation* seperti *translasi*, skala, dan *flipping horizontal* diterapkan untuk meningkatkan keberagaman dataset dan membantu

model mengenali objek dalam berbagai kondisi. Selama pelatihan, model divalidasi menggunakan *threshold IoU* untuk memastikan akurasi deteksi. Evaluasi dilakukan dengan mengukur *precision* dan *recall*, serta menggunakan *confusion matrix* untuk menilai kinerja deteksi secara keseluruhan. Rencana pelatihan ini bertujuan untuk menghasilkan model yang dapat mendeteksi bola dengan akurasi tinggi dalam berbagai kondisi permainan.

3.4 Pengembangan Algoritma Deteksi Hit

Algoritma deteksi *hit* dirancang untuk menentukan apakah bola yang dilempar pemain mengenai target di layar. Proses deteksi *hit* melibatkan beberapa langkah matematika, termasuk perhitungan radius bola, kalibrasi manual untuk rentang radius, serta pengecekan apakah bola menyentuh target.

Perhitungan Radius Bola: Radius bola akan dihitung berdasarkan dimensi *bounding box* yang mengelilingi bola yang terdeteksi. Dengan asumsi bahwa bola memiliki bentuk lingkaran, radius bola dapat dihitung sebagai setengah dari dimensi terkecil antara lebar (*width*) dan tinggi (*height*) *bounding box* yang mengelilinginya. Di mana w adalah lebar *bounding box* dan h adalah tinggi *bounding box*. Rumus ini digunakan karena pada umumnya objek yang terdeteksi berbentuk persegi panjang, dan kita mengasumsikan bola berada dalam kotak tersebut. Menggunakan ukuran terkecil sebagai diameter memungkinkan kita untuk memperkirakan radius bola yang sesuai. Rumus (3.1) matematikanya adalah:

$$r_{bola} = \frac{1}{2} \cdot \min(w, h) \quad (3.2)$$

Kalibrasi Manual Rentang Radius: Sebelum memulai deteksi *hit*, dilakukan kalibrasi manual untuk menentukan rentang radius yang valid. Proses kalibrasi ini melibatkan penempelan bola pada tembok untuk mengukur radius yang terdeteksi, yang nantinya digunakan untuk menentukan rentang radius *minimum* dan *maksimum* yang valid untuk deteksi. Rentang ini memungkinkan kita untuk memfilter deteksi bola yang berada dalam ukuran yang sah dan dapat dianggap sebagai objek yang valid. Kalibrasi manual ini penting untuk memastikan bahwa bola yang terdeteksi sesuai dengan ukuran yang realistis dan berada dalam batas ukuran yang sah. Rentang radius ini dapat digambarkan dengan rumus (3.3):

$$r_{min} \leq r_{bola} \leq r_{max} \quad (3.4)$$

Perhitungan Jarak Bola ke Target: Setelah radius bola terdeteksi dan divalidasi, langkah selanjutnya adalah mengecek apakah bola mengenai target dengan menghitung jarak antara titik tengah bola dan titik tengah target. Jarak ini dihitung dengan menggunakan rumus (3.5) jarak Euclidean, yang memberikan jarak antara dua titik dalam ruang dua dimensi di

mana (x_1, y_1) adalah koordinat titik tengah bola dan (x_2, y_2) adalah koordinat titik tengah target. Jarak ini digunakan untuk menentukan apakah bola cukup dekat untuk dianggap mengenai target.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3.6)$$

Pengecekan Apakah Bola Menyentuh Target: Untuk memverifikasi apakah bola benar-benar menyentuh target, kita membandingkan jarak yang telah dihitung dengan jumlah radius bola dan radius target. Bola dianggap mengenai target jika jarak antara bola dan target lebih kecil atau sama dengan jumlah kedua radius tersebut, yang dapat dituliskan (3.7) sebagai:

$$d \leq r_{bola} + r_{target} \quad (3.8)$$

Jika kondisi ini terpenuhi, maka bola dianggap menyentuh target dan perhitungan skor dilakukan. Jika tidak, bola dianggap *miss* dan tidak menyentuh target.

3.5 Implementasi Algoritma Deteksi Bola dan *Hit* pada Gim

Gim lempar bola ini dimainkan oleh hingga tiga pemain secara bersamaan, dengan masing-masing pemain memiliki daerah permainan yang berbeda, yang ditandai dengan warna bola yang akan dimainkan oleh masing-masing pemain. Setiap pemain akan melempar bola ke arah target yang muncul di layar, dan algoritma deteksi bola serta deteksi *hit* akan digunakan untuk memverifikasi apakah bola yang dilempar mengenai target atau tidak.

Selain memverifikasi apakah bola mengenai target, algoritma juga memeriksa kecocokan warna bola dengan warna target. Hal ini dilakukan untuk memastikan bahwa hanya bola dengan warna yang sesuai dengan target yang dihitung sebagai *hit*. Misalnya, bola berwarna merah hanya akan dihitung sebagai *hit* jika mengenai target merah, bola kuning hanya bisa mengenai target kuning, dan seterusnya. Proses verifikasi ini penting untuk memastikan integritas permainan, di mana hanya bola yang sesuai dengan warna target yang dianggap berhasil mengenai target.

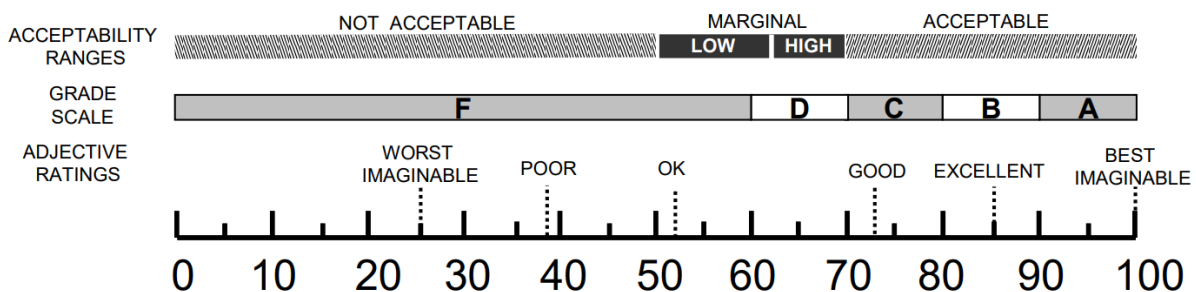
Seluruh kontrol permainan sepenuhnya bergantung pada deteksi bola dan *hit*, di mana sistem akan mendeteksi bola yang dilempar pemain dengan model terbaik hasil pelatihan dan memverifikasi apakah bola tersebut mengenai target berdasarkan algoritma deteksi *hit* yang telah diterapkan. Proses ini memastikan bahwa interaksi pemain dengan gim bersifat interaktif, responsif, dan mengandalkan deteksi visual untuk menentukan hasil dari setiap aksi pemain.

Dengan pengimplementasian algoritma deteksi bola dan *hit* ini, gim tersebut dapat digunakan sebagai platform untuk menguji akurasi dan efektivitas algoritma dalam mendeteksi

interaksi bola dan target, serta memberikan pengalaman bermain yang menyenangkan dan kompetitif bagi pemain.

3.6 Pengujian dan Evaluasi Implementasi

Setelah implementasi selesai, pengujian dilakukan untuk mengevaluasi kinerja sistem deteksi bola dan *hit*. Pengujian ini dilakukan dalam lingkungan yang terkontrol untuk memastikan bahwa sistem dapat mendeteksi bola dengan akurasi tinggi dalam berbagai kondisi permainan. Evaluasi dilakukan untuk mengukur kecepatan inferensi, akurasi deteksi, dan pengalaman pengguna. Selain itu, dilakukan juga evaluasi terhadap responsivitas permainan untuk memastikan bahwa pemain dapat merasakan pengalaman bermain yang lancar dan interaktif.



Gambar 3.3 *System Usability Scale (SUS) Score*

Sumber: (Bangor et al., 2009)

Untuk memastikan hasil yang lebih representatif, pengujian ini akan melibatkan 5 pemain yang akan berpartisipasi dalam serangkaian skenario permainan yang mencerminkan berbagai kondisi pencahayaan dan variasi gerakan bola. Para pemain ini akan mencoba sistem dalam beberapa sesi, dan hasil deteksi serta pengalaman mereka akan dicatat untuk dianalisis lebih lanjut. Untuk mengukur tingkat kepuasan dan kegunaan sistem, pengujian ini akan menggunakan System Usability Scale (SUS) sebagai metode evaluasi kuesioner (Bangor et al., 2009). SUS akan digunakan untuk mendapatkan skor usability dari para pemain setelah mereka mencoba sistem. Skor SUS dihitung berdasarkan skala 0 hingga 100 seperti pada Gambar 3.3, lalu disederhanakan menjadi 0-10 dengan interpretasi sesuai Tabel 3.1 berikut:

Tabel 3.1 *System Usability Scale (SUS) Adjective Score*

SUS Score	Kategori	Interpretasi
0 - 2,5	Worst Imaginable	Sangat tidak dapat digunakan, pengalaman pengguna sangat buruk.
2,6 - 3,9	Poor	Sistem sulit digunakan, membutuhkan perbaikan besar.
4 - 5,2	OK (Marginal – Low)	Bisa digunakan, tetapi masih memiliki banyak kekurangan.
5,3 - 7,3	OK (Marginal – High)	Masih dalam kategori rata-rata, tetapi pengalaman pengguna perlu ditingkatkan.
7,4 - 8,5	Good	Sistem nyaman digunakan, hanya membutuhkan sedikit peningkatan.
8,6 - 9	Excellent	Sangat mudah digunakan, pengalaman pengguna sangat baik.
9,1 - 10	Best Imaginable	Pengalaman terbaik yang dapat dibayangkan, sistem hampir sempurna dalam usability.

Dengan menggunakan metode SUS ini, hasil evaluasi dari pemain dapat membantu mengidentifikasi area yang perlu diperbaiki, serta memberikan gambaran mengenai tingkat kepuasan pengguna terhadap sistem yang telah dikembangkan.

BAB IV

HASIL DAN PEMBAHASAN

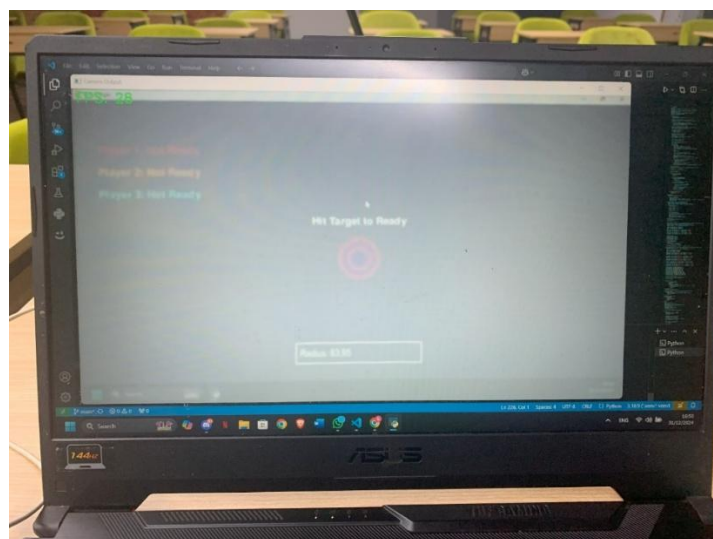
Dalam pengembangan ini akan digunakan beberapa library, software dan hardware sebagai berikut. Dari sisi permainan, *platform* pengembangan akan menggunakan Pygame versi 2.6.0. Pygame dipilih karena akan mempermudah integrasi dengan YOLOv8 karena keduanya dapat berjalan dalam satu lingkungan pengembangan Python versi 3.10.9 yang seragam. Pelabelan akan menggunakan *software* LabelImg v1.8.1. Untuk Computer Vision, algoritma library OpenCV versi 4.9.0 digunakan. Sementara itu YOLOv8 dipilih sebagai model deteksi utama dengan menggunakan yolov8n.pt sebagai model *pre-trained*. Google Colab digunakan untuk melatih model YOLOv8 dengan memanfaatkan GPU Tesla T4 yang tersedia dan melibatkan 8 pekerja paralel untuk mempercepat proses pelatihan.



(a)



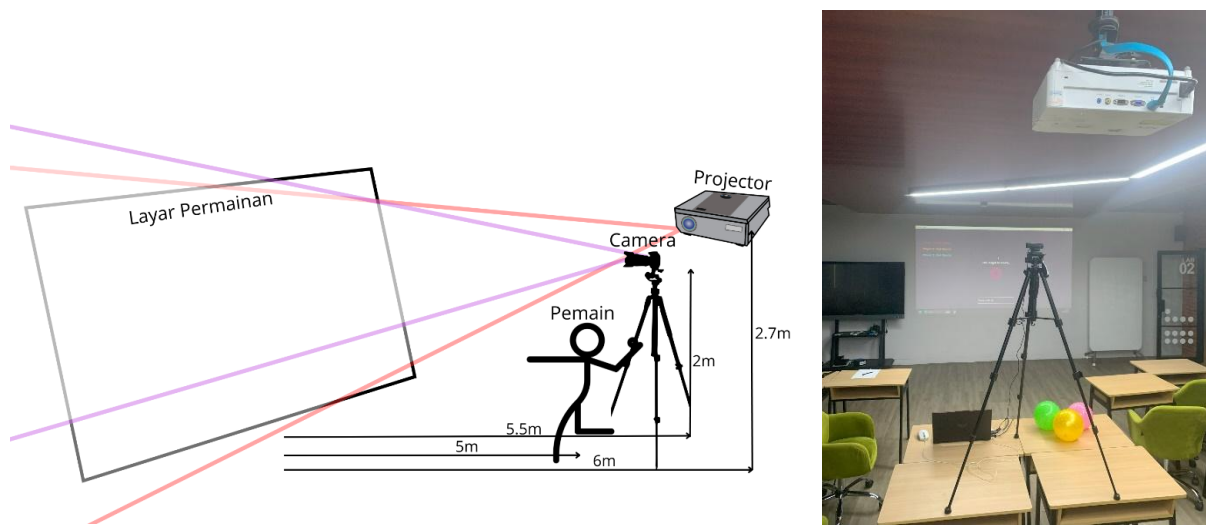
(b)



(c)

Gambar 4.1 (a) Kamera, (b) proyektor, dan (c) laptop

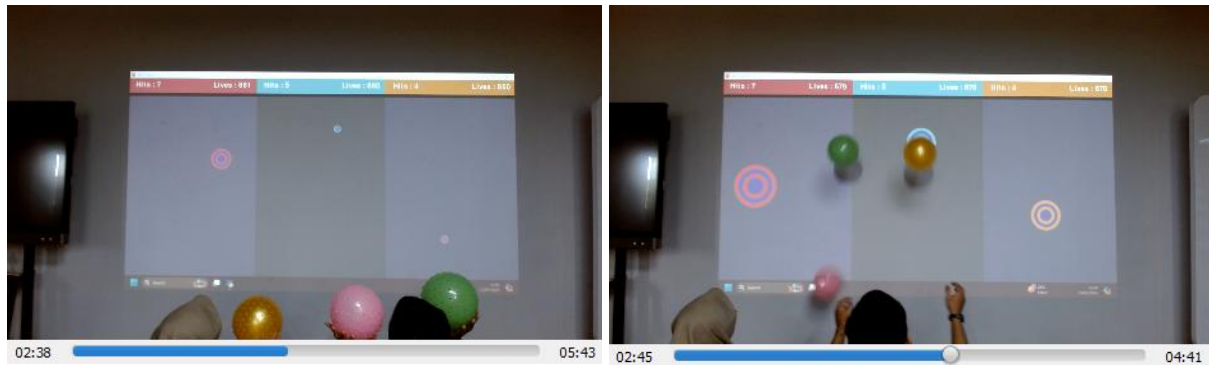
Perangkat keras utama yang digunakan dalam eksperimen ini meliputi kamera Logitech C920 HD Pro dengan resolusi 1080p pada 30 FPS, proyektor NEC VE303X *standard-throw* dengan kecerahan 3000 lumens, serta laptop ASUS TUF dengan prosesor Intel Core i7 generasi ke-11, RAM 16GB, GPU NVIDIA GeForce GTX 1650 yang dapat dilihat pada Gambar 4.1 diatas. Untuk memastikan deteksi objek yang akurat, digunakan lingkungan yang terkontrol dengan kamera ditempatkan sejauh 5meter dari area target, dengan pencahayaan seragam menggunakan dua lampu LED 5000K yang menghasilkan intensitas 500 lux. Eksperimen dilakukan di ruangan dengan latar belakang putih dan area bermain seluas 3m x 6m untuk meminimalkan gangguan visual.



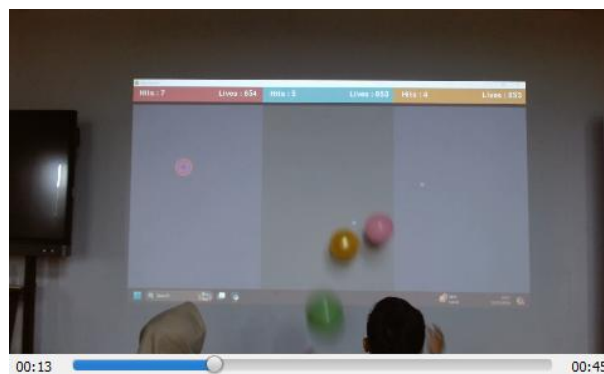
Gambar 4.2 Diagram penempatan perangkat keras

Pada lingkungan ideal yang terkontrol, penempatan kamera, proyektor, dan layar dirancang sesuai dengan konfigurasi optimal yang ditunjukkan pada Gambar 4.2. Seluruh pengujian dilakukan menggunakan skema ini untuk memastikan akurasi deteksi dan konsistensi hasil. Namun, untuk memudahkan replikasi, konfigurasi ini hanya akan dijadikan sebagai referensi utama. Penempatan perangkat memiliki peran yang sangat penting dalam mendukung performa sistem. Kamera ditempatkan pada posisi yang memberikan sudut pandang optimal untuk menangkap bola saat dilempar, dengan cakupan penuh seluruh area permainan pada layar. Sementara itu, proyektor diposisikan sedemikian rupa agar target digital dapat ditampilkan dengan jelas di layar, tanpa distorsi visual yang dapat mengganggu proses deteksi.

4.1 Pengumpulan Data



(a)

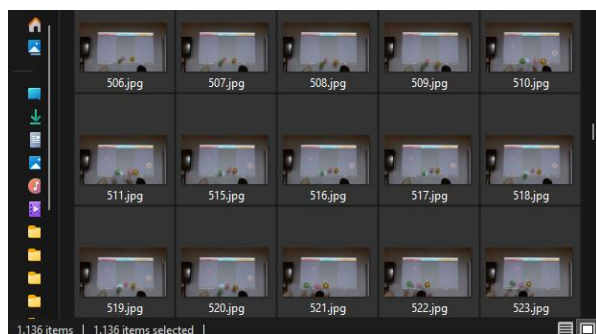


Gambar 4.3 (a) Dua video awal simulasi permainan, (b) Video final setelah pemotongan

Pada tahap ini dilakukan pengumpulan video simulasi permainan sebagai sumber data untuk melatih sistem deteksi bola menggunakan teknologi YOLOv8. Dua video awal yang diperoleh seperti pada Gambar 4.3 (a) memiliki durasi masing-masing 5 menit 43 detik dan 4 menit 41 detik. Sebelum proses ekstraksi *frame* dilakukan, video tersebut melalui tahap pemotongan konten, yaitu menghilangkan seluruh bagian video yang tidak relevan, khususnya bagian yang tidak menampilkan bola yang dilemparkan ke arah layar. Langkah ini bertujuan untuk memperbaiki efisiensi dan fokus data yang digunakan untuk pelatihan.



(a)



(b)

Gambar 4.4 Hasil ekstraksi gambar (a) 1368 gambar (b) 1136 gambar

Setelah pemotongan, dihasilkan satu video final dengan durasi 45 detik seperti pada Gambar 4.3 (b), di mana seluruh *framenya* menampilkan aktivitas lemparan bola ke arah layar. Video dengan kecepatan 30 *frame per second* (FPS) ini seharusnya menghasilkan 1350 gambar. Namun, selama proses ekstraksi gambar yang menggunakan kode Python sederhana seperti Gambar 4.5, beberapa gambar tambahan terdeteksi akibat gangguan minor dalam proses ekstraksi. Akibatnya, jumlah total gambar yang diekstrak adalah 1368 gambar yang dapat dilihat pada Gambar 4.4 (a), yang sedikit lebih banyak dari perhitungan awal. Perbedaan ini tidak berdampak negatif pada hasil karena setiap gambar tetap relevan dan berisi informasi tentang bola yang dilemparkan.

```

1 import cv2
2 cap = cv2.VideoCapture('BolaPolos.mp4')
3
4 count = 0
5
6 while True:
7     success, img = cap.read()
8     cv2.imwrite(f'Bola/{count}.jpg',img)
9     count +=1
10    cv2.imshow("Original Image ", img)
11    cv2.waitKey(1)

```

Gambar 4.5 Kode ekstraksi gambar dari tiap *frame* video

Setelah 1368 gambar berhasil diekstraksi dari video, proses selanjutnya adalah pembersihan data, yaitu menyaring dataset untuk memastikan bahwa hanya gambar yang memenuhi kriteria yang digunakan untuk pelatihan model YOLOv8. Dalam tahap ini, gambar-gambar dengan potensi masalah seperti duplikasi, bola yang buram atau terpotong, serta gambar di mana bola tidak sepenuhnya terlihat, dihapus dari dataset. Dari proses ini, dataset yang awalnya berjumlah 1368 gambar berkurang menjadi 1136 gambar yang dapat dilihat pada Gambar 4.4 (b).

4.2 Pelabelan Data

Tahap berikutnya adalah *labeling* data, di mana setiap gambar diberikan label manual menggunakan tool LabelImg. Pada setiap gambar, *bounding box* (kotak pembatas) digambar di sekitar bola yang terlihat, dan label diberikan berdasarkan warna bola: *pinkball* untuk bola merah muda, *greenball* untuk bola hijau, dan *yellowball* untuk bola kuning. Informasi ini disimpan dalam *file* teks .txt, dengan nama *file* yang sama seperti gambar yang dilabeli, dan mengikuti format standar YOLO, yaitu koordinat tengah, lebar, dan tinggi dari *bounding box* seperti pada Gambar 4.6.

0	0.252083	0.206019	0.052083	0.091667
2	0.330208	0.243519	0.045833	0.088889
1	0.394792	0.213889	0.057292	0.101852

Gambar 4.6 Isi file text.txt hasil pelabelan

Setelah semua gambar dilabeli, dataset disusun sesuai struktur yang diperlukan untuk pelatihan model YOLOv8. Gambar dan *file* label dibagi menjadi tiga folder utama:

- Train (796 gambar):** Folder ini digunakan untuk melatih model agar dapat mengenali pola dari data.
- Val (227 gambar):** Folder ini digunakan untuk memvalidasi performa model selama pelatihan, membantu mencegah *overfitting*.
- Test (113 gambar):** Folder ini digunakan untuk menguji kemampuan model pada data baru yang belum pernah dilihat sebelumnya.

Setiap folder memiliki sub-folder *images* (berisi gambar) dan *labels* (berisi *file* teks label untuk setiap gambar). Pembagian dataset ini dirancang untuk memastikan pelatihan model berlangsung optimal, dengan model yang tidak hanya mampu mengenali pola dari data pelatihan tetapi juga dapat divalidasi dan diuji pada data yang berbeda.

4.3 Pelatihan Model Deteksi Bola

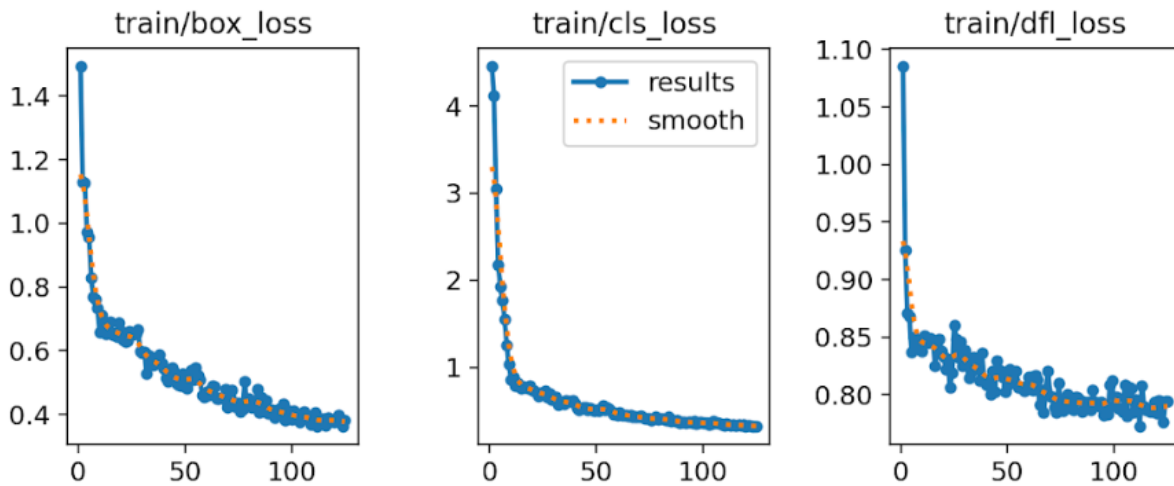
Setelah melalui beberapa iterasi pelatihan, konfigurasi optimal untuk model YOLOv8 berhasil ditentukan. Proses ini melibatkan eksperimen dengan berbagai *hyperparameter* pelatihan seperti *learning rate*, *batch size*, jumlah *epoch*, dan *augmentasi* data, hingga diperoleh kombinasi yang menghasilkan performa terbaik.

Model dilatih selama 150 epoch dengan *Learning rate* optimal yang digunakan adalah 0.001, memberikan keseimbangan antara kecepatan konvergensi dan stabilitas pelatihan. Nilai ini memastikan model belajar secara bertahap tanpa melompat terlalu jauh dalam pembaruan bobot. *Batch size* ditetapkan pada 16, yang memberikan efisiensi dalam pemrosesan data tanpa membebani memori GPU, menjaga kecepatan pelatihan tetap stabil.

Augmentasi data yang diterapkan meliputi rotasi hingga 15 derajat, *scaling* untuk mensimulasikan bola pada jarak yang berbeda, *flipping* horizontal untuk variasi arah, serta penyesuaian *brightness* dan *contrast* untuk menangani kondisi pencahayaan yang beragam. Augmentasi ini memastikan model mampu mengenali bola dalam berbagai situasi.

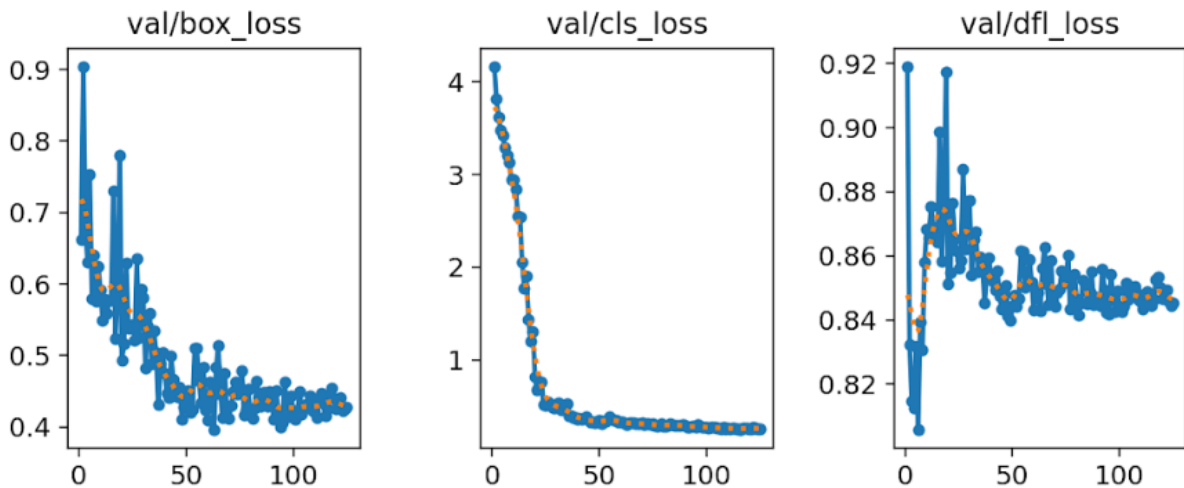
Arsitektur yang digunakan adalah YOLOv8n (*Nano*), varian YOLOv8 yang ringan dan dirancang untuk efisiensi. Model ini dipilih karena mampu memberikan kecepatan inferensi

tinggi, menjadikannya ideal untuk aplikasi *real-time* tanpa mengorbankan akurasi secara signifikan. *Optimizer* Adam digunakan karena stabilitasnya dalam memperbarui bobot, yang mempercepat proses konvergensi dibandingkan metode seperti SGD.



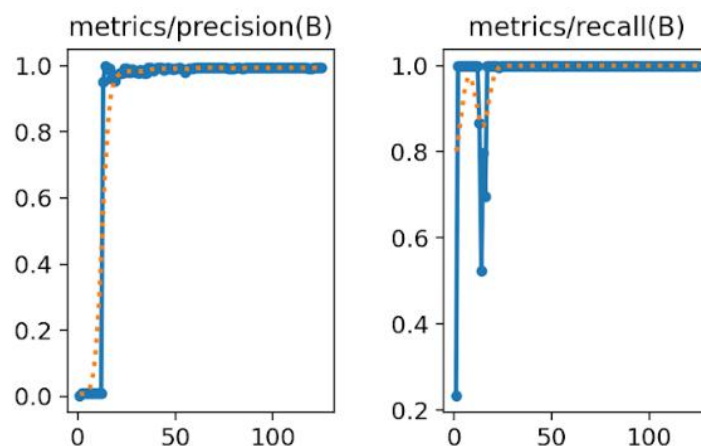
Gambar 4.7 Hasil evaluasi *train loss*

Pada pelatihan model, terdapat penurunan yang signifikan pada *train loss* di setiap *epoch* yang dapat dilihat pada Gambar 4.7. Nilai *train/box_loss*, yang mengukur kesalahan dalam prediksi posisi objek, menunjukkan penurunan yang stabil, dari 1.49470 pada *epoch* pertama menjadi 0.59371 pada *epoch* 30. Ini menunjukkan bahwa model semakin baik dalam memprediksi posisi objek seiring waktu. Begitu pula dengan *train/cls_loss* (kesalahan dalam klasifikasi objek), yang mengalami penurunan dari 4.4611 menjadi 0.65012 pada *epoch* 30, yang berarti model semakin baik dalam mengenali dan mengklasifikasikan objek. *Train/dfl_loss*, yang mengukur kesalahan dalam distribusi kelas objek, juga menurun dari 1.08560 menjadi 0.83815 pada *epoch* 30, menunjukkan bahwa model semakin tepat dalam mendeteksi distribusi probabilitas untuk kelas objek.



Gambar 4.8 Hasil evaluasi *validation loss*

Pada *validation loss*, yang mengukur kinerja model pada data yang tidak terlihat selama pelatihan, juga menunjukkan penurunan meskipun ada beberapa fluktuasi seperti pada Gambar 4.8. *Val/box_loss* menurun dari 0.66234 pada *epoch* pertama menjadi 0.58105 pada *epoch* 30. Ini menunjukkan bahwa model tidak hanya mengurangi kesalahan pada data pelatihan tetapi juga dapat menggeneralisasi dengan baik pada data yang belum pernah dilihat. Demikian juga dengan *val/cls_loss*, yang menunjukkan penurunan dari 4.1617 menjadi 0.50483 pada *epoch* 30, dan *val/dfl_loss* yang juga menurun dari 0.91898 menjadi 0.87735 pada *epoch* 30, memberikan bukti bahwa model terus meningkatkan akurasi pada data validasi.

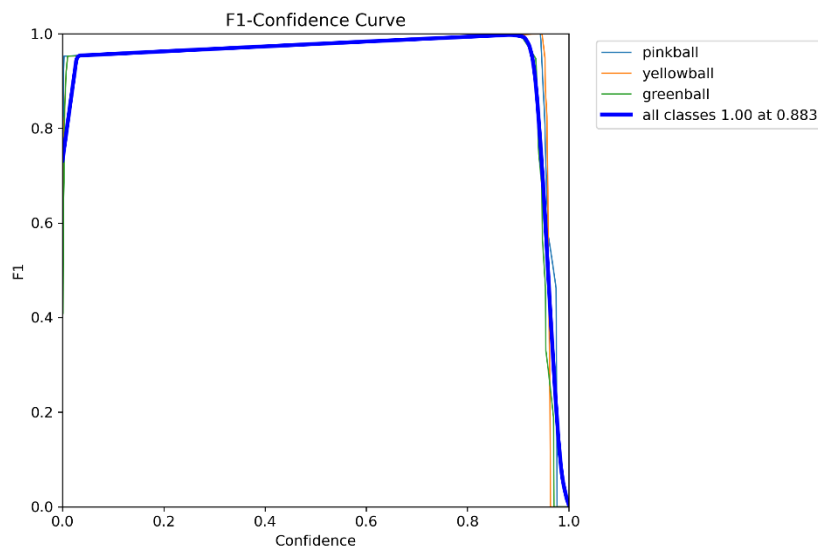


Gambar 4.9 Hasil evaluasi metrik *precision-recall*

Precision pada *epoch* 30 untuk kelas *pinkball*, *yellowball*, dan *greenball* masing-masing sangat tinggi yang dapat dilihat pada Gambar 4.9. *Precision* untuk *pinkball* mencapai 0.98069, untuk *yellowball* adalah 1.0, dan untuk *greenball* juga 1.0. Hal ini menunjukkan bahwa model

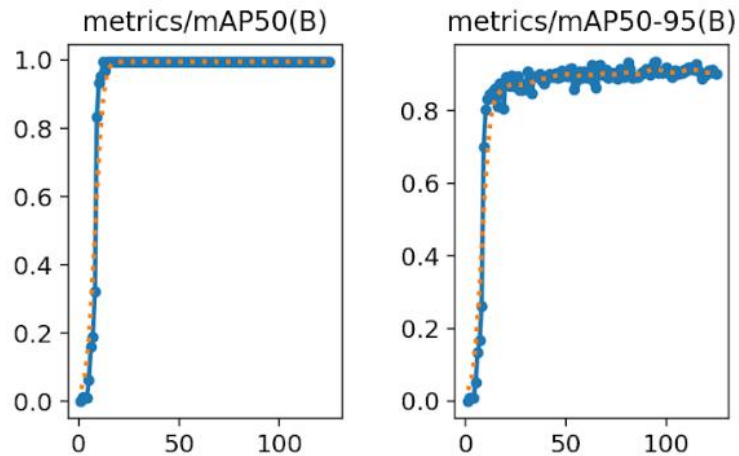
sangat tepat dalam memprediksi objek yang relevan tanpa banyak menghasilkan *false positives* (prediksi yang salah). *Recall* juga menunjukkan hasil yang sangat baik seperti pada Gambar 4.9, dengan nilai 1.0 untuk ketiga kelas tersebut, yaitu *pinkball*, *yellowball*, dan *greenball*. Ini berarti bahwa model berhasil mendeteksi hampir semua objek positif yang relevan tanpa banyak *false negatives* (objek positif yang terlewatkan).

Pada *epoch* 1, model menunjukkan nilai *F1 score* yang rendah, dengan *pinkball* mencapai 0.003474, *yellowball* 0.020410, dan *greenball* 0.019802. Namun, setelah dilatih hingga *epoch* 30, nilai *F1 score* mengalami peningkatan signifikan. Pada *epoch* 30, *pinkball* mencapai 0.7826, *yellowball* 0.8631, dan *greenball* 0.8294. Peningkatan ini menunjukkan bahwa model berhasil memperbaiki kemampuannya dalam mendeteksi bola, dengan meningkatkan keseimbangan antara *precision* dan *recall* pada setiap kelas bola.



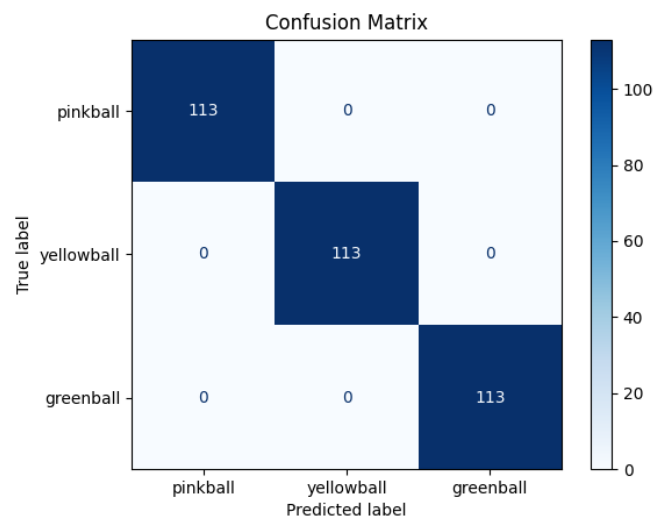
Gambar 4.10 Hasil evaluasi metrik F1 *Confidence Curve*

Selain itu, *F1-curve* yang ditampilkan pada Gambar 4.10 menunjukkan hubungan antara tingkat *confidence* model dengan nilai *F1 score* untuk setiap kelas bola. Pada *F1-curve*, dapat dilihat bahwa ketika *confidence* model meningkat, nilai *F1 score* juga meningkat secara signifikan. Hal ini menunjukkan bahwa model memberikan prediksi yang lebih akurat pada tingkat *confidence* yang lebih tinggi, dengan *F1 score* yang semakin mendekati nilai 1.00. Kurva ini juga memperlihatkan bahwa model cenderung lebih baik dalam mendeteksi bola ketika *confidence* lebih tinggi, yang semakin meningkatkan performa deteksi secara keseluruhan.



Gambar 4.11 Hasil evaluasi metrik mAP50 dan mAP50-95

Pada Gambar 4.11 dapat terlihat bahwa mAP50 pada *epoch* 30 untuk kelas *pinkball*, *yellowball*, dan *greenball* mencapai 0.995 untuk *pinkball*, 1.0 untuk *yellowball*, dan 1.0 untuk *greenball*, yang menandakan bahwa model memiliki tingkat akurasi yang sangat tinggi dalam mendeteksi objek pada ambang batas IoU 50% (tumpang tindih kotak pembatas yang rendah). mAP50-95, yang mengukur kinerja model pada berbagai ambang batas IoU dari 50% hingga 95%, juga menunjukkan hasil yang baik, dengan 0.874 untuk *pinkball*, 0.900 untuk *yellowball*, dan 0.887 untuk *greenball*. Ini menunjukkan bahwa meskipun ambang batas IoU yang lebih ketat diterapkan, model tetap mampu mendeteksi objek dengan akurasi yang tinggi.



Gambar 4.12 Hasil evaluasi metrik *confusion matrix*

True Positives (TP): Pada *confusion matrix*, terlihat bahwa model berhasil mengklasifikasikan dengan benar 113 objek *pinkball* sebagai *pinkball*, 113 objek *yellowball*

sebagai *yellowball*, dan 113 objek *greenball* sebagai *greenball*. *False Positives* (FP) dan *False Negatives* (FN): Tidak ada kesalahan klasifikasi dalam *confusion matrix* ini seperti yang terlihat pada Gambar 4.12, karena semua prediksi model tepat, dengan tidak ada objek yang salah diklasifikasikan sebagai kelas lain.

4.4 Pengembangan Algoritma Deteksi *Hit*

Pada permainan ini, deteksi *hit* (kena target) awalnya menggunakan kontrol sementara berupa klik mouse untuk memudahkan pengujian sistem sebelum implementasi penuh dengan model YOLO. Saat pemain mengklik layar, sistem memeriksa apakah koordinat klik mouse berada dalam jangkauan target yang ada. Deteksi ini dilakukan dengan menghitung jarak antara posisi klik dan pusat target menggunakan rumus Euclidean. Jika jarak tersebut lebih kecil atau sama dengan ukuran target, klik dianggap berhasil mengenai target (*hit*), dan target akan dihapus dari layar.

Selain itu, setiap kali target terkena, skor pemain akan bertambah. Skor ini dicatat dalam variabel yang mewakili jumlah target yang berhasil dipukul. Informasi skor dan jumlah nyawa pemain yang tersisa ditampilkan secara *real-time* di layar atas menggunakan fungsi `draw_top_bar`, memberikan umpan balik langsung kepada pemain.

Meskipun deteksi ini saat ini bergantung pada klik mouse, sistem telah dirancang untuk mendukung integrasi penuh dengan model YOLO. Ketika model YOLO diimplementasikan, deteksi tidak lagi bergantung pada klik mouse, melainkan menggunakan *bounding box* yang dihasilkan oleh YOLO untuk mendeteksi bola secara *real-time*. Pusat bola yang terdeteksi oleh YOLO akan digunakan untuk memeriksa apakah bola berada dalam jangkauan target. Logika ini memungkinkan permainan untuk menangkap interaksi pemain yang lebih realistis dengan bola fisik.

Perhitungan skor dirancang serupa untuk kedua mode. Setiap kali target terkena, baik dengan klik mouse maupun bola yang terdeteksi oleh YOLO, skor pemain bertambah satu. Sebaliknya, jika target mengecil hingga menghilang tanpa terkena, hal ini dianggap sebagai kesalahan, dan nyawa pemain akan berkurang. Dengan demikian, permainan ini memberikan tantangan kepada pemain untuk mengenai target sebelum target menghilang, sambil mempersiapkan transisi yang mulus ke deteksi berbasis YOLO untuk pengalaman yang lebih interaktif dan dinamis.

Fitur utama dalam Gim ini, yaitu deteksi *hit* dan deteksi objek, menggunakan logika berbasis algoritma YOLO (*You Only Look Once*) untuk mendeteksi bola secara *real-time* dan memastikan pengalaman bermain yang *imersif*. Deteksi objek dilakukan dengan mengolah

frame video dari kamera yang kemudian diproses menggunakan algoritma YOLOv8. Kamera menangkap posisi bola sebagai objek utama, menghasilkan koordinat pusat (x , y), dimensi ($width$, $height$), radius objek, dan label warna bola yang sesuai dengan pemain. Karena sudah dikalibrasi di awal maka koordinat dan dimensi ini juga akan muncul pada daerah yang sama pada layar permainan. Radius bola dihitung seperti pada Gambar 4.13:

```
radius = min(width, height) / 2
```

Gambar 4.13 Kode Perhitungan Radius

Hanya objek yang memenuhi syarat radius yang telah dikalibrasi yang akan dijelaskan lebih lanjut pada sub bab selanjutnya. Dan memiliki tingkat kepercayaan deteksi di atas ambang batas dalam hal ini adalah 0.7 yang akan diproses lebih lanjut.

Selanjutnya, untuk mendeteksi *hit*, sistem membandingkan posisi bola dengan posisi target pada layar permainan. Setiap target memiliki koordinat pusat (x_t , y_t) dan radius r_t , yang terus diperbarui secara dinamis karena target akan membesar dan mengecil. Ketika bola mendekati target, jarak antara pusat bola (x_b , y_b) dan pusat target dihitung menggunakan rumus jarak Euclidean seperti pada Gambar 4.14

```
dis = math.sqrt((self.x - x)**2 + (self.y - y)**2)
```

Gambar 4.14 Kode Perhitungan Euclidean

Jika jarak ini lebih kecil atau sama dengan jumlah radius target dan bola ($r_t + r_b$) seperti pada Gambar 4.15, sistem mencatat bahwa bola berhasil mengenai target (*hit*). Selain itu, radius bola juga divalidasi agar sesuai dengan rentang yang telah dikalibrasi sebelumnya.

```
if dis <= self.size + radius:
    targets.remove(target)
    return True
```

Gambar 4.15 Kode Pengecekan Apakah Target dan Bola Bersentuhan

4.5 Implementasi Algoritma Deteksi Bola dan *Hit* pada Gim

4.5.1 Implementasi Deteksi Bola dan *Hit*

Proses implementasi model YOLOv8 ke dalam Gim dimulai dengan memuat model yang telah dilatih sebelumnya melalui file `bolapolos_best.pt` pada Gambar 4.16 baris ke 14. Model ini dirancang khusus untuk mendeteksi tiga jenis bola berdasarkan warna—*pinkball*, *yellowball*, dan *greenball*. Kamera diatur untuk menangkap input video secara *real-time* dengan resolusi 1280x720 piksel dan kecepatan 30 FPS, menghasilkan data visual yang kemudian diproses untuk mendeteksi keberadaan bola di area permainan. Sebelum digunakan,

kamera melalui tahap kalibrasi menggunakan matriks transformasi perspektif untuk memastikan koordinat yang dihasilkan sejajar dengan layar permainan dengan file `calibration.p` pada Gambar 4.16 baris ke 5. Kalibrasi ini penting untuk mencegah kesalahan deteksi dan menjamin akurasi tinggi.

```

1 pygame.init()
2 WIDTH, HEIGHT = 1280, 720
3
4 # Load calibration file for warp perspective
5 fileObj = open('calibration.p', 'rb')
6 circles = pickle.load(fileObj)
7 fileObj.close()
8
9 # Define perspective transform points
10pts1 = np.float32([circles[0], circles[1], circles[2], circles[3]])
11pts2 = np.float32([[0, 0], [WIDTH, 0], [0, HEIGHT], [WIDTH, HEIGHT]])
12matrix = cv2.getPerspectiveTransform(pts1, pts2)
13
14model = YOLO("bolapolos_best.pt")
15cap = cv2.VideoCapture("BolaPolos.mp4")
16cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
17cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

```

Gambar 4.16 Inisiasi model dan kalibrasi kamera

Setiap *frame* video yang ditangkap oleh kamera diproses oleh YOLOv8 untuk mendeteksi objek bola. Model ini memindai gambar dan menghasilkan output berupa *bounding box* yang mencakup informasi posisi bola, label kelas berdasarkan warna, dan tingkat kepercayaan (*confidence score*). Untuk memastikan hanya deteksi valid yang diproses lebih lanjut, bola yang terdeteksi harus memiliki tingkat kepercayaan minimal 75%. Radius bola dihitung berdasarkan dimensi *bounding box* menggunakan rumus geometri dan divalidasi terhadap rentang radius yang telah dikalibrasi sebelumnya. Hanya bola dengan radius yang berada dalam rentang valid yang dipertimbangkan sebagai bagian dari mekanisme permainan. Hal ini lebih jelasnya dapat dilihat pada Gambar 4.17.

```

1 def main():
2     run = True
3     targets1, targets2, targets3 = [], [], []
4     detected_circles = []
5     target1_pressed, target2_pressed, target3_pressed = 0, 0, 0
6     clicks = [0, 0, 0]
7     misses1, misses2, misses3 = 0, 0, 0
8     start_time = time.time()
9     hit_timestamps = {0: [], 1: [], 2: []}
10    last_detected = {0: 0, 1: 0, 2: 0}
11    clock = pygame.time.Clock()
12
13    pygame.time.set_timer(TARGET_EVENT, TARGET_INCREMENT)
14
15    while run:

```

```

16         clock.tick(60)
17         elapsed_time = time.time() - start_time
18         success, img = cap.read()
19         if not success:
20             break
21
22         scale_factor = 3
23         output_width = WIDTH // scale_factor
24         output_height = HEIGHT // scale_factor
25
26         img = warpImage(img, circles, output_width, output_height)
27         img = cv2.resize(img, (WIDTH, HEIGHT))
28         results = model(img, stream=True)
29
30         for r in results:
31             boxes = r.boxes
32             for box in boxes:
33                 x1, y1, x2, y2 = box.xyxy[0]
34                 x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
35                 center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
36                 cls = int(box.cls[0])
37                 conf = float(box.conf[0])
38                 width = x2 - x1
39                 height = y2 - y1
40
41                 radius = math.sqrt((width**2 + height**2) / 2)
42
43                 if conf > 0.75:
44                     if MIN_RADIUS <= radius <= MAX_RADIUS:
45                         current_time = time.time()
46
47                         scaled_radius = int(radius * (WIDTH /
48 cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
49                         scaled_x = int(center_x * (WIDTH /
50 cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
51                         scaled_y = int(center_y * (HEIGHT /
52 cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
53
54                         if current_time - last_detected[cls] >=
DETECTION_DELAY:
55                             detected_circles.append((scaled_x,
56 scaled_y, scaled_radius, cls, conf))
57                             last_detected[cls] = current_time
58
59                             if cls == 0:
60                                 clicks[cls] += 1
61                                 if check_collision(targets1, (x1 + x2)
62 // 2, (y1 + y2) // 2, radius):
63                                     target1_pressed += 1
64                             elif cls == 1:
65                                 clicks[cls] += 1
66
67                             hit_timestamps[cls].append(current_time)
68                             if check_collision(targets2, (x1 + x2)
69 // 2, (y1 + y2) // 2, radius):
70                                 target2_pressed += 1
71                             elif cls == 2:
72                                 clicks[cls] += 1
73
74                             hit_timestamps[cls].append(current_time)

```

```

67                                     if check_collision(targets3, (x1 + x2)
// 2, (y1 + y2) // 2, radius):
68                                     target3_pressed += 1
69
70                                     cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
71                                     label = f"{model.names[cls]} {conf:.2f} {radius:.1f}"
72                                     cv2.putText(img, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
73
74                                     fps = int(clock.get_fps())
75                                     cv2.putText(img, f"FPS: {fps}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
76
77                                     cv2.imshow("Camera Output", img)
78
79                                     for event in pygame.event.get():
80                                         if event.type == pygame.QUIT:
81                                             run = False
82                                             break
83
84                                         if event.type == pygame.KEYDOWN:
85                                             if event.key == pygame.K_r:
86                                                 return
87
88                                         if event.type == TARGET_EVENT:
89                                             x = random.randint(TARGET_PADDING, int(WIDTH/3) -
TARGET_PADDING)
90                                             y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
91                                             if len(targets1) <= 0:
92                                                 target1 = Target(x,y)
93                                                 targets1.append(target1)
94
95                                             x = random.randint(int(WIDTH/3) + TARGET_PADDING,
int((WIDTH/3)*2) - TARGET_PADDING)
96                                             y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
97                                             if len(targets2) <= 0:
98                                                 target2 = Target(x,y)
99                                                 targets2.append(target2)
100
101                                             x = random.randint(int((WIDTH/3)*2) + TARGET_PADDING,
int((WIDTH/3)*3) - TARGET_PADDING)
102                                             y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
103                                             if len(targets3) <= 0:
104                                                 target3 = Target(x,y)
105                                                 targets3.append(target3)
106
107                                     misses1 = update_targets(targets1, misses1)
108                                     misses2 = update_targets(targets2, misses2)
109                                     misses3 = update_targets(targets3, misses3)
110
111                                     if misses1 >= LIVES or misses2 >= LIVES or misses3 >= LIVES or
target1_pressed >= HITS or target2_pressed >= HITS or target3_pressed >=
HITS:
112                                         end_screen(WIN, elapsed_time, target1_pressed,
target2_pressed, target3_pressed, misses1, misses2, misses3, clicks) #
end game
113                                     return

```

```

114
115     draw(WIN, targets1, targets2, targets3, detected_circles)
116     draw_top_bar(WIN, elapsed_time, target1_pressed, misses1,
117     target2_pressed, misses2, target3_pressed, misses3)
118     pygame.display.update()
119
120     if cv2.waitKey(1) & 0xFF == ord('q'):
121         break
122
123     cap.release()
124     cv2.destroyAllWindows()
125     pygame.quit()

```

Gambar 4.17 Kode fungsi main

Selanjutnya, sinkronisasi dilakukan antara data deteksi dan elemen permainan. Ketika bola yang terdeteksi memenuhi kriteria dan mengenai target pada layar permainan, sistem mencatat *hit* dan memperbarui skor pemain. Selain itu, animasi visual ditampilkan untuk memberikan umpan balik langsung kepada pemain, sehingga meningkatkan interaktivitas dan imersi permainan. Jika bola tidak mengenai target, sistem mencatat "miss" dan memperhitungkan pengurangan akurasi pemain yang akan ditampilkan di akhir permainan.

Integrasi ini juga mencakup optimalisasi performa untuk mendukung deteksi *real-time* dengan latensi rendah. Dengan waktu inferensi rata-rata 8 ms per *frame*, model YOLOv8 memungkinkan permainan berjalan dengan responsif tanpa mengganggu alur gameplay. Secara keseluruhan, integrasi model ini memastikan bahwa setiap interaksi pemain dengan bola fisik dapat tercermin secara langsung dalam mekanisme permainan digital, menciptakan pengalaman bermain yang lebih realistis dan menyenangkan.

4.5.2 Feedback Real-Time dan Sistem Skor

Setelah *hit* terdeteksi, sistem memberikan umpan balik visual berupa animasi atau perubahan warna target dan mencatat skor pemain. Target yang terkena akan segera diganti dengan target baru di lokasi berbeda untuk menjaga dinamika permainan. Fitur ini memastikan deteksi yang responsif dan akurat, memberikan pengalaman bermain yang lebih realistis dan interaktif bagi pemain.

Pada akhir permainan, sistem secara otomatis menghitung dan menampilkan berbagai statistik untuk setiap pemain berdasarkan performa mereka selama permainan. Statistik ini mencakup jumlah *hits*, jumlah *misses*, jumlah *damages*, tingkat akurasi, dan penentuan pemenang. Semua statistik ini ditampilkan kepada pemain dalam format visual yang sederhana dan mudah dipahami yang dapat dilihat pada Gambar 4.18. Hal ini memberikan umpan balik

kepada pemain tentang performa mereka dan menambah elemen evaluasi serta pembelajaran dalam permainan.



Gambar 4.18 Antarmuka statistik di akhir permainan

Jumlah *hits* mencatat keberhasilan pemain dalam menghancurkan target. Sistem mencatat setiap kali bola yang dilempar pemain mengenai target dalam radius yang sesuai. Statistik ini menjadi indikator utama keberhasilan pemain selama permainan. Di sisi lain, jumlah *misses* dihitung dari semua lemparan bola yang tidak mengenai target. Hal ini mencakup bola yang tidak mengenai area permainan atau bola yang mengenai target di zona pemain lain. Formula sederhana digunakan untuk menghitungnya, yaitu jumlah lemparan dikurangi jumlah *hits*.

Jumlah *damages* dihitung berdasarkan jumlah target yang muncul dan menghilang tanpa terkena bola. Statistik ini mencerminkan seberapa banyak target yang terlewatkan oleh pemain selama permainan. Tingkat akurasi, di sisi lain, menjadi ukuran penting untuk mengevaluasi efektivitas pemain dalam melempar bola. Akurasi dihitung dalam bentuk persentase menggunakan formula (4.1) berikut:

$$\text{Akurasi (\%)} = \frac{\text{Hits}}{\text{Total Lemparan}} \cdot 100 \quad (4.2)$$

Statistik ini memberikan gambaran jelas tentang efisiensi setiap pemain. Jika pemain tidak melakukan lemparan sama sekali, akurasi akan dianggap nol persen.

Penentuan pemenang dilakukan berdasarkan jumlah *hits* tertinggi yang dicapai oleh pemain. Jika terjadi seri dalam jumlah *hits*, maka pemenang akan ditentukan berdasarkan jumlah *damages* paling sedikit. Apabila jumlah *hits* dan *damages* juga sama, maka akurasi tertinggi digunakan sebagai faktor penentu terakhir. Dengan mekanisme ini, sistem memastikan kompetisi berjalan secara adil dan transparan. Statistik yang disajikan tidak hanya membantu pemain untuk mengevaluasi performa mereka, tetapi juga memperkuat elemen kompetisi dalam permainan. Penyajian data yang objektif dan transparan membuat pengalaman bermain menjadi lebih menarik dan *imersif*, sekaligus memberikan wawasan yang bermanfaat bagi pemain untuk meningkatkan kemampuan mereka di masa mendatang.

4.6 Pengujian dan Evaluasi Implementasi

Tahap pengujian dan evaluasi bertujuan untuk menilai kinerja sistem secara teknis dan pengalaman pengguna. Pengujian ini mencakup pengukuran akurasi model deteksi, responsivitas permainan, dan tingkat kepuasan pengguna terhadap sistem. Berikut adalah struktur evaluasi yang dilakukan:

4.6.1 Pengujian Teknis

Pada skenario pengujian, 5 pemain diberikan kesempatan untuk bermain dengan target *hit* sebanyak 20 dan nyawa (*lives*) sebanyak 3. Permainan berlangsung tanpa aturan tentang siapa yang bermain dengan siapa, memungkinkan data permainan mencerminkan situasi yang lebih realistis seperti di lapangan. Demografi kelima pemain adalah sebagai berikut:

a. MRAF 23 Tahun (Pekerja sebagai programmer)

Kebiasaan: Setelah jam kerja, MRAF menghabiskan waktu lebih dari 7 jam di depan layar PC ataupun *mobile* di luar pekerjaan, bermain gim ataupun media sosial.

Aktivitas Fisik: Sangat sedikit aktifitas fisik, hanya berjalan kaki di kantor, dan di rumah.

b. NFYP 21 (Mahasiswa Informatika, Semester Akhir)

Kebiasaan: Karena tidak ada lagi mata kuliah maka bisa menghabiskan waktu lebih dari 12 jam di depan layar PC ataupun *mobile*, bermain gim ataupun media sosial.

Aktivitas Fisik: Minim aktifitas fisik, lebih sering rebahan jika tidak di depan layar atau berjalan keluar mengambil pesanan makanan *online*.

c. ARR 24 (Pekerja WFH sebagai QA)

Kebiasaan: Karena jam kerja yang fleksibel jadi lebih santai, bisa menghabiskan waktu lebih dari 10 jam di depan layar PC ataupun *mobile* di luar pekerjaan, bermain gim ataupun media sosial.

Aktivitas Fisik: Sedikit aktifitas fisik, kadang keluar jalan-jalan untuk *refreshing* bareng teman-teman.

d. MFR 23 (Mahasiswa Informatika, Semester Akhir)

Kebiasaan: Karena tidak ada lagi mata kuliah maka bisa menghabiskan waktu lebih dari 12 jam di depan layar PC ataupun *mobile*, bermain gim ataupun media sosial.

Aktivitas Fisik: Minim aktifitas fisik, lebih sering rebahan jika tidak di depan layar atau berjalan keluar mengambil pesanan makanan *online*.

e. RRA 22 (Mahasiswa Informatika, Semester Akhir)

Kebiasaan: Karena tidak ada lagi mata kuliah maka bisa menghabiskan waktu lebih dari 13 jam di depan layar PC ataupun *mobile*, bermain gim ataupun media sosial.

Aktivitas Fisik: Minim aktifitas fisik, lebih sering rebahan jika tidak di depan layar atau berjalan ke dapur untuk memasak makanan.

Demografi pemain memberikan gambaran yang lebih mendalam mengenai kebiasaan digital dan aktivitas fisik mereka yang relevan dalam pengujian ini. Sebagian besar pemain menghabiskan waktu lebih dari 7 jam sehari di depan layar, baik untuk bermain gim maupun mengakses media sosial. Gaya hidup ini mencerminkan pola perilaku yang sering ditemukan pada generasi muda yang aktif dalam dunia digital namun minim dalam melakukan aktivitas fisik.

Model ini mampu mendeteksi hingga tiga bola secara bersamaan dengan rata-rata tingkat deteksi *hit* sebesar 85% dalam kondisi pencahayaan normal hingga rendah. Data diperoleh dari tiga pemain yang paling sering bermain bersama dalam tiga kali pengulangan permainan untuk memastikan konsistensi dan relevansi hasil. Evaluasi model dilakukan dengan menggunakan algoritma deteksi *hit*, yang menentukan apakah bola yang terdeteksi berada dalam jangkauan target berdasarkan perhitungan jarak. Algoritma deteksi *hit* tidak langsung mengetahui apakah bola benar-benar mengenai target di dunia nyata, melainkan mengonfirmasi kontak berdasarkan lokasi bola yang terdeteksi dan menghitung jaraknya dengan target. Jika jarak antara bola yang terdeteksi dan target berada dalam ambang batas tertentu, maka sistem menganggapnya sebagai *hit*; jika tidak, maka dianggap *miss*.

Dalam Tabel 4.1, performa model diukur menggunakan empat metrik utama berdasarkan cara algoritma deteksi *hit* bekerja dan bagaimana peristiwa terjadi di dunia nyata:

Tabel 4.1 Hasil Pengujian YOLOv8 di permainan

Aspek	Player 1	Player 2	Player 3
Mengenai target TP	54	51	47
Terdeteksi sebagai miss TN	15	21	19
Tidak terdeteksi FN	6	9	9
Terdeteksi di lokasi yang salah FP	3	6	5
Akurasi TP+TN / TP+TN+FN+FP	88%	83%	83%

- a. *True Positive* (TP) - Bola benar-benar mengenai target di dunia nyata dan model mendeteksi kontak dengan benar.
 1. **Definisi:** Di dunia nyata, bola dilempar pemain dan mengenai target secara fisik. Model kemudian berhasil mendeteksi bola di lokasi yang benar dan menghitung jaraknya dengan target, sehingga algoritma deteksi *hit* mengonfirmasi bahwa bola bersentuhan dengan target dan gim menyatakannya sebagai *HIT*.
 2. **Contoh:** Pemain melempar bola langsung ke target, bola benar-benar menyentuh target, dan sistem mendeteksi serta mengonfirmasi kontak tersebut sebagai *hit* yang valid.
- b. *True Negative* (TN) – Bola benar-benar meleset dari target di dunia nyata dan model mendeteksi hal ini dengan benar.
 1. **Definisi:** Di dunia nyata, bola dilempar tetapi tidak mengenai target. Model mendeteksi bola di luar area target dan menghitung bahwa jaraknya tidak berada dalam ambang batas *hit*, sehingga algoritma deteksi *hit* mengonfirmasi bahwa bola tidak bersentuhan dengan target dan gim menyatakannya sebagai *MISS*.
 2. **Contoh:** Pemain melempar bola tetapi tidak mengenai target, dan sistem mendeteksi bola di luar area target, sehingga dianggap sebagai *miss* yang valid.
- c. *False Negative* (FN) – Bola mengenai target di dunia nyata tetapi model gagal mendeteksinya dengan benar.
 1. **Definisi:** Di dunia nyata, bola mengenai target secara fisik, tetapi model gagal mendeteksi bola di lokasi yang seharusnya atau menganggap bola berada di luar area *hit*, sehingga algoritma deteksi *hit* tidak dapat mengonfirmasi kontak, dan gim salah menyatakannya sebagai *MISS*.
 2. **Contoh:** Pemain melempar bola tepat ke target, bola menyentuh target secara nyata, tetapi karena kesalahan dalam deteksi objek, sistem tidak menemukan bola di posisi yang sesuai, sehingga *hit* tidak terhitung.
- d. *False Positive* (FP) – Bola tidak mengenai target di dunia nyata tetapi model salah mendeteksi adanya kontak.

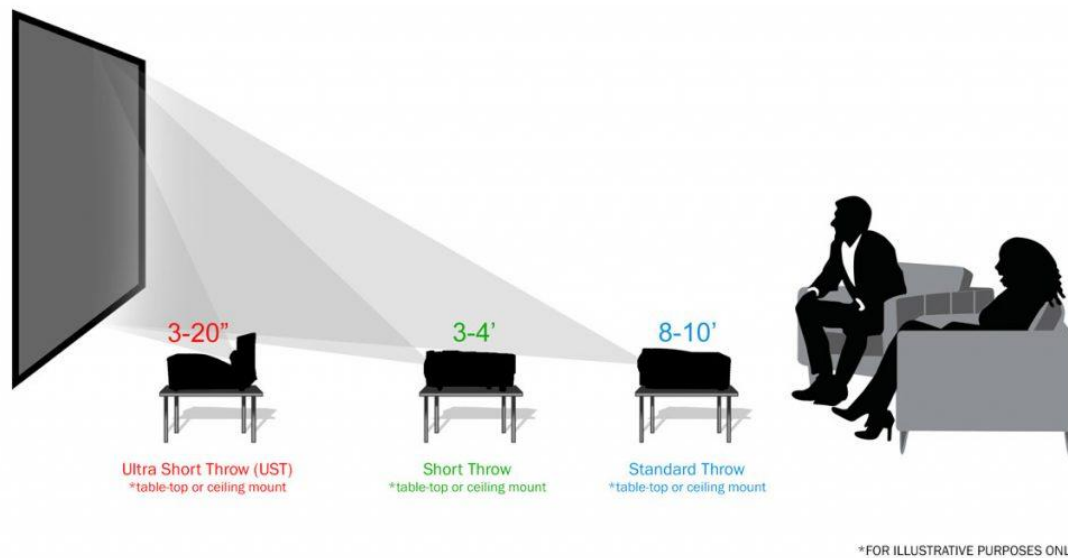
1. **Definisi:** Di dunia nyata, bola dilempar tetapi meleset dari target, tidak menyentuhnya sama sekali. Namun, karena kesalahan deteksi, model salah menganggap bola berada dalam jangkauan target dan algoritma deteksi *hit* secara keliru mengonfirmasi kontak, sehingga gim salah menyatakannya sebagai *HIT*.
2. **Contoh:** Pemain melempar bola, tetapi bola sebenarnya tidak mengenai target. Namun, model salah mendeteksi bola seolah-olah berada dalam area *hit*, sehingga sistem mencatatnya sebagai *hit* yang tidak valid.

Berdasarkan hasil pengujian dalam Tabel 4.1 model ini memiliki akurasi rata-rata di atas 83%, yang menunjukkan bahwa algoritma deteksi *hit* bekerja cukup baik dalam menghitung jarak antara bola yang terdeteksi dan target untuk menentukan status *hit* atau *miss*. Namun, kesalahan masih dapat terjadi, terutama jika bola tidak terdeteksi di lokasi yang benar (FN) atau jika model secara keliru menempatkan bola dalam jangkauan target (FP).

Selain itu, model ini memiliki fleksibilitas untuk dikembangkan lebih lanjut, seperti dengan menyesuaikan ambang batas deteksi *hit*, meningkatkan akurasi deteksi bola, atau mengembangkan fitur tambahan seperti deteksi kecepatan lemparan dan penyesuaian tingkat kesulitan permainan. Dalam pengujian ini, ditemukan bahwa meskipun posisi kamera beberapa kali dipindahkan dan terdapat variasi pencahayaan, algoritma deteksi *hit* tetap mampu beradaptasi dengan baik terhadap perubahan lingkungan sambil mempertahankan tingkat akurasi yang memadai.

Ditemukan fenomena menarik dalam pengujian ini, yaitu ketika jenis proyektor yang digunakan diganti dari proyektor *standard-throw* ke proyektor *short-throw* atau *ultra-short-throw*, akurasi model mengalami penurunan drastis. Hal ini diduga terjadi karena perbedaan karakteristik pencahayaan yang dihasilkan oleh kedua jenis proyektor tersebut dibandingkan dengan proyektor *standard-throw* dapat dilihat pada Gambar 4.19. Proyektor *standard-throw* yang digunakan saat pelatihan model memancarkan cahaya secara merata ke seluruh ruangan, sementara proyektor *short-throw* dan *ultra-short-throw* memancarkan cahaya dari sudut yang lebih tajam dan dalam jarak yang lebih dekat ke permukaan target. Akibatnya, distribusi cahaya yang mengenai bola menjadi tidak seragam, sehingga *shadow* yang dihasilkan menjadi tidak stabil. Dalam beberapa kasus, sebagian permukaan bola tidak terkena cahaya dengan cukup baik, menyebabkan model kesulitan mengenali bola sebagai objek yang sesuai dengan dataset pelatihan. Fenomena ini lebih terasa pada *ultra-short-throw* proyektor, di mana sudut pencahayaan yang sangat curam semakin memperburuk ketidakseimbangan cahaya pada objek. Kondisi ini meningkatkan kemungkinan *False Negative* (FN), di mana bola seharusnya

terdeteksi tetapi model gagal mengenalinya karena pencahayaan yang berbeda dari saat pelatihan.



Gambar 4.19 Perbandingan proyektor *ultra-short-throw*, *short-throw*, dan *standard-throw*

Sumber : (Rodgers, 2017)

Selain itu, penggunaan proyektor *short-throw* dan *ultra-short-throw* juga menyebabkan ketidaksesuaian dengan dataset pelatihan, yang berpengaruh terhadap kemampuan model dalam mendeteksi bola dengan benar. Perbedaan spektrum warna, intensitas cahaya, serta distribusi pencahayaan yang dihasilkan oleh proyektor ini berbeda dari proyektor *standard-throw*, yang dapat menyebabkan kesulitan model dalam mengenali bola, terutama dalam kondisi pencahayaan yang dinamis. Akibatnya, *False Positive* (FP) juga meningkat, di mana model salah mendeteksi objek lain sebagai bola karena tampilan yang berbeda dari dataset pelatihan. Hal ini menegaskan bahwa stabilitas pencahayaan sangat berpengaruh terhadap akurasi deteksi model, dan pemilihan jenis proyektor yang sesuai sangat penting untuk memastikan kinerja sistem tetap optimal.

Pada Tabel 4.2 menunjukkan bahwa pengujian model YOLOv8 dengan menggunakan 10 data gambar berbeda dalam lingkungan terkontrol menghasilkan rata-rata *confidence* sebesar 93%, mencerminkan akurasi tinggi dalam mendeteksi tiga jenis bola (*pinkball*, *yellowball*, dan *greenball*). Rata-rata *confidence* individu masing-masing bola adalah 93.1%

(pinkball), 95.2% (yellowball), dan 90.6% (greenball), menunjukkan konsistensi deteksi yang baik dengan sedikit variasi antar jenis bola.

Tabel 4.2 Hasil Pengujian Model YOLOv8

Gambar	PinkBall (%)	YellowBall (%)	GreenBall (%)	Rata Rata Confidence (%)
1	90	91	85	88.7
2	90	95	91	92.0
3	88	98	94	93.3
4	92	96	94	94.0
5	98	96	93	95.7
6	96	96	91	94.3
7	96	97	93	95.3
8	96	96	82	91.3
9	89	95	94	92.7
10	96	92	89	92.3
Rata-rata	93.1	95.2	90.6	93.0

Model juga mampu merespons dengan latensi rata-rata 30 ms, memastikan pengalaman bermain yang *real-time*. Selain itu, fitur deteksi *hit* dan sistem skor berfungsi optimal, mencatat interaksi pemain dengan target secara presisi. Pengujian menunjukkan stabilitas sistem tanpa kegagalan deteksi signifikan, meskipun terdapat variasi kecepatan lemparan bola. Hasil ini mengonfirmasi kemampuan model untuk mendukung pengalaman permainan yang dinamis dan akurat, menjadikannya cocok untuk implementasi skenario nyata.

4.6.2 Respons Pemain dan Evaluasi Gim

Setelah implementasi dan pengujian teknis selesai, evaluasi terhadap respons pemain dilakukan untuk memahami bagaimana Gim ini diterima oleh pengguna. Survei ini melibatkan lima pemain yang bermain dalam skenario multi-pemain. Tujuan utama dari survei ini adalah untuk mengevaluasi tingkat kesenangan, interaktivitas, dan efektivitas Gim dalam mendorong aktivitas fisik yang lebih intens. Selain itu, survei juga menilai apakah antarmuka awal Gim mudah dipahami oleh pemain baru.

Pertanyaan dalam survei dirancang untuk mencakup aspek emosional dan fungsional dari pengalaman bermain. Aspek emosional mencakup kesenangan dan interaktivitas, sedangkan aspek fungsional fokus pada antarmuka pengguna dan dampak Gim terhadap aktivitas fisik pemain. Hasil survei seperti pada

Tabel 4.3. memberikan wawasan penting yang menjadi dasar untuk perbaikan lebih lanjut dalam desain gim.

Tabel 4.3 Hasil Survei Pemain Skala 10

Pemain	Apakah permainan ini menyenangkan dan interaktif?	Apakah permainan ini mendorong aktivitas fisik lebih intens?	Apakah Anda merasa antarmuka Gim mudah dipahami?
1	9	8	8
2	8	6	6
3	10	9	9
4	9	9	8
5	8	6	8
Rata-Rata	8.8	7.6	7.8

Hasil survei menunjukkan bahwa kesenangan dan interaktivitas permainan memperoleh skor rata-rata 8.8, yang masuk dalam kategori *Excellent* berdasarkan *System Usability Scale* (SUS). Skor ini menunjukkan bahwa mayoritas pemain menikmati permainan dengan baik, merasakan pengalaman yang *imersif*, serta menganggap bahwa aspek interaktivitas gim telah berhasil dieksekusi dengan optimal. Sementara itu, pada aspek dorongan terhadap aktivitas fisik, permainan mendapatkan skor rata-rata 7.6, yang termasuk dalam kategori *Good*. Hal ini menunjukkan bahwa gim cukup efektif dalam meningkatkan aktivitas fisik pemain dibandingkan aktivitas sehari-hari mereka. Namun, skor ini juga mengindikasikan bahwa beberapa pemain merasa tantangan fisik yang diberikan masih dapat ditingkatkan untuk memberikan stimulasi yang lebih intens dan menarik.

Pada aspek kemudahan memahami antarmuka gim, skor rata-rata yang diperoleh adalah 7.8, yang juga masuk dalam kategori *Good*. Skor ini menunjukkan bahwa sebagian besar pemain menganggap antarmuka gim cukup intuitif dan mudah dipahami, meskipun masih ada ruang untuk perbaikan. Beberapa pemain mungkin memerlukan panduan yang lebih jelas atau tutorial interaktif agar lebih cepat memahami mekanisme permainan, terutama bagi pengguna baru yang belum terbiasa dengan sistem kontrol dan tampilan gim.

Secara keseluruhan, evaluasi dengan *System Usability Scale* (Bangor et al., 2009) menunjukkan bahwa gim ini memiliki tingkat kepuasan pengguna yang tinggi, terutama dalam aspek kesenangan dan interaktivitas. Namun, ada beberapa aspek yang masih dapat ditingkatkan, seperti penyesuaian tingkat tantangan fisik agar lebih menarik bagi pemain

dengan berbagai tingkat kebugaran, serta penyempurnaan elemen antarmuka agar lebih ramah bagi pemain baru. Dengan melakukan iterasi desain berbasis umpan balik pengguna, diharapkan gim ini dapat terus berkembang dan memberikan pengalaman bermain yang lebih optimal serta mendukung keterlibatan pemain secara lebih efektif.

4.7 Diskusi

Dalam penelitian ini, selain hasil yang diperoleh terkait dengan teknologi YOLOv8 dalam gim lempar bola berbasis gerakan, kami juga menemukan beberapa temuan menarik yang dapat membuka peluang untuk pengembangan lebih lanjut dan penerapan teknologi ini dalam berbagai konteks.

- a. **Efisiensi Penggunaan Perangkat Keras:** Salah satu pencapaian utama adalah penggunaan perangkat keras yang sederhana, yang dapat mengurangi biaya dan kompleksitas tanpa mengurangi kualitas deteksi objek. Hal ini membuka kemungkinan untuk aplikasi gim serupa yang lebih terjangkau bagi lebih banyak orang, menjadikan teknologi ini lebih mudah diakses oleh berbagai kalangan.
- b. **Potensi Penggunaan di Berbagai Skenario:** PoC ini memiliki potensi besar untuk diimplementasikan dalam berbagai konteks, seperti pendidikan jasmani di sekolah, hiburan keluarga, atau kegiatan komunitas. Teknologi ini tidak hanya berfungsi sebagai hiburan, tetapi juga dapat digunakan untuk meningkatkan aktivitas fisik dalam berbagai *setting*.
- c. **Responsivitas Real-Time yang Luar Biasa:** Saat bola mengenai target, sistem memberikan umpan balik langsung berupa skor atau visual lain yang menandakan keberhasilan dalam rata-rata 30ms. Responsivitas ini penting untuk pengalaman bermain yang *imersif* dan mengurangi latensi dalam permainan.
- d. **Motivasi Pemain untuk Bergerak Lebih Aktif:** Meskipun bukan tujuan utama, penelitian menemukan bahwa banyak pemain merasa lebih termotivasi untuk berpartisipasi dalam aktivitas fisik setelah bermain gim ini. Sebanyak 76% pemain melaporkan bahwa mereka merasa lebih aktif setelah bermain, yang menunjukkan bahwa teknologi berbasis Computer Vision dapat menjadi alat yang efektif untuk mendorong pemain, terutama generasi muda, untuk bergerak lebih banyak.
- e. **Potensi Kolaborasi Sosial dalam Mode Multi-Pemain:** Pemain dapat berinteraksi dengan satu sama lain meskipun berada di ruang yang sama, memperkaya pengalaman sosial mereka. Meskipun ini bukan tujuan utama dari penelitian, aspek sosial ini membuka

peluang untuk pengembangan lebih lanjut yang lebih fokus pada interaksi sosial dalam gim berbasis teknologi.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini berhasil mengembangkan *Proof of Concept* (PoC) yang bertujuan untuk mengintegrasikan teknologi Computer Vision berbasis YOLOv8 dalam gim interaktif untuk meningkatkan aktivitas fisik generasi muda di era digital. Berdasarkan hasil penelitian yang dilakukan, beberapa kesimpulan utama yang dapat diambil adalah sebagai berikut:

- a. **Implementasi Teknologi YOLOv8 dalam Gim Lempar Bola untuk Deteksi Bola dan Algoritma Deteksi Hit:** Penelitian ini berhasil mengimplementasikan teknologi Computer Vision menggunakan YOLOv8 untuk mendeteksi pergerakan bola dalam gim lempar bola berbasis gerakan secara real-time. Selain itu, penelitian ini juga mengembangkan algoritma deteksi hit yang memungkinkan sistem untuk menentukan apakah bola mengenai target dengan akurat. Algoritma deteksi hit bekerja dengan memverifikasi posisi bola yang terdeteksi oleh YOLOv8 terhadap area target yang telah ditentukan, memberikan umpan balik langsung kepada pemain. Sistem ini menggunakan perangkat keras sederhana seperti kamera dan proyektor *standard-throw*, membuktikan bahwa teknologi ini dapat diterapkan secara efektif meskipun dengan sumber daya terbatas.
- b. **Performa YOLOv8 dalam Deteksi Bola dan Respons dalam Gim Lempar Bola:** YOLOv8 mampu mendeteksi bola dengan akurasi tinggi, mencapai nilai mAP50-95 sebesar 93%, dan memberikan respons yang cepat dalam gim berbasis gerakan. Berdasarkan hasil pengujian, akurasi deteksi bola dalam permainan mencapai lebih dari 83%, menunjukkan efektivitas model dalam mendeteksi bola dengan tepat. Latensi respons YOLOv8 rata-rata sekitar 30 ms, memastikan pengalaman bermain yang lancar dan real-time. Hasil ini mengonfirmasi bahwa model ini dapat digunakan secara efektif dalam permainan dinamis yang memerlukan deteksi objek cepat dan akurat.

5.2 Saran Pengembangan

Untuk pengembangan dan penelitian lebih lanjut, beberapa saran yang dapat dipertimbangkan adalah:

- a. **Uji Coba di Lingkungan Dunia Nyata:** Untuk mendapatkan gambaran yang lebih lengkap mengenai keberhasilan gim dalam meningkatkan aktivitas fisik dan keterlibatan pemain, disarankan untuk melakukan uji coba di luar kondisi laboratorium yang terkontrol, seperti di sekolah, pusat kebugaran, atau komunitas publik.
- b. **Pengembangan Mode Permainan yang Lebih Variatif:** Pengembangan lebih lanjut dapat mencakup penambahan mode permainan yang lebih variatif dengan tingkat kesulitan yang lebih tinggi, atau mode *multiplayer* yang mendukung lebih banyak pemain.
- c. **Penerapan dalam Terapi dan Rehabilitasi Fisik:** Gim ini memiliki potensi untuk diadaptasi dalam rehabilitasi fisik atau terapi, di mana pemain dapat melakukan gerakan-gerakan yang diperlukan dalam terapi sambil tetap bermain dan bersenang-senang.
- d. **Pengembangan Mode Online dan Multiplayer:** Mode *multiplayer online* dapat menjadi langkah selanjutnya untuk memperluas audiens gim ini, memungkinkan interaksi antar pemain dari berbagai lokasi secara *real-time*.
- e. **Integrasi dengan Teknologi AR dan VR:** Teknologi *Augmented Reality* (AR) atau *Virtual Reality* (VR) dapat digunakan untuk meningkatkan pengalaman bermain dan menciptakan lingkungan permainan yang lebih *imersif*, membawa gim ini ke tingkat berikutnya.
- f. **Optimasi Perangkat Keras:** Pengujian dengan perangkat keras yang lebih murah namun tetap efektif untuk penggunaan massal dapat menjadi fokus dalam penelitian selanjutnya. Penggunaan kamera atau dengan berbagai jenis proyektor (*standard, short, dan ultra-short throw*) dengan spesifikasi lebih rendah bisa membuat gim ini lebih terjangkau tanpa mengurangi kualitas deteksi objek

DAFTAR PUSTAKA

- Abdullah, A., Amran, G. A., Tahmid, S. M. A., Alabrah, A., AL-Bakhrani, A. A., & Ali, A. (2024). A Deep-Learning-Based Model for the Detection of Diseased Tomato Leaves. *Agronomy*, 14(7). <https://doi.org/10.3390/agronomy14071593>
- Asmara, R. A., Anugrah, M. R. S., Wibowo, D. W., Arai, K., Burhanuddin, M. A., Handayani, A. N., & Damayanti, F. A. (2024). YOLO-based object detection performance evaluation for automatic target aimbot in first-person shooter games. *Bulletin of Electrical Engineering and Informatics*, 13(4), 2456–2470. <https://doi.org/10.11591/eei.v13i4.6895>
- Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean; adding an adjective rating. *Journal of Usability Studies*, 4(3), 114–123.
- BPS. (2018). Dalam Angka Dalam Angka. In *Kota Kediri Dalam Angka*. <https://www.badankebijakan.kemkes.go.id/ski-2023-dalam-angka/>
- Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, 82(6), 9243–9275. <https://doi.org/10.1007/s11042-022-13644-y>
- Doskarayev, B., Omarov, N., Omarov, B., Ismagulova, Z., Kozhamkulova, Z., Nurlybaeva, E., & Kasimova, G. (2023). Development of Computer Vision-enabled Augmented Reality Games to Increase Motivation for Sports. *International Journal of Advanced Computer Science and Applications*, 14(4), 245–252. <https://doi.org/10.14569/IJACSA.2023.0140428>
- Erwin Syahrudin, Ema Utami, & Anggit Dwi Hartanto. (2024). Enhanced YOLOv8 with OpenCV for Blind-Friendly Object Detection and Distance Estimation. *Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi)*, 8(2), 199–207. <https://doi.org/10.29207/resti.v8i2.5529>
- Fullerton, T. (2024). Game Design Workshop: A Playcentric Approach to Creating Innovative Games, Fifth Edition. In *Game Design Workshop: A Playcentric Approach to Creating Innovative Games, Fifth Edition* (5th Editio). A K Peters/CRC Press. <https://doi.org/10.1201/9781003460268>
- Gao, Q., Deng, H., & Zhang, G. (2024). A Contraband Detection Scheme in X-ray Security Images Based on Improved YOLOv8s Network Model. *Sensors*, 24(4). <https://doi.org/10.3390/s24041158>
- Giannios, I., & Margounakis, D. G. (2022). INSIDE. In *Research Anthology on Virtual*

- Environments and Building the Metaverse* (pp. 611–629). IGI Global.
<https://doi.org/10.4018/978-1-6684-7597-3.ch030>
- Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision, 2015 Inter*, 1440–1448. <https://doi.org/10.1109/ICCV.2015.169>
- Guennouni, S., Ahaitouf, A., & Mansouri, A. (2015). Multiple object detection using OpenCV on an embedded platform. *Colloquium in Information Science and Technology, CIST, 2015-Janua*(January), 374–377. <https://doi.org/10.1109/CIST.2014.7016649>
- Gündüz, M. Ş., & Işık, G. (2023). A new YOLO-based method for social distancing from real-time videos. *Neural Computing and Applications*, 35(21), 15261–15271. <https://doi.org/10.1007/s00521-023-08556-3>
- Hedenfalk, Z., & Rehnström, T. (2024). *Comparison of image preprocessing techniques on parking sign detection in Sweden*. KTH Royal Institute of Technology.
- Huang, H., Wang, B., Xiao, J., & Zhu, T. (2024). Improved small-object detection using YOLOv8: A comparative study. *Applied and Computational Engineering*, 41(1), 80–88. <https://doi.org/10.54254/2755-2721/41/20230714>
- Huang, Y.-Q., Zheng, J.-C., Sun, S.-D., Yang, C.-F., & Liu, J. (2020). Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections. *Applied Sciences*, 10(9). <https://doi.org/10.3390/app10093079>
- Hussain, M. (2023). YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines*, 11(7). <https://doi.org/10.3390/machines11070677>
- Iswaratio, V. H. (2022). State of Mobile 2022: Finance. In *Appannie.Com*. <https://www.data.ai/en/go/state-of-mobile-2022-indonesia/>
- Jalil, F., & Voronkov, I. (2016). *Object Detection using Image Processing*. 1–6. <http://arxiv.org/abs/1611.07791>
- Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147, 70–90. <https://doi.org/10.1016/j.compag.2018.02.016>
- Khude, S. A., Patil, N. V., Darade, S. S., Galande, S. S., & Maruti, A. (2024). *Custom Dataset Using Yolo. April*.
- Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD:

- Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 9905 LNCS* (pp. 21–37). Springer International Publishing. https://doi.org/10.1007/978-3-319-46448-0_2
- Lü Interactive Playground. (2024). *Lü Interactive Playground: Opening Schools To New Worlds Making Each Child Bloom As A Whole*. Play-Lu.Com. play-lu.com
- Mahardhika, G. P. (2019). Gim Proses Donor Darah Sebagai Alat Promosi Donor Darah. *JMAI (Jurnal Multimedia \& Artificial Intelligence)*. <https://api.semanticscholar.org/CorpusID:198469131>
- Mueller, F., Agamanolis, S., & Picard, R. (2003). Exertion interfaces: Sports over a distance for social bonding and fun. In *Conference on Human Factors in Computing Systems - Proceedings*.
- Mueller, F. F., Cole, L., O'Brien, S., & Walmink, W. (2006). Airhockey over a distance. *Conference on Human Factors in Computing Systems - Proceedings*, 1133–1138. <https://doi.org/10.1145/1125451.1125665>
- Nelson, R. (2023). State of Mobile 2023: Southeast Asia (SEA). In *Data.Ai*. <https://www.data.ai>
- Patel, N. L., Patel, K. A., & Datta, J. S. (2024). Integrating YOLOv8 with Fuzzy Logic System: A Novel Hybrid Approach for Efficient Parking Slot Availability Management. *ITM Web of Conferences*, 65, 03009. <https://doi.org/10.1051/itmconf/20246503009>
- Reddy, S. A. (2023). Vehicle Counting And Traffic Congestion Detection Using YOLOv3. *International Journal For Multidisciplinary Research*, 5(4), 3–10. <https://doi.org/10.36948/ijfmr.2023.v05i04.4616>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-Decem*, 779–788. <https://doi.org/10.1109/CVPR.2016.91>
- Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137–1149. <https://doi.org/10.1109/TPAMI.2016.2577031>
- Rodgers, D. (2017). *The Difference between Standard, Short Throw and UST Projectors*. Elite Screens. <https://projectorscreenresource.com/2017/10/19/the-difference-between-standard-short-throw-and-ust-projectors/>

- Sharafaldeen, J., Rizk, M., Heller, D., Baghdadi, A., & Diguët, J.-P. (2022). Marine Object Detection Based on Top-View Scenes Using Deep Learning on Edge Devices. *2022 International Conference on Smart Systems and Power Management (IC2SPM)*, 35–40. <https://doi.org/10.1109/IC2SPM56638.2022.9988928>
- Sholahuddin, M. R., Harika, M., Awaludin, I., Dewi, Y. C., Dhia Fauzan, F., Sudimulya, B. P., & Widarta, V. P. (2023). Optimizing YOLOv8 for Real-Time CCTV Surveillance: A Trade-off Between Speed and Accuracy. *Jurnal Online Informatika*, 8(2), 261–270. <https://doi.org/10.15575/join.v8i2.1196>
- Simon, M., Amende, K., Kraus, A., Honer, J., Sämann, T., Kaulbersch, H., Milz, S., & Gross, H. M. (2019). Complexer-YOLO: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1190–1199. <https://doi.org/10.1109/CVPRW.2019.00158>
- Sun, Y., & Loparo, K. (2019). Context aware image annotation in active learning with batch mode. *Proceedings - International Computer Software and Applications Conference*, 1, 952–953. <https://doi.org/10.1109/COMPSAC.2019.00157>
- Terven, J., Córdova-Esparza, D. M., & Romero-González, J. A. (2023). A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, 5(4), 1680–1716. <https://doi.org/10.3390/make5040083>
- Tzutalin. (2015). *LabelImg*. Git code (2015). <https://github.com/tzutalin/labelImg%0A%0A>
- Uke, N. J. (2012). *Cultivating Research in Computer Vision within Graduates and Post-Graduates using Open Source*. 1(4), 1–6.
- Wang, Y., Fu, B., Fu, L., & Xia, C. (2023). In Situ Sea Cucumber Detection across Multiple Underwater Scenes Based on Convolutional Neural Networks and Image Enhancements. *Sensors*, 23(4). <https://doi.org/10.3390/s23042037>
- Woodward, K., Kanjo, E., Parker, W., & Patrick, B. (2024). A stand-alone proximity-based gaming wearable for remote physical activity monitoring. *Data-Centric Engineering*, 5(12). <https://doi.org/10.1017/dce.2024.6>
- World Health Organization. (2022). Global status report on physical activity 2022. In *WHO Press, World Health Organization*. <https://www.who.int/teams/health-promotion/physical-activity/global-status-report-on-physical-activity-2022>

LAMPIRAN

```
1 import cv2
2 import numpy as np
3 import pickle
4
5 width, height = 1280, 720
6 cap = cv2.VideoCapture(0)
7 fileName = 'calibration.p'
8
9 cap.set(3, width)
10 cap.set(4, height)
11 circles = np.zeros((4, 2), int)
12 counter = 0
13
14 def mousePoints(event, x, y, flags, params):
15     global counter
16     if event == cv2.EVENT_LBUTTONDOWN:
17         circles[counter] = x, y
18         counter += 1
19         print(circles)
20
21 while True:
22     success, img = cap.read()
23
24     if counter == 4:
25         fileObj = open(fileName, 'wb')
26         pickle.dump(circles, fileObj)
27         fileObj.close()
28
29         pts1 = np.float32([circles[0], circles[1], circles[2],
30 circles[3]])
31         pts2 = np.float32([[0, 0], [width, 0], [0, height], [width,
32 height]])
33         matrix = cv2.getPerspectiveTransform(pts1, pts2)
34         imgOutput = cv2.warpPerspective(img, matrix, (width, height))
35         cv2.imshow("Output Image", imgOutput)
36
37         for x in range(0, 4):
38             cv2.circle(img, (circles[x][0], circles[x][1]), 3, (0, 255, 0),
39 cv2.FILLED)
40
41         cv2.imshow("Original Image", img)
42         cv2.setMouseCallback("Original Image", mousePoints)
43
44         key = cv2.waitKey(1) & 0xFF
45         if key == ord('q'):
46             break
47         elif key == ord('r'):
48             circles = np.zeros((4, 2), int)
49             counter = 0
50
51 cap.release()
52 cv2.destroyAllWindows()
```

File Calibration.py

```
1 import math
2 import random
```

```

3  import time
4  import pygame
5  import pickle
6  import cv2
7  import numpy as np
8  # import cvzone
9  from ultralytics import YOLO
10 import logging
11
12 logging.getLogger().setLevel(logging.ERROR)
13
14 pygame.init()
15 WIDTH, HEIGHT = 1280, 720
16
17 # Load calibration file for warp perspective
18 fileObj = open('calibration.p', 'rb')
19 circles = pickle.load(fileObj)
20 fileObj.close()
21
22 # Define perspective transform points
23 pts1 = np.float32([circles[0], circles[1], circles[2], circles[3]])
24 pts2 = np.float32([[0, 0], [WIDTH, 0], [0, HEIGHT], [WIDTH, HEIGHT]])
25 matrix = cv2.getPerspectiveTransform(pts1, pts2)
26
27 model = YOLO("bolapolos_best.pt")
28 cap = cv2.VideoCapture(0)
29 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
30 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
31
32 WIN = pygame.display.set_mode((WIDTH, HEIGHT), pygame.RESIZABLE)
33 pygame.display.set_caption("Game Lempar Bola")
34
35 TARGET_INCREMENT = 400
36 TARGET_EVENT = pygame.USEREVENT
37
38 DETECTION_DELAY = 2.0
39 TARGET_PADDING = 80
40 MIN_RADIUS = 83
41 MAX_RADIUS = 93
42 P1_COLOR = (222, 81, 123) # Pink
43 P2_COLOR = (232, 173, 70) # Yellow
44 P3_COLOR = (111, 194, 186) # Green
45 BG1_COLOR = (79, 68, 89)
46 BG2_COLOR = (64, 64, 64)
47 LIVES = 2
48 HITS = 20
49 TOP_BAR_HEIGHT = 50
50 last_detected = {0: 0, 1: 0, 2: 0}
51 hit_timestamps = {1: [], 2: [], 3: []}
52
53 LABEL_FONT = pygame.font.SysFont("Fredoka-Bold.ttf", 35)
54
55 class Target:
56     MAX_SIZE = 70
57     GROWTH_RATE = 0.3
58     SECOND_COLOR = 122, 60, 186
59
60     def __init__(self, x, y, size=0):
61         self.x = x
62         self.y = y

```

```

63         self.size = size
64         self.grow = True
65
66     def update(self):
67         if self.size + self.GROWTH_RATE >= self.MAX_SIZE:
68             self.grow = False
69
70         if self.grow:
71             self.size += self.GROWTH_RATE
72         else:
73             self.size -= self.GROWTH_RATE
74
75     def draw(self, win, color):
76         pygame.draw.circle(win, color, (self.x, self.y), self.size)
77         pygame.draw.circle(win, self.SECOND_COLOR, (self.x, self.y),
self.size * 0.8)
78         pygame.draw.circle(win, color, (self.x, self.y), self.size *
0.6)
79         pygame.draw.circle(win, self.SECOND_COLOR, (self.x, self.y),
self.size * 0.4)
80
81     def collide(self, x, y, radius):
82         dis = math.sqrt((self.x - x)**2 + (self.y - y)**2)
83         return dis <= self.size + radius
84
85 # Function to apply warp perspective to the camera frame
86 def warpImage(img, circles, width, height):
87     pts1 = np.float32([circles[0], circles[1], circles[2],
circles[3]])
88     pts2 = np.float32([[0, 0], [width, 0], [0, height], [width,
height]])
89     matrix = cv2.getPerspectiveTransform(pts1, pts2)
90     imgWarped = cv2.warpPerspective(img, matrix, (width, height))
91     return imgWarped
92
93 def main_menu():
94     global MIN_RADIUS, MAX_RADIUS # Agar nilai bisa diubah oleh input
pengguna
95     run = True
96     ready_players = {0: False, 1: False, 2: False} # Status kesiapan
setiap pemain
97     target_center = Target(WIDTH // 2, HEIGHT // 2, size=50) # Target
di tengah
98     clock = pygame.time.Clock()
99     input_active = False
100    input_box = pygame.Rect(WIDTH // 2 - 150, HEIGHT // 2 + 200, 300,
50)
101    user_input = f"{MIN_RADIUS},{MAX_RADIUS}"
102    font = pygame.font.SysFont("Arial", 24)
103
104    while run:
105        clock.tick(60)
106        WIN.fill(BG1_COLOR) # Bersihkan layar
107        target_center.draw(WIN, P1_COLOR) # Gambar target utama di
tengah
108
109        # Tampilkan teks instruksi
110        title_text = LABEL_FONT.render("Hit Target to Ready", 1,
"white")

```

```

111     WIN.blit(title_text, (WIDTH // 2 - title_text.get_width() //
112 2, HEIGHT // 2 - 100))
113     # Tampilkan status pemain
114     for i, color in enumerate([P1_COLOR, P2_COLOR, P3_COLOR]):
115         status = "Ready" if ready_players[i] else "Not Ready"
116         status_text = LABEL_FONT.render(f"Player {i + 1}:
117 {status}", 1, color)
118         WIN.blit(status_text, (50, 100 + i * 50))
119     # Input box untuk radius
120     pygame.draw.rect(WIN, (255, 255, 255), input_box, 2)
121     input_text = font.render(f"Radius: {user_input}", True, (255,
122 255, 255))
123     WIN.blit(input_text, (input_box.x + 10, input_box.y + 10))
124     # Deteksi lingkaran menggunakan kamera
125     success, img = cap.read()
126     if not success:
127         pygame.display.update()
128         continue
129
130     scale_factor = 1 # Adjust this if needed
131     output_width = WIDTH // scale_factor
132     output_height = HEIGHT // scale_factor
133
134     img = warpImage(img, circles, output_width, output_height)
135     img = cv2.resize(img, (WIDTH, HEIGHT))
136
137     # img = cv2.resize(img, (WIDTH, HEIGHT))
138     results = model(img, stream=True)
139
140     # Terapkan warp perspective dan scaling
141     # img = warpImage(img, circles, WIDTH, HEIGHT)
142     results = model(img, stream=True)
143
144     for r in results:
145         boxes = r.boxes
146         for box in boxes:
147             x1, y1, x2, y2 = box.xyxy[0]
148             x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
149             center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
150             cls = int(box.cls[0]) # Warna (0: pink, 1: kuning, 2:
151 hijau)
152             color = P1_COLOR if cls == 0 else P2_COLOR if cls == 1
153             else P3_COLOR
154
155             width = x2 - x1
156             height = y2 - y1
157
158             # Calculate the radius of the detected bounding box
159             radius = min(width, height) / 2
160             pygame.draw.circle(WIN, color, (center_x, center_y),
161 radius, 2)
162
163             label = f"{radius:.1f}"
164             label_surface = LABEL_FONT.render(label, 1, "white")
165             WIN.blit(label_surface, (center_x - radius, center_y -
166 radius))
167
168             # Debugging

```

```

164         print(f"Detected circle at ({center_x}, {center_y})
with radius {radius}")
165
166         # Jika target berhasil dipukul oleh deteksi, ubah
status pemain
167         if MIN_RADIUS <= radius <= MAX_RADIUS:
168             if target_center.collide(center_x, center_y,
radius):
169                 ready_players[cls] = True
170                 print(f"Player {cls + 1} ready!")
171         # Draw bounding box
172         cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
173         # Add label with class name and confidence
174         label = f"{model.names[cls]} {radius:.1f}"
175         cv2.putText(img, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
176
177         # Add FPS to the image
178         fps = int(clock.get_fps())
179         cv2.putText(img, f"FPS: {fps}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
180
181         # Show the image
182         cv2.imshow("Camera Output", img)
183
184         # Tampilkan perubahan layar
185         pygame.display.update()
186
187         # Proses input pengguna
188         for event in pygame.event.get():
189             if event.type == pygame.QUIT:
190                 pygame.quit()
191                 quit()
192
193         # Input keyboard untuk radius
194         if event.type == pygame.MOUSEBUTTONDOWN:
195             if input_box.collidepoint(event.pos):
196                 input_active = True
197             else:
198                 input_active = False
199
200         if event.type == pygame.KEYDOWN and input_active:
201             if event.key == pygame.K_RETURN:
202                 try:
203                     # Update nilai MIN_RADIUS dan MAX_RADIUS
204                     min_r, max_r = map(int, user_input.split(","))
205                     MIN_RADIUS, MAX_RADIUS = min_r, max_r
206                 except ValueError:
207                     pass # Abaikan jika input tidak valid
208                 user_input = f"{MIN_RADIUS},{MAX_RADIUS}" # Reset
input box
209             elif event.key == pygame.K_BACKSPACE:
210                 user_input = user_input[:-1]
211             else:
212                 user_input += event.unicode
213
214         # Jika semua pemain siap, keluar dari menu utama
215         if all(ready_players.values()):
216             countdown()
217         return # Proceed to the next part of the game

```

```

218
219def countdown():
220     countdown_numbers = [3, 2, 1, "Go!"]
221     font = pygame.font.SysFont("Arial", 200)
222
223     for number in countdown_numbers:
224         WIN.fill(BG1_COLOR)
225         countdown_text = font.render(str(number), 1, "white")
226         WIN.blit(countdown_text, (WIDTH // 2 -
countdown_text.get_width() // 2, HEIGHT // 2 -
countdown_text.get_height() // 2))
227         pygame.display.update()
228         time.sleep(1) # Wait for 1 second before showing the next
number
229
230def draw(win, targets1, targets2, targets3, detected_circles):
231     pygame.draw.rect(win, (BG1_COLOR), (0, 0, WIDTH/3, HEIGHT))
232     pygame.draw.rect(win, (BG2_COLOR), (WIDTH/3, 0, (WIDTH/3) * 2,
HEIGHT))
233     pygame.draw.rect(win, (BG1_COLOR), ((WIDTH/3) * 2, 0, (WIDTH/3) *
3, HEIGHT))
234
235     for target1 in targets1:
236         target1.draw(win, (P1_COLOR))
237
238     for target2 in targets2:
239         target2.draw(win, (P2_COLOR))
240
241     for target3 in targets3:
242         target3.draw(win, (P3_COLOR))
243
244     # Draw detected circles
245     for circle in detected_circles:
246         x, y, radius, cls, conf = circle
247         color = P1_COLOR if cls == 0 else P2_COLOR if cls == 1 else
P3_COLOR
248         pygame.draw.circle(win, color, (x, y), radius, 2)
249         # Optionally add a label for debugging
250         label = f"{conf:.2f}"
251         label_surface = LABEL_FONT.render(label, 1, "white")
252         win.blit(label_surface, (x - radius, y - radius))
253
254def format_time(secs):
255     ms = math.floor(int(secs * 1000 % 1000) / 100)
256     s = int(round(secs % 60, 1))
257     m = int(secs // 60)
258
259     return f"{m:02d}:{s:02d}:{ms}"
260
261def draw_top_bar(win, elapsed_time, target1_pressed, misses1,
target2_pressed, misses2, target3_pressed, misses3):
262     pygame.draw.rect(win, (P1_COLOR), (0, 0, WIDTH/3, TOP_BAR_HEIGHT))
263     pygame.draw.rect(win, (P2_COLOR), (WIDTH/3, 0, (WIDTH/3) * 2,
TOP_BAR_HEIGHT))
264     pygame.draw.rect(win, (P3_COLOR), ((WIDTH/3) * 2, 0, (WIDTH/3) *
3, TOP_BAR_HEIGHT))
265     pygame.draw.rect(win, ("black"), (0, TOP_BAR_HEIGHT, WIDTH, 10))
266
267     speed1 = round(target1_pressed / elapsed_time, 1)
268     speed2 = round(target2_pressed / elapsed_time, 1)

```

```

269     speed3 = round(target3_pressed / elapsed_time, 1)
270
271     hits1_label = LABEL_FONT.render(f"Hits : {target1_pressed}", 1,
"white",)
272     hits2_label = LABEL_FONT.render(f"Hits : {target2_pressed}", 1,
"white")
273     hits3_label = LABEL_FONT.render(f"Hits : {target3_pressed}", 1,
"white")
274
275     lives1_label = LABEL_FONT.render(f"Lives : {LIVES - misses1}", 1,
"white")
276     lives2_label = LABEL_FONT.render(f"Lives : {LIVES - misses2}", 1,
"white")
277     lives3_label = LABEL_FONT.render(f"Lives : {LIVES - misses3}", 1,
"white")
278
279
280     win.blit(hits1_label, (20, 10))
281     win.blit(lives1_label, (280, 10))
282     win.blit(hits2_label, (450, 10))
283     win.blit(lives2_label, (710, 10))
284     win.blit(hits3_label, (880, 10))
285     win.blit(lives3_label, (1140, 10))
286
287 def calculate_speed(hit_timestamps):
288     if len(hit_timestamps) < 2:
289         return 0.0 # Not enough data to calculate speed
290
291     # Calculate the time intervals between consecutive hits
292     intervals = [
293         hit_timestamps[i] - hit_timestamps[i - 1]
294         for i in range(1, len(hit_timestamps))
295     ]
296
297     # Compute the average time interval between hits
298     avg_interval = sum(intervals) / len(intervals)
299
300     # Calculate speed (hits per second)
301     return round(1 / avg_interval, 2) # Speed = 1 / average_interval
302
303 def end_screen(win, elapsed_time, target1_pressed, target2_pressed,
target3_pressed, misses1, misses2, misses3, clicks):
304     win.fill(BG1_COLOR)
305     time_label = LABEL_FONT.render(f"Time :
{format_time(elapsed_time)}", 1, "white")
306
307     # speed1 = calculate_speed(hit_timestamps[0])
308     # speed2 = calculate_speed(hit_timestamps[1])
309     # speed3 = calculate_speed(hit_timestamps[2])
310
311     # speed1_label = LABEL_FONT.render(f"Player 1 Speed: {speed1}
t/s", 1, "white")
312     # speed2_label = LABEL_FONT.render(f"Player 2 Speed: {speed2}
t/s", 1, "white")
313     # speed3_label = LABEL_FONT.render(f"Player 3 Speed: {speed3}
t/s", 1, "white")
314
315     hits1_label = LABEL_FONT.render(f"Player 1 Hits :
{target1_pressed}", 1, "white")

```

```

316     hits2_label = LABEL_FONT.render(f"Player 2 Hits :
{target2_pressed}", 1, "white")
317     hits3_label = LABEL_FONT.render(f"Player 3 Hits :
{target3_pressed}", 1, "white")
318
319     damages1_label = LABEL_FONT.render(f"Player 1 Damages :
{misses1}", 1, "white")
320     damages2_label = LABEL_FONT.render(f"Player 2 Damages :
{misses2}", 1, "white")
321     damages3_label = LABEL_FONT.render(f"Player 3 Damages :
{misses3}", 1, "white")
322
323     accuracy1 = round(target1_pressed / clicks[0] * 100, 1) if
clicks[0] > 0 else 0.0
324     accuracy2 = round(target2_pressed / clicks[1] * 100, 1) if
clicks[1] > 0 else 0.0
325     accuracy3 = round(target3_pressed / clicks[2] * 100, 1) if
clicks[2] > 0 else 0.0
326
327     accuracy1_label = LABEL_FONT.render(f"Player 1 Accuracy :
{accuracy1}%", 1, "white")
328     accuracy2_label = LABEL_FONT.render(f"Player 2 Accuracy :
{accuracy2}%", 1, "white")
329     accuracy3_label = LABEL_FONT.render(f"Player 3 Accuracy :
{accuracy3}%", 1, "white")
330
331     misses1_label = LABEL_FONT.render(f"Player 1 Misses : {clicks[0] -
target1_pressed}", 1, "white")
332     misses2_label = LABEL_FONT.render(f"Player 2 Misses : {clicks[1] -
target2_pressed}", 1, "white")
333     misses3_label = LABEL_FONT.render(f"Player 3 Misses : {clicks[2] -
target3_pressed}", 1, "white")
334
335     # Display stats on the screen in the corresponding areas
336     win.blit(hits1_label, (20, 60))
337     win.blit(hits2_label, (450, 60))
338     win.blit(hits3_label, (880, 60))
339     win.blit(damages1_label, (20, 100))
340     win.blit(damages2_label, (450, 100))
341     win.blit(damages3_label, (880, 100))
342     win.blit(accuracy1_label, (20, 140))
343     win.blit(accuracy2_label, (450, 140))
344     win.blit(accuracy3_label, (880, 140))
345     # win.blit(speed1_label, (20, 180))
346     # win.blit(speed2_label, (450, 180))
347     # win.blit(speed3_label, (880, 180))
348     win.blit(misses1_label, (20, 180))
349     win.blit(misses2_label, (450, 180))
350     win.blit(misses3_label, (880, 180))
351
352     # Calculate the winner
353     winner = max(
354         [(target1_pressed, "Player 1"), (target2_pressed, "Player 2"),
(target3_pressed, "Player 3")],
355         key=lambda x: x[0]
356     )
357
358     winner_label = LABEL_FONT.render(f"Winner: {winner[1]}!", 1,
"yellow")

```

```

359     winner_stats_label = LABEL_FONT.render(f"Hits: {winner[0]},
Accuracy: {accuracy1 if winner[1] == 'Player 1' else accuracy2 if
winner[1] == 'Player 2' else accuracy3}%", 1, "yellow")
360
361     # Display winner at the center of the screen
362     win.blit(winner_label, (WIDTH / 2 - winner_label.get_width() / 2,
HEIGHT / 2 - 100))
363     win.blit(winner_stats_label, (WIDTH / 2 -
winner_stats_label.get_width() / 2, HEIGHT / 2 + 20))
364
365     # Restart instruction
366     restart_label = LABEL_FONT.render("Press R to Restart", 1,
"white")
367     win.blit(restart_label, (WIDTH / 2 - restart_label.get_width() /
2, HEIGHT / 2 + 60))
368
369     pygame.display.update()
370
371     # Tunggu input pengguna
372     while True:
373         for event in pygame.event.get():
374             if event.type == pygame.QUIT:
375                 pygame.quit()
376                 quit()
377             if event.type == pygame.KEYDOWN:
378                 if event.key == pygame.K_r: # Restart
379                     return # Keluar dari layar akhir untuk memulai
ulang game
380                 elif event.key == pygame.K_q: # Quit
381                     pygame.quit()
382                     quit()
383
384 def get_middle(surface):
385     return WIDTH / 2 - surface.get_width() / 2
386
387 def check_collision(targets, x, y, radius, min_radius=MIN_RADIUS,
max_radius=MAX_RADIUS):
388     for target in targets:
389         if target.collide(x, y, radius):
390             targets.remove(target)
391             return True
392     # if distance <= target.size + radius and min_radius <= radius
<= max_radius:
393         #         targets.remove(target)
394         #         return True
395     return False
396
397 def update_targets(targets, misses):
398     for target in targets:
399         target.update()
400         if target.size <= 0:
401             targets.remove(target)
402             misses += 1
403     return misses
404
405 def main():
406     # Reset semua variabel
407     run = True
408     targets1, targets2, targets3 = [], [], []
409     detected_circles = []

```

```

410     target1_pressed, target2_pressed, target3_pressed = 0, 0, 0
411     clicks = [0, 0, 0]
412     misses1, misses2, misses3 = 0, 0, 0
413     start_time = time.time()
414     hit_timestamps = {0: [], 1: [], 2: []}
415     last_detected = {0: 0, 1: 0, 2: 0}
416     clock = pygame.time.Clock()
417
418     pygame.time.set_timer(TARGET_EVENT, TARGET_INCREMENT)
419
420     while run:
421         clock.tick(60)
422         elapsed_time = time.time() - start_time
423         success, img = cap.read()
424         if not success:
425             break
426
427         scale_factor = 3 # Adjust this if needed
428         output_width = WIDTH // scale_factor
429         output_height = HEIGHT // scale_factor
430
431         img = warpImage(img, circles, output_width, output_height)
432         img = cv2.resize(img, (WIDTH, HEIGHT))
433
434         # img = cv2.resize(img, (WIDTH, HEIGHT))
435         results = model(img, stream=True)
436
437         for r in results:
438             boxes = r.boxes
439             for box in boxes:
440                 x1, y1, x2, y2 = box.xyxy[0]
441                 x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
442                 center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
443                 cls = int(box.cls[0]) # Color class (0: pink, 1:
green, 2: yellow)
444                 conf = float(box.conf[0])
445                 width = x2 - x1
446                 height = y2 - y1
447
448                 # Calculate the radius of the detected bounding box
449                 radius = min(width, height) / 2
450
451                 # Check if the radius is within the valid range
452                 if conf > 0.75:
453                     if MIN_RADIUS <= radius <= MAX_RADIUS:
454                         current_time = time.time()
455
456                         # Scale positions for pygame if necessary
457                         scaled_radius = int(radius * (WIDTH /
cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
458                         scaled_x = int(center_x * (WIDTH /
cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
459                         scaled_y = int(center_y * (HEIGHT /
cap.get(cv2.CAP_PROP_FRAME_HEIGHT))) # Assuming uniform scaling
460
461                         # Check if the detection cooldown has passed
462                         if current_time - last_detected[cls] >=
DETECTION_DELAY:
463                             # detected_circles.append((scaled_x,
scaled_y, scaled_radius, cls, conf))

```

```

464 last_detected[cls] = current_time #
Update last detection time
465 # Save the current frame as an image
466 timestamp = time.time()
467
468 # Process the detection based on the class
469 if cls == 0: # Pink ball for target1
470     filename =
f"detected_pink_{timestamp:.2f}.png"
471     clicks[cls] += 1
472     if check_collision(targets1, (x1 + x2)
// 2, (y1 + y2) // 2, radius):
473         target1_pressed += 1
474     elif cls == 1: # Yellow ball for target2
475         filename =
f"detected_yellow_{timestamp:.2f}.png"
476         clicks[cls] += 1
477
hit_timestamps[cls].append(current_time)
478     if check_collision(targets2, (x1 + x2)
// 2, (y1 + y2) // 2, radius):
479         target2_pressed += 1
480     elif cls == 2: # Green ball for target3
481         filename =
f"detected_green_{timestamp:.2f}.png"
482         clicks[cls] += 1
483
hit_timestamps[cls].append(current_time)
484     if check_collision(targets3, (x1 + x2)
// 2, (y1 + y2) // 2, radius):
485         target3_pressed += 1
486         # cv2.imwrite(filename, img) # Save the
image
487
488         # Draw bounding box
489         cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
490         # Add label with class name and confidence
491         label = f"{model.names[cls]} {conf:.2f} {radius:.1f}"
492         cv2.putText(img, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
493
494         # Add FPS to the image
495         fps = int(clock.get_fps())
496         cv2.putText(img, f"FPS: {fps}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
497
498         # Show the image
499         cv2.imshow("Camera Output", img)
500
501         for event in pygame.event.get():
502             if event.type == pygame.QUIT:
503                 run = False
504                 break
505
506             if event.type == pygame.KEYDOWN:
507                 if event.key == pygame.K_r: # Jika tombol R ditekan
508                     return # Keluar dari loop `main()` untuk memulai
ulang game
509
510             if event.type == TARGET_EVENT:

```

```

511         x = random.randint(TARGET_PADDING, int(WIDTH/3) -
TARGET_PADDING)
512         y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
513         if len(targets1) <= 0:
514             target1 = Target(x,y)
515             targets1.append(target1)
516
517         x = random.randint(int(WIDTH/3) + TARGET_PADDING,
int((WIDTH/3)*2) - TARGET_PADDING)
518         y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
519         if len(targets2) <= 0:
520             target2 = Target(x,y)
521             targets2.append(target2)
522
523         x = random.randint(int((WIDTH/3)*2) + TARGET_PADDING,
int((WIDTH/3)*3) - TARGET_PADDING)
524         y = random.randint(TARGET_PADDING + TOP_BAR_HEIGHT,
HEIGHT - TARGET_PADDING)
525         if len(targets3) <= 0:
526             target3 = Target(x,y)
527             targets3.append(target3)
528
529         misses1 = update_targets(targets1, misses1)
530         misses2 = update_targets(targets2, misses2)
531         misses3 = update_targets(targets3, misses3)
532
533         if misses1 >= LIVES or misses2 >= LIVES or misses3 >= LIVES or
target1_pressed >= HITS or target2_pressed >= HITS or target3_pressed >=
HITS:
534             end_screen(WIN, elapsed_time, target1_pressed,
target2_pressed, target3_pressed, misses1, misses2, misses3, clicks) #
end game
535         return
536
537         draw(WIN, targets1, targets2, targets3, detected_circles)
538         draw_top_bar(WIN, elapsed_time, target1_pressed, misses1,
target2_pressed, misses2, target3_pressed, misses3)
539         pygame.display.update()
540
541         if cv2.waitKey(1) & 0xFF == ord('q'):
542             break
543
544         cap.release()
545         cv2.destroyAllWindows()
546         pygame.quit()
547
548 if __name__ == "__main__":
549     while True:
550         main_menu() # Menu utama sebelum game dimulai
551         main() # Panggil kembali `main()` untuk memulai ulang game

```

File Main.py