# Performance Analysis of YOLOv5 for Real-Time Object Detection and Manipulation in Unity on CPU and GPU

Kankipati Venkata Meghana*, Kondakindi Supriya*,
Korumilli Siva Kumar*, Mula Venkata Surya Mahendra*,
Lekshmi C. R.*
*Amrita School of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore, India
Emails: {cb.sc.u4aie24022, cb.sc.u4aie24025, cb.sc.u4aie24027, cb.sc.u4aie24033}@cb.students.amrita.edu,
cr_lekshmi@cb.amrita.edu

***Abstract*—Real-time object detection and interaction are essential for various applications, including human-computer interaction (HCI), augmented reality, and robotics. This study examines the efficiency of object-based interaction using an entry-level GPU (NVIDIA GTX 1050) and a CPU-only setup (Intel Core i7-9750H) with an inbuilt webcam. The pretrained YOLOv5s model was used for real-time object detection, and performance was assessed in terms of frame rate (FPS), inference speed, and responsiveness. Experimental results indicate that GPU acceleration significantly enhances performance, achieving 12 FPS compared to 8 FPS on the CPU. Without using asynchronous programming in Unity, FPS was limited to 5 on both setups, but implementing it within the Unity's C# script increased FPS to 12 on the GPU and 8 on the CPU. This demonstrates that software optimizations can improve real-time performance even on resource-limited hardware. A comparative analysis with prior studies highlights the trade-offs between different YOLO versions and hardware configurations. Our findings suggest that real-time object interaction can be efficiently achieved on entry-level hardware, making cost-effective HCI solutions feasible. This research contributes to the development of interactive applications by optimizing computational resources without requiring high-end systems.**

***Index Terms*—Unity, YOLOv5,Object Manipulation, Real-time Detection, Human-Computer Interaction (HCI), async, TCP Communication, Virtual Interaction**

## I. Introduction

Object detection plays a significant role in enabling computers to analyze and interpret visual data, facilitating various applications, particularly in human-computer interaction (HCI). With the recent advent of powerful computers, we recognize human actions and interact accordingly, thus revolutionizing the way we interact with computers [12]. Conventional methods of HCI predominantly rely on peripherals such as keyboards, mouse, and touchscreens. Meanwhile, immersive, hands-free interactions often necessitate additional hardware such as controllers, specialized sensors, or high-performance computing resources. These approaches, although effective, impose limitations in terms of accessibility and cost.

*Corresponding author: Lekshmi C.R. (Email: cr_lekshmi@cb.amrita.edu)

In this paper, we propose an alternative technique that utilizes YOLOv5s for real-time object detection, followed by object manipulation in a three-dimensional (3D) virtual environment. Our objective is to achieve near real-time responsiveness by employing asynchronous programming, which helps minimize latency and ensures a smooth user experience. Unlike existing VR-based object detection methods, which are largely confined to gaming and require external hardware, our approach operates with minimal computational resources, relying only on an entry-level GPU and an integrated webcam.

A major advantage of our method lies in its ability to facilitate intuitive, hands-free interaction without the need for additional equipment. By mapping detected objects to specific actions within Unity, we introduce an innovative interaction mechanism that extends beyond gaming into areas such as virtual prototyping, educational simulations, and accessibility solutions. Real-time performance is essential for such applications, and by incorporating asynchronous programming, we aim to optimize computational efficiency, ensuring minimal latency during interactions.

The novelty of our approach is rooted in its lightweight implementation, employing a pretrained YOLOv5s model alongside a TCP socket connection to integrate object detection with Unity-based manipulations. This framework demonstrates the feasibility of real-time, cost-effective object-based interactions and can be further adapted for web-based applications and IoT-enabled systems. The proposed system has diverse applications. In gaming and virtual reality, it provides an alternative interaction mechanism that does not require additional hardware, allowing users to control in-game elements using simple object-based gestures. Beyond the entertainment industry, this method has potential applications in assistive technologies, offering individuals with motor impairments an intuitive means of interacting with computers. Furthermore, it enhances smart environments by enabling natural control over IoT devices and can be incorporated into educational platforms, where gesture-based interactions can improve engagement in learning simulations. By integrating Python with HTML instead of

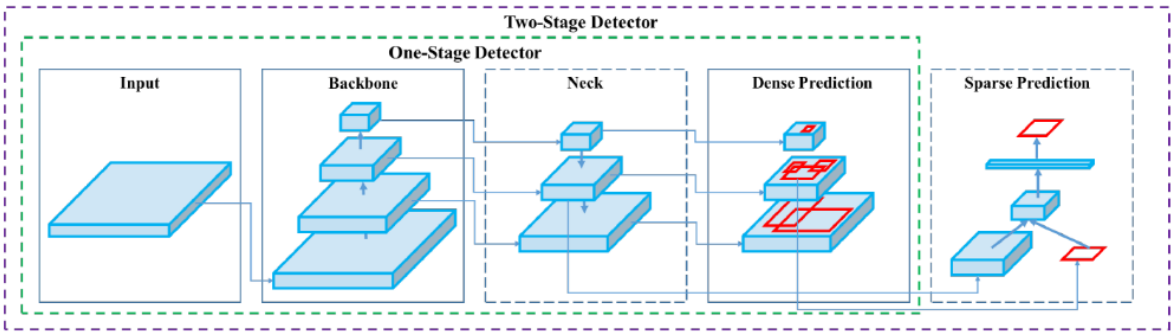


Fig. 1: YOLOv5s object detection process [6]

Unity, the system can also be extended for gesture-based navigation on web-based platforms, further broadening its practical scope.

Despite its advantages, the approach presents several challenges that must be addressed to ensure optimal performance. One primary concern is maintaining real-time responsiveness, particularly on lower-end hardware. While asynchronous programming helps optimize computational efficiency, achieving consistently smooth performance without noticeable lag remains a technical challenge. Additionally, the accuracy of YOLO in detecting objects under various lighting conditions, occlusions, and cluttered backgrounds may lead to occasional false detections or gesture misinterpretations. The use of a TCP socket for communication also introduces a minor data transfer overhead, which, if not optimized, could impact system responsiveness. Furthermore, although YOLOv5s provides a solid foundation, extending the system to recognize dynamic and complex gestures may require training custom models to enhance adaptability.

This paper presents a novel approach for real-time object manipulation using YOLOv5s in Unity [14], providing a cost-effective and accessible solution for hands-free interactions. By addressing key challenges, we aim to advance object-based interaction beyond traditional gaming applications, demonstrating its potential in various fields, including education, assistive technology, and smart environments. Our framework lays the foundation for further exploration of real-time computer vision applications, contributing to the evolution of intuitive and seamless human-computer interaction.

## II. Related Work

The ability to manipulate objects in a virtual environment relies heavily on precise object detection and efficient real-time performance. Deep learning models such as YOLO have gained significant attention due to their ability to achieve high detection accuracy with minimal latency. Recent advancements have enabled the integration of these models into 3D engines like Unity, facilitating real-time object manipulation in interactive environments.

Several studies have explored YOLO's potential in various domains. For instance, an AI-powered VR object management system was introduced for real-time 3D object creation, where users could design and manipulate virtual objects interactively [10]. While this system demonstrated effective object control, it required high-end computational resources, limiting its practicality on low-spec devices. Another study integrated YOLOv4 with Microsoft HoloLens for robotic vision applications, showcasing object detection capabilities in augmented reality environments [8]. A real-time communication system between HoloLens and Ubuntu systems to enable real-time object detection using the YOLO model is introduced in [4]. However, the system experienced performance issues, particularly a reduction in frames per second (FPS) due to hardware constraints, making it less suitable for real-time tasks on entry-level systems.

A more recent approach utilized a YOLOv5-based multi-scale gesture recognition model, which improved detection speed and achieved promising results in gesture recognition tasks [11]. Nevertheless, the system faced limitations in adapting to varying lighting conditions, impacting detection consistency. In an effort to compare modern detection models, researchers evaluated YOLOv5, YOLOv6, and YOLOv8 for real-time hand gesture recognition [5]. While all models exhibited strong performance, the requirement for high-quality datasets and substantial computational power remained a notable challenge. In addition to object detection, Unity [14] has been explored as a platform for developing interactive learning environments. For example, a Unity-based educational platform was developed to teach Python programming concepts through game-based learning [3]. While this system successfully improved learner engagement, it required extensive programming knowledge in both Unity and Python for effective implementation.

Despite these advancements, achieving real-time object manipulation with minimal hardware requirements remains a challenge. In this paper, we propose a cost-effective and hardware-efficient approach to real-time object detection and manipulation using YOLOv5s and Unity. Our method enables seamless interaction in a 3D environment without requiring high-end computational resources. The key contributions of our work are:

TABLE I: Hardware and Software Specifications for CPU and GPU-Based Object Detection

| Setup | CPU-Only Setup | Entry-Level GPU Setup |
|---|---|---|
| Laptop Model | Lenovo 83DR | Dell Inspiron 7591 |
| Processor | AMD Ryzen 7 8845HS (16 CPUs, 3.8GHz) | Intel Core i7-9750H (6 cores, 12 threads) |
| RAM | 16GB | 16GB |
| Operating System | Windows 11 Home 64-bit (Build 22631) | Windows 11 Home (Build 22631) |
| Graphics | Radeon 780M Integrated Graphics (No GPU) | NVIDIA GeForce GTX 1050 (3GB VRAM) |
| Python Version | 3.9.13 | 3.10.0 |
| Torch Version | 2.6.0+cpu | 2.5.1+cu118 |
| OpenCV Version | 4.11.0 | 4.10.0 |
| Camera Used | Realtek Laptop Camera | Realtek Laptop Camera |
| Processing Mode | CPU-only | CUDA acceleration for YOLO model |
| Performance (FPS) | 8 FPS | 12 FPS |

- We integrate YOLOv5s for real-time object detection and map detected objects to actions in Unity, allowing dynamic object manipulation without external controllers or additional hardware.
- Unlike existing systems that rely on high-performance GPUs or specialized devices such as VR controllers, our approach operates efficiently on entry-level GPUs and standard webcams, making it more accessible and cost-effective.
- To enhance real-time responsiveness, we implement asynchronous programming that reduce processing latency, ensuring smoother interaction and faster object manipulation in Unity.
- We establish a robust communication bridge between Python (YOLO-based detection) and Unity using TCP sockets, facilitating low-latency data transfer and real-time synchronization.
- While demonstrated within a Unity-based environment, our system can be extended to other human-computer interaction (HCI) applications, including gesture-based control for software interfaces, assistive technologies, and interactive web-based environments.

## III. Methodology

Our system integrates YOLOv5s for real-time object detection with Unity-based object manipulation. A webcam captures frames, which are processed by YOLOv5s in Python, and the detection results are sent to Unity via a TCP socket. Unity interprets these results to execute object movements while using asynchronous data handling to maintain smooth performance. The system is tested on both GPU and CPU setups to ensure adaptability across different hardware configurations. The following sections detail the YOLOv5s architecture and the execution pipeline.

### *A. YOLOv5s Architecture*

YOLOv5s consists of three key components [6]: the **Backbone**, which employs CSPDarknet53 for efficient feature extraction; the **Neck**, which integrates PANet for multi-scale feature fusion; and the **Head**, which utilizes anchor-based detection to predict object classes and bounding boxes. This architecture enables real-time object detection while maintaining computational efficiency and is utilized in a lot of applications like weapons detection for security and video surveillance [2], concrete crack detection [15],multi-scale feature extraction [7] etc. The schematic representation of the object detection process of YOLOv5s is shown in Figure 1.

Among various YOLO variants, we selected YOLOv5s due to its balance between speed, accuracy, and hardware efficiency. Unlike YOLOv4, which requires high computational power, YOLOv5s is optimized for both GPU and CPU environments, making it more suitable for real-time applications [6]. While YOLOv7 and YOLOv8 offer improvements in accuracy, they come at the cost of higher latency and resource demands. YOLOv5s, on the other hand, provides a lightweight yet effective solution with pre-trained models from the Ultralytics repository [1], ensuring fast inference and seamless deployment. These characteristics make it an ideal choice for our interactive system.

### *B. Pipeline Overview*

We first set up the pre-trained YOLOv5s variant on our system, storing it in the cache. The model, implemented using PyTorch and loaded from the Ultralytics repository, runs on GPU (NVIDIA GTX 1050) as one setup and on CPU for the other.

To enable real-time gesture-based interaction, we established a TCP socket connection between Python and Unity. The Python script captures webcam frames using OpenCV, processes them with YOLO, and sends the detection results to Unity. The frame rate is monitored in both GPU and CPU setups to evaluate performance.

In Unity, a 3D environment is created where detected objects trigger movements. Unity has been utilized in a lot of similar applications [13], [1], [9]. A C# script asynchronously receives data from Python and translates object detection outputs into corresponding actions. Quaternion-based motion transitions ensure smooth and realistic interactions.

The system was tested on different hardware configurations, with asynchronous programming in Unity reducing lag and ensuring near real-time movement execution. This approach provides a cost-effective, low-latency solution for interactive

[1]https://github.com/ultralytics

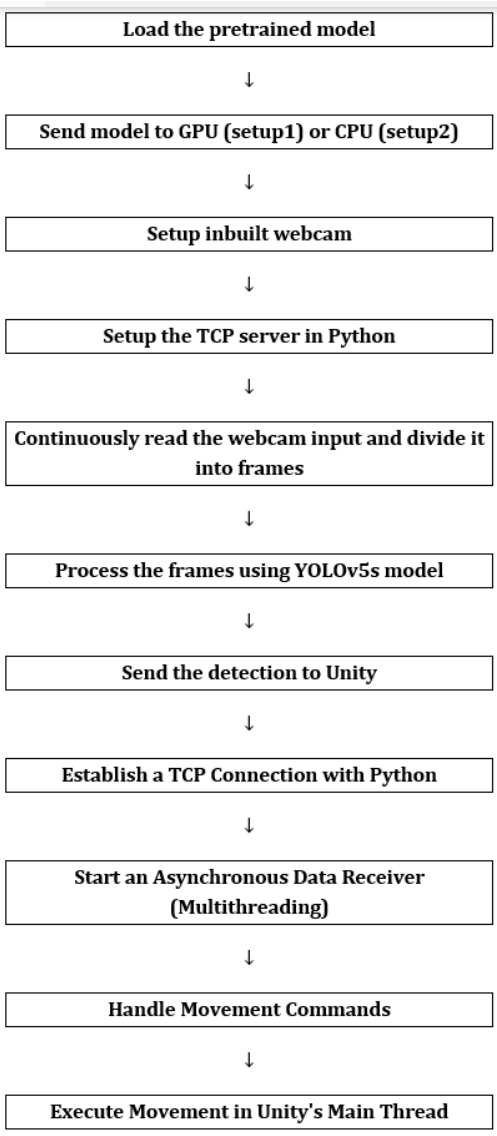


Fig. 2: Flow of Execution

computer vision applications. Figure 2 shows the flow of execution of the entire process.

## IV. Hardware Configurations and Performance Analysis

This study implements a real-time object detection and manipulation system using a pre-trained YOLOv5s model, evaluated on two computational setups: an entry-level GPU system with an NVIDIA GTX 1050 and a CPU-only system featuring an AMD Ryzen 7 8845HS processor. The proposed system integrates OpenCV for real-time image processing and utilizes a TCP socket connection for seamless communication between Python (YOLO) and Unity.

Initially, the YOLOv5s model is loaded using the Ultralytics repository, where the system determines hardware availability. If a CUDA-enabled GPU is detected, computations are offloaded to the GPU to enhance inference speed; otherwise, processing remains on the CPU. The webcam is initialized via OpenCV, and live frames are continuously captured and transmitted to the YOLO model for object detection. A client-server architecture is employed, with Python acting as the server and Unity as the client, facilitating real-time data exchange over a TCP socket bound to `127.0.0.1:5005`.

The Unity implementation features a C# script that establishes a non-blocking, asynchronous connection to the Python server. A dedicated background thread continuously listens for incoming data to ensure uninterrupted processing. Upon receiving detection results, the system extracts the relevant classification label and translates it into navigational commands. Detection of a person triggers forward movement, whereas the absence of a person prompts a right-turn action. The integration of Unity's coroutine-based motion control ensures smooth transitions, mitigating abrupt changes in movement.

For object detection, YOLOv5s is configured with an input image size of 640×640, a confidence threshold of 0.25, and an Intersection over Union (IoU) threshold of 0.45. Experimental results demonstrate that the GPU-accelerated system achieves an average inference speed of 12 frames per second (FPS), enabling real-time interaction with minimal latency. Conversely, the CPU-only setup exhibits a reduced performance of 8 FPS, highlighting the computational limitations of non-GPU environments.

TABLE II: Experimental Setup and Hyperparameters

| Hyper-Parameter | Value |
|---|---|
| Model Variant | YOLOv5s |
| Confidence Threshold | 0.25 |
| IoU Threshold | 0.45 |
| Input Size | 640 × 640 |
| Processing Device | CUDA (GPU) / CPU |
| Host | 127.0.0.1 |
| Port | 5005 |
| Read Timeout | 50 ms |
| Movement Speed | 10f |

### A. Experimental Setup

To evaluate the performance of real-time object detection and manipulation using YOLOv5s in Unity, we implemented our system on two different hardware configurations: one utilizing only a CPU and another employing an entry-level GPU for acceleration. In both cases, the system relied on an inbuilt webcam for video streaming and object detection.

*1) CPU-Only Setup:* In the CPU-based configuration, we used a Lenovo 83DR laptop equipped with an AMD Ryzen 7 8845HS processor featuring 16 logical cores (base clock: 3.8 GHz). The system includes 16GB RAM and operates on Windows 11 Home 64-bit (Build 22631).

As this setup does not include a dedicated GPU, all object detection processes were executed using the CPU. The Realtek laptop camera was used for real-time video input. The following software versions were used in this setup:

- Python 3.9.13
- Torch 2.6.0+cpu
- OpenCV 4.11.0

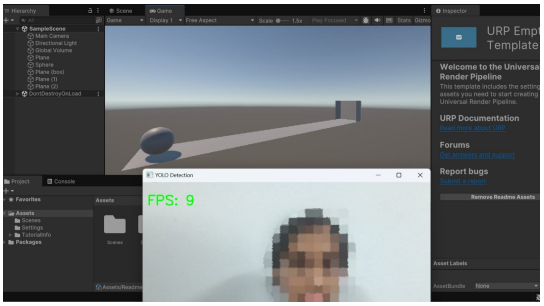


Fig. 3: Detection result in CPU-only setup

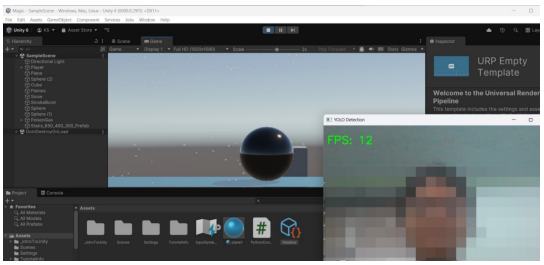


Fig. 4: Detection result of GPU accelerated setup

*2) Entry-Level GPU Setup:* For the GPU-based configuration, we employed a Dell Inspiron 7591 laptop featuring an Intel Core i7-9750H processor (6 cores, 12 threads) and 16GB of RAM, running on Windows 11 Home (Build 22631).

In this setup, the YOLO model was offloaded to an NVIDIA GeForce GTX 1050 GPU (3GB VRAM) to leverage CUDA acceleration. The software versions used in this configuration were:

- Python 3.10.0
- Torch 2.5.1+cu118
- OpenCV 4.10.0

## V. Results and Analysis

The performance of the proposed real-time object detection and manipulation system was evaluated based on frame rate (FPS), inference speed, system responsiveness, and the impact of optimization techniques such as asynchronous programming.

### A. Comparison of CPU and GPU Performance

YOLOv5s achieved an average frame rate of 12 FPS on the GPU, whereas the CPU-only setup yielded 8 FPS. The results highlight the computational burden of deep learning inference when executed without dedicated hardware acceleration. Table III presents a comparative analysis of the system's performance on CPU and GPU setups.

TABLE III: Performance Comparison of CPU and GPU Setups

| Metric | CPU | GPU |
|---|---|---|
| FPS (Avg.) | 8 | 12 |
| Latency (ms) | 110 | 85 |

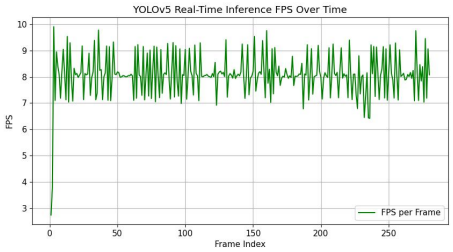


Fig. 5: Real time inference FPS for CPU setup

### B. Latency Analysis

We measured the system's end-to-end latency, which includes:

- Frame Acquisition Time
- Inference Time
- Transmission Delay
- Response Execution Time

The total latency in the GPU setup was approximately 85ms per frame, ensuring near real-time operation, while the CPU setup exhibited a latency of 110ms, leading to perceptible delays in object interaction.

### C. Visualization of Results

Figures 3 and 4 illustrate the object detection performance in CPU and GPU setups, respectively. The CPU-based implementation (Figure 3) operates at 8 FPS, causing detection delays and occasional frame skips, reducing tracking smoothness. In contrast, the GPU-based setup (Figure 4) achieves 12 FPS with a latency of 85ms per frame, ensuring smoother detection and real-time tracking. This highlights the importance of hardware acceleration and software optimizations like asynchronous programming for improved system responsiveness in interactive applications.

Figures 5 and 6 illustrate frames-per-second performance in CPU and GPU setups, respectively, it shows the FPS vs frame index graph (frame index is the nth frame at which the fps is taken). As shown in the figure 5, we can observe an average of 8 fps, explaining the relatively slower speed and higher latency in the case of real-time object detection tasks. While on the other hand, Figure 6 demonstrates better performance with an FPS of 12, indicating the GPU's ability to do faster computations and handle heavy load tasks more efficiently. This shows the advantage of using a GPU for applications where low latency and real-time detection and response are essential.

### D. Inferences

YOLOv5 was chosen for its compatibility with both CPU and GPU, and its native PyTorch support. The YOLOv5s variant was used for a good balance between accuracy and latency—YOLOv5n is faster but less accurate. For lower lag, YOLOv5n can be used; for higher accuracy, larger YOLO variants are better.

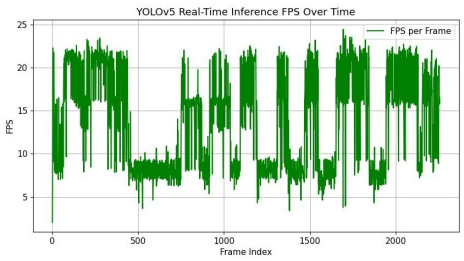


Fig. 6: Real time inference FPS for GPU setup

Reducing image resolution, preprocessing (resizing), and using asynchronous programming to improve real-time performance. Accuracy will be improved by fine-tuning a pretrained model on specific datasets (e.g., gesture datasets). Setting a cap on FPS can help maintain stability during detection.

Using UDP instead of TCP speeds up data transfer between Python and Unity. Profiling tools in both Unity and PyTorch can be used to optimize bottlenecks like detection time and communication.

Inference performance was improved, by using asynchronous programming, which was implemented in Unity using async ReceiveData(), allowing for smooth and non-blocking updates when the data is transferred. FPS stability was maintained by using lightweight models like YOLOv5s and also using images of lesser scaling can degrade accuracy and precision, so this was avoided. Selective inference can help further reduce the processing load by focusing on processing only the necessary input, which was implemented using cv.resize(). This avoids processing the full frame when unnecessary.

## VI. Conclusion

This paper compares real-time object-based interaction on an entry-level GTX 1050 GPU and a CPU-only setup using an inbuilt webcam and YOLOv5s. We demonstrate that using asynchronous programming in Unity C# scripts significantly boosts FPS, improving GPU performance from 5 to 12 FPS. Our cost-effective, hands-free HCI system enables interactive applications on affordable hardware, offering scalability for app and game developers. Beyond gaming, this framework extends to web-based interfaces, AR, and accessibility applications by integrating Python with HTML for gesture-based interactions.

## Acknowledgment

**Ethical Statement:** This research adheres to ethical guidelines for human-computer interaction studies. All individuals whose images appear in the screenshots have provided informed consent for their inclusion and publication in this research. The study complies with data privacy regulations and does not pose any risks to user privacy. Future deployments will adhere to ethical and legal standards, including Institutional Review Board (IRB) approval when necessary.

## References

[1] Víctor Hugo Andaluz, Fernando A Chicaiza, Cristian Gallardo, Washington X Quevedo, José Varela, Jorge S Sánchez, and Oscar Arteaga. Unity3d-matlab simulator in real time for robotics applications. In *Augmented Reality, Virtual Reality, and Computer Graphics: Third International Conference, AVR 2016, Lecce, Italy, June 15-18, 2016. Proceedings, Part I 3*, pages 246–263. Springer, 2016.

[2] Abdul Hanan Ashraf, Muhammad Imran, Abdulrahman M Qahtani, Abdulmajeed Alsufyani, Omar Almutiry, Awais Mahmood, Muhammad Attique, and Mohamed Habib. Weapons detection for security and video surveillance using cnn and yolo-v5s. *CMC-Comput. Mater. Contin*, 70(4):2761–2775, 2022.

[3] S Graceline Jasmine, Sujay Doshi, Alan K Alex, U Yadukrishnan, and Febin Daya JL. A game-based approach to teach basic python programming. *Journal of Engineering Education Transformations*, 38(2), 2024.

[4] Jiazhen Guo, Peng Chen, Yinlai Jiang, Hiroshi Yokoi, and Shunta Togo. Real-time object detection with deep learning for robot vision on mixed reality device. In *2021 IEEE 3rd global conference on life sciences and technologies (LifeTech)*, pages 82–83. IEEE, 2021.

[5] Nourdine Herbaz, Hassan El Idrissi, and Abdelmajid Badri. Deep learning empowered hand gesture recognition: using yolo techniques. In *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–7. IEEE, 2023.

[6] Rahima Khanam and Muhammad Hussain. What is yolov5: A deep look into the internal features of the popular object detector. *arXiv preprint arXiv:2407.20892*, 2024.

[7] Xiao-lin LI, Fu-gang WANG, Peng-fei ZHANG, and Lin-yu ZHANG. Yolov5s algorithm optimization based on multi-scale feature extraction. *Computer Engineering & Science*, 45(06):1054, 2023.

[8] Mikołaj Łysakowski, Kamil Żywanowski, Adam Banaszczyk, Michał R Nowicki, Piotr Skrzypczyński, and Sławomir K Tadeja. Real-time onboard object detection for augmented reality: Enhancing head-mounted display with yolov8. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 364–371. IEEE, 2023.

[9] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John A Castro-Vargas, Sergio Orts-Escolano, and Jose Garcia-Rodriguez. A visually realistic grasping system for object manipulation and interaction in virtual reality environments. *Computers & Graphics*, 83:77–86, 2019.

[10] Sanzhar Otkilbayev, Madina Ipalakova, Dana Tsoy, and Yevgenia Daineko. Development of an object management system in virtual reality. In *International Conference on Extended Reality*, pages 267–275. Springer, 2024.

[11] Ledan Qian, Xiao Zhou, Xuankang Mou, and Yi Li. Multi-scale tiny region gesture recognition towards 3d object manipulation in industrial design. In *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pages 369–376. IEEE, 2021.

[12] Kaushik Ranade, Tanmay Khule, and Riddhi More. Object recognition in human computer interaction:-a comparative analysis. *arXiv preprint arXiv:2411.04263*, 2024.

[13] Alvaro Villegas, Pablo Perez, Redouane Kachach, Francisco Pereira, and Ester Gonzalez-Sosa. Realistic training in vr using physical manipulation. In *2020 IEEE conference on virtual reality and 3D user interfaces abstracts and workshops (VRW)*, pages 109–118. IEEE, 2020.

[14] Jingming Xie. Research on key technologies base unity3d game engine. In *2012 7th international conference on computer science & education (ICCSE)*, pages 695–699. IEEE, 2012.

[15] Zhen Yu. Yolo v5s-based deep learning approach for concrete cracks detection. In *SHS Web of Conferences*, volume 144, page 03015. EDP Sciences, 2022.