Acta Mechanica Slovaca
ISSN 1335-2393
www.actamechanica.sk

Use of Gaming Engines for Robotic Simulations and Computer Vision

Leo Brada [1,*], Marek Málik [1], Erik Prada [1] and Ľubica Miková [1]

[1] Technical University of Kosice, Faculty of Mechanical Engineering, Department of Industrial Automation and Mechatronics, Park Komenského 8, 042 00 Košice

Abstract: Gaming engines have evolved beyond their traditional role in entertainment, becoming powerful tools for robotic simulations and computer vision applications. Their ability to generate high-fidelity, real-time simulations makes them valuable for developing and testing robotic systems in virtual environments. The experiment demonstrated the use of the Unreal Engine, and its computer vision plugin called UnrealCV. A realistic factory environment was created in order to test the capability of the computer vision plugin. The article evaluates UnrealCV and its features along with their potential use in future work. Data from depth cameras are visualized in the form of 3D mesh and Yolo v5s AI image recognition software was tested.

Keywords: gaming engine, robotics, Unreal Engine, UnrealCV

1. Introduction

In recent years, the rapid advancement of gaming technology has transcended beyond entertainment, finding significant applications in various fields such as architecture, automotive, simulation, robotics, and computer vision. The gaming engine is a software framework mainly used for video game development. It was created as a platform for developers to create games faster by including relevant libraries, plugins, tools, and editors. Immersive environments, real-time physics simulations, implementation of artificial intelligence (AI), and machine learning are very powerful features not only for gaming development but also for robotics. Modern engines such as Unity and Unreal Engine are always innovating and implementing new powerful features to further elevate the usability of the engines. Adaptability, usability, and almost unlimited scenarios are the benefits of using the gaming engine for robotic simulations. Academic researchers are creating many plugins and tools to incorporate robotic systems into gaming engines and create digital twins. The creation of high-fidelity and photorealistic environments is greatly useful for AI and computer vision (CV) training. Because of their powerful rendering capabilities, high fidelity and photorealistic environments can be created thus allowing engines to create a synthetic dataset for robot navigation, object detection, multiple robot interaction, and computer vision. In robotics computer vision is a system that allows robots to interpret visual and sensory data and understand their surroundings. CV is used everywhere in robotics from simple ultrasonic distance sensors to RGBD cameras and LiDAR (Light detection and ranging) sensors. These sensors combined with AI and complex algorithms will allow robots to detect objects in the environment, create a visual map of the environment, and react adequately to the situation. In order to teach robots how to recognize their surroundings it is needed to train the AI to control the robot's movements and actions. Training can be achieved either by letting the robot move in the real environment and gather data or by simulating the environment and training the robot virtually.

*** Corresponding author:** Leo Brada, ***E-mail address:*** *leo.brada@tuke.sk*

Robot simulation is very beneficial for simulating CV because contrary to the real-world environment, the robot cannot be damaged by the dynamic environment, and a variety of environments and situations can be simulated.

2. Practical Use

Unity and Unreal Engine were concluded to be most suitable for robotic simulation [1]. The main difference between the two is the use cases. Unity is better used for mobile games, 2D projects, and AR (augmented reality) applications. In general, Unity is used for lightweight applications. On the other hand, the Unreal engine is used for high-end games and simulations, photorealistic scene rendering, and VR (virtual reality) applications. The significant drawback of gaming engines is their hardware requirements. Although scene rendering is highly optimized, the physics simulation can still be computationally intensive which can limit the scale of simulation. Unity is ideal if the hardware running the simulation is not powerful. Unreal engine requires more computational power but offers more complex features such as ray tracing and Lumen technology. Inverse and forward kinematics are also very beneficial for simulating complex mechanisms. In comparison to industry standard simulators such as Gazebo, Webots, and NVIDIA Isaac Sim, gaming engines offer robust simulation environments with realistic physics and photorealistic graphics. Gaming engines are also widely available without the need for an expensive license and offer extensive documentation. Due to large communities using the engines, many custom projects, plugins, and sample projects were created, thus the creation process can be drastically accelerated. When talking about robotic systems, the increasingly popular middleware called Robot Operating System (ROS) is used. This middleware allows components of a robot such as actuators and sensors to communicate with each other. Gaming engines cannot communicate with ROS out of the box. The popularity of game engines in robotic simulation allowed the creation of communication interfaces between the engine and ROS system [2]. Bi-directional communication can be established and used to create digital twins, teleoperation, and machine learning. For example, Unity was used as a platform for monitoring robotic welders utilizing Rosbridge as a communication protocol between ROS and Unity [3]. Robotic teleoperation can also be achieved in Unity and Unreal Engine. Teleoperation of robotic arms [4], hands [5], drones, unmanned ground vehicles [6], and a lot more. Complicated terrain such as the sea floor and underwater environment can be created to simulate submarine movement and navigation [7]. Drone simulations are also very useful for control and navigation in the environment even for the detection of obstacles at high speeds [8]. AirSim created by Microsoft is used for drone and autonomous vehicle simulation utilizing CV [9]. CV is often simulated by the OpenCV [10] framework which is an open-source library. For the Unreal engine, a custom UnrealCV plugin was created based on the OpenCV [11]. Dynamic environment can also be achieved for testing CV and more dynamic robot navigation algorithms for example in Unreal Engine by utilizing the Chaos physics system [12].

3. Experimental Work

For this experiment, the Unreal Engine (UE) was chosen as the simulation environment. The UnrealCV plugin was tested and experimented with to explore the possible limitations [13]. Firstly, it is necessary to choose the version of UE because the plugin was created for the latest versions of UE4, but one version was created for UE5.2. This version was used and experimented with. The installation process was simple but required manual rebuilding of the project in Visual Studio. After the successful rebuilding of the project, the plugin was active and ready. To add immersion to the simulation the 3D model of a robot was downloaded. The chosen robot was Jethexa by Hiwonder which is a hexapod robot powered by an NVIDIA Jetson controller. The model is this robot was acquired and imported into the UE. A third-person template was used to speed up the process of setting up the control system. The CV system will be applied to any active camera when playing in the editor. Since the default pawn has a camera, this was used for the CV. UnrealCV works by writing commands into the UE command line while the simulation runs. The command for changing the camera mode is "vset /viewmode [mode]". As shown in Figure. 1 there are 4 modes used: lit (standard view), depth, normal, and object_mask. Apart from object_mask, all other modes worked well. Object_mask is supposed to execute object segmentation and assign every object to a different color, but this

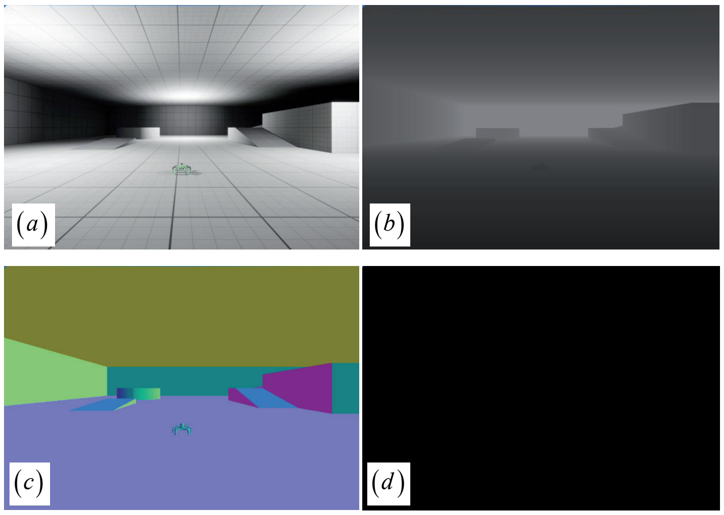

Figure 1: Pictures of simulation modes: a) lit, b) depth, c) normal, d) object_mask.

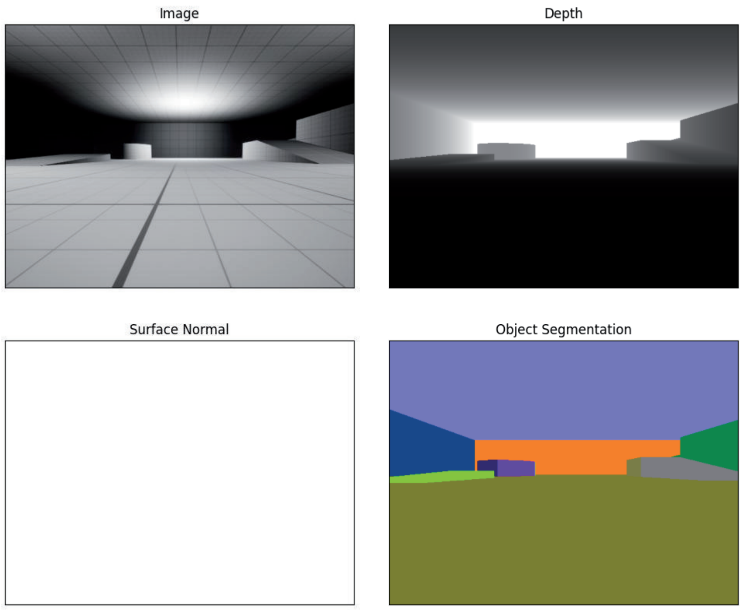

Figure 2: Computer vision outputs generated by the Python script.

didn't work on this default map. On different maps, the object segmentation worked properly. Depth mode is working as intended but also unrealistically because of the unlimited range of the camera. This was later modified. Otherwise, the simulation works as expected.

UnrealCV offers Python scripts for the remote connection and extraction of camera data. Firstly, the IP and port of the simulation were required for connection. UnrealCV automatically creates the communication port for the Python client to connect to. The default IP address of the UE project is 127.0.0.1 and the port was set to 9000. The command "vget /unrealcv/status" was used to display the port of the simulation. Python scripting offers many possibilities of use such as creating and modifying cameras, data gathering and processing, spawning, destroying, and modifying objects in the simulation environment, and more. Python script utilizes the Matplotlib library to display the output of the CV. As shown in Figure 2 multiple camera modes can be displayed simultaneously using the subplot function. This approach is very effective and allows further data processing. Object segmentation works in the default map but the normals are not displayed.

The camera used by UnrealCV is called FusionCamSensor. If this specific camera is not present in the simulation environment either inside an actor or as an object in 3D space the Python script will spawn one. The spawned camera will be placed exactly where the main pawn camera is located at the time of connecting to the simulation. As shown in Figure. 2 the camera is placed lower to the ground because the robot has the FusionCamSensor on top of it thus the Python script doesn't need to spawn a new one. This is important because robots need to move in the environment and gather data. Because the Python script gathers the depth data in the form of a float array. Float arrays can be modified to give the sensor distance limit by using the numpy.clip command. In this manner the range of the camera can be limited to, for example, 7.5 m as in Figure. 3. 1 cm is equal to 1 unreal unit. Data from the depth sensor can be extracted and visualized in MATLAB using the Scipy library. The Savemat command allows saving the data in a MATLAB-style .mat file. The mesh function can load and visualize this file in a 3D mesh surface. To properly match the camera output the created mesh was mirrored with the "fliplr" command. Figures 4 and 5 show the flipped 3D mesh in MATLAB. The X and Y axis displays the pixel placement of the data, and the Z axis displays

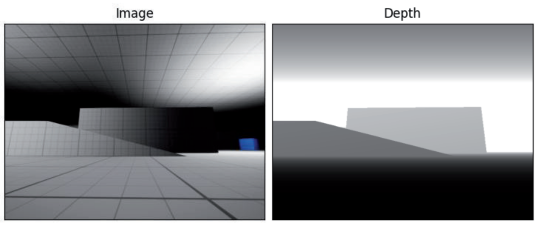

Figure 3: Depth camera with limited range.

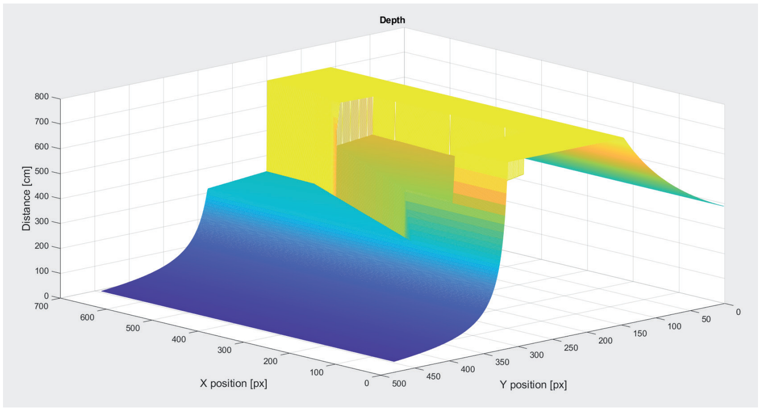

Figure 4: Mesh of depth variables from diagonal view.

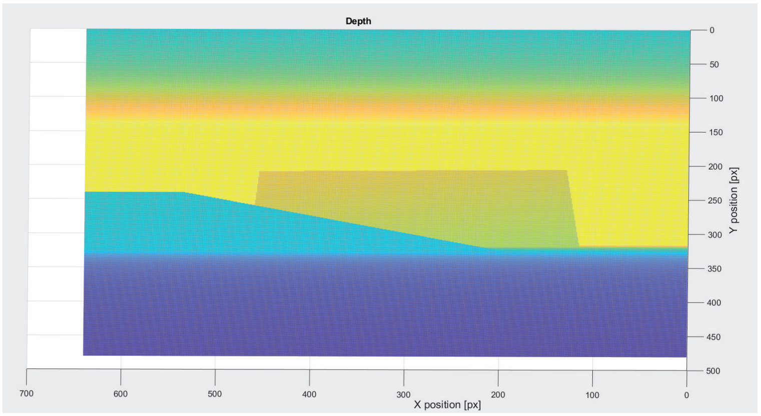

Figure 5: Mesh of depth variables from a top view.

the distance from the sensor. 7.5 meters was the limit of the camera thus this distance is coloured yellow. The graph colour changes to blue as the distance decreases.

The big advantage of using UE is the ability to create photorealistic environments due to the robust rendering and realistic lighting. Combined with free-to-use highly detailed models and photorealistic textures, it is possible to recreate real-life locations and simulate complex environments. For example, in Figure. 6 the factory environment with assembly lines and robotic welders. In Figure. 7 multiple photos of the environment are present along with different lighting conditions. Unreal Engine can realistically simulate lighting and shadows with the option to use ray tracing to achieve more realistic environments. The possibilities are virtually endless. In Figure. 8 the output of depth camera in the factory environment is displayed.

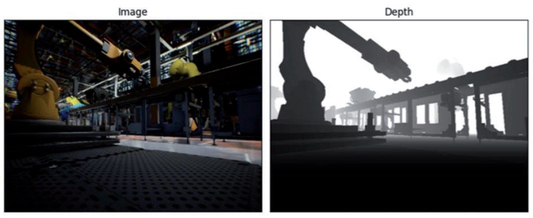

Figure 8: Depth camera in the factory environment.

As shown in Figure. 9, one problem that occurred with object segmentation is the process of segmentation based on the actors of the scene. The floor is composed of many square actors thus the segmentation coloured them all different colours. Similarly, in office spaces depending on the scene creation the table, chairs, and accessories are either segmented separately or as one solid object. This might be problematic in the future thus great care must be taken in creating the environments. Image

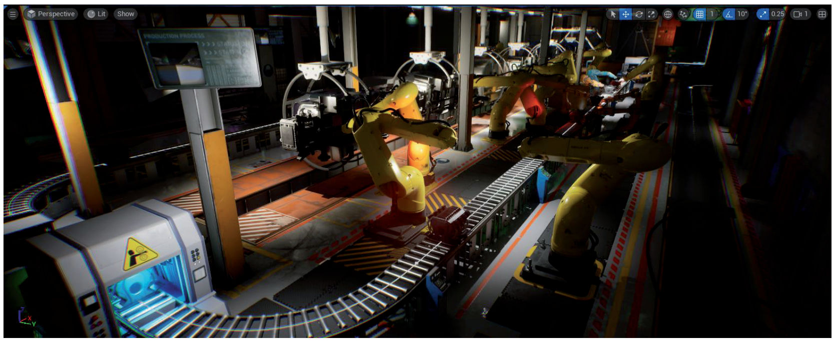

Figure 6: Photorealistic environment of the factory.

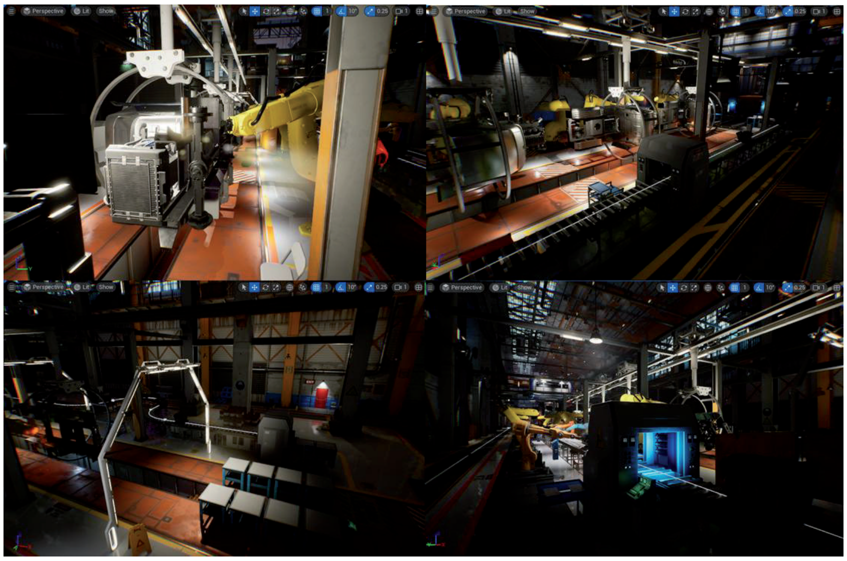

Figure 7: Factory environment from different angles.

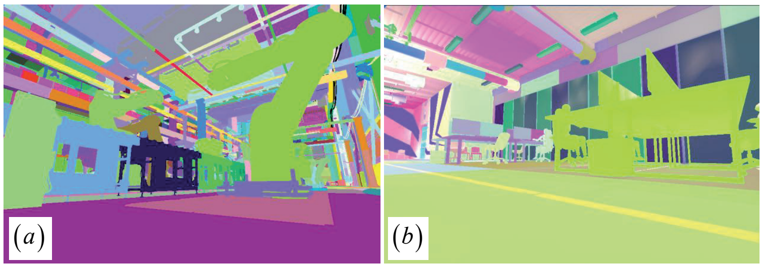

Figure 9: Object segmentation in the a) factory and b) office environment.

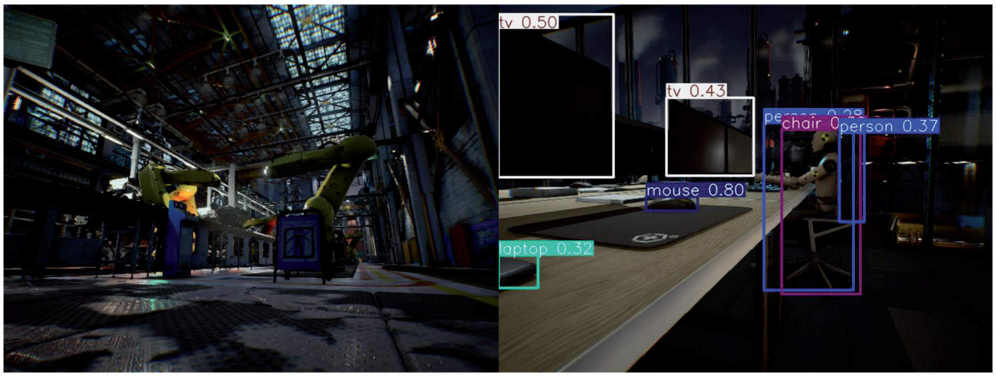

Figure 10: Yolo v5 image recognition in factory and office environments.

recognition was also tested via the Ultralytics Yolo vision AI. Output data from the cameras are fed directly into the AI via the Python script. This allows users to further improve the system by allowing it to recognize objects, patterns, and more. It can also be trained to the needs of the simulation. Pre-trained Yolo v5s [14] was tested although the factory environment is not suited and needs to be trained for the environment. The office environment was better suited for the AI. As shown in Figure. 10, the AI could recognize objects in an office environment such as a mouse, chair, person, and computer monitors. Also, computer vision can be used in combination with ROS to create a virtual map of the robot's surroundings thus making him understand where he is currently and what are his surroundings. Visual SLAM (simultaneous localization and mapping) can be used by implementing communication with ROS by sending the data from the CV to the SLAM algorithms. This approach is great for testing the robot's navigation and perception and will be experimented on in the future,

4. Conclusions

Robotic simulation and computer vision are essential parts of developing robotic systems. Game engines are used in the industry not only for visualization but for robotic simulations too. By leveraging photorealistic rendering, real-time physics, and synthetic data generation they improve robotic simulations and offer more possibilities for realistic computer vision simulations. The use of Unity and Unreal engine were discussed. Unreal engine was used as the simulation environment due to its high-fidelity rendering, realistic lighting, and implementation of the UnrealCV plugin. UnrealCV proved to be a suitable plugin for computer vision in the virtual environment and will be used in the future for robotic navigation in the virtual environment. MATLAB was used to visualize the depth camera data in 3D mesh for a simple understanding of distances from the camera. Problems with object segmentation occurred but the attention to the creation process of the environment negates it. AI implementation is also possible and will be very beneficial for future testing. Pre-trained Yolo v5s algorithm was used but for the warehouse and factory environment, it needs to be retrained.

Acknowledgments

The authors would like to thank the Slovak Grant Agency projects KEGA 008TUKE-4/2024 named Implementation of machine learning methods in the teaching of industrial automation and robotics, and VEGA 1/0409/25 named Unconventional and

soft robotic structures which was supported by the Ministry of Education of the Slovak Republic.

References

1. Zarco, L., Siegert, J., Schlegel, T. and Bauernhansl, T., 2021. Scope and delimitation of game engine simulations for ultra-flexible production environments. Procedia CIRP, 104, pp.792-797.
2. Whitney, D., Rosen, E., Ullman, D., Phillips, E. and Tellex, S., 2018, October. Ros reality: A virtual reality framework using consumer-grade hardware for ros-enabled robots. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1-9). IEEE.
3. Sita, E., Horváth, C.M., Thomessen, T., Korondi, P. and Pipe, A.G., 2017, December. ROS-Unity3D based system for monitoring of an industrial robotic process. In 2017 IEEE/SICE International Symposium on System Integration (SII) (pp. 1047-1052). IEEE.
4. Shamaine, C.X.E., Qiao, Y., Henry, J., McNevin, K. and Murray, N., 2020, September. RoSTAR: ROS-based telerobotic control via augmented reality. In 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP) (pp. 1-6). IEEE.
5. Martinez-Gonzalez, P., Oprea, S., Garcia-Garcia, A., Jover-Alvarez, A., Orts-Escolano, S. and Garcia-Rodriguez, J., 2020. Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. Virtual Reality, 24, pp.271-288.
6. Zaman, N., Tavakkoli, A. and Papachristos, C., 2020, April. 'Tele-robotics via an efficient immersive virtual reality architecture. In Proc. 3rd Int. Workshop Virtual, Augmented, Mixed Reality HRI.
7. Chaudhary, A., Mishra, R., Kalyan, B. and Chitre, M., 2021, September. Development of an underwater simulator using unity3d and robot operating system. In OCEANS 2021: San Diego–Porto (pp. 1-7). IEEE.
8. Meng, W., Hu, Y., Lin, J., Lin, F. and Teo, R., 2015, November. ROS+ unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments. In IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society (pp. 002562-002567). IEEE.
9. Shah, S., Dey, D., Lovett, C. and Kapoor, A., 2018. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and Service Robotics: Results of the 11th International Conference (pp. 621-635). Springer International Publishing.
10. OpenCV team, OpenCV, from https://opencv.org/
11. Qiu, W. and Yuille, A., 2016. Unrealcv: Connecting computer vision to unreal engine. In Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14 (pp. 909-916). Springer International Publishing.
12. Chaudhary, A., Tiwari, K. and Bera, A., 2023. HEROES: Unreal Engine-based Human and Emergency Robot Operation Education System. arXiv preprint arXiv:2309.14508.
13. Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Kim, T. S., & Wang, Y. (2017, October). Unrealcv: Virtual worlds for computer vision. In Proceedings of the 25th ACM international conference on Multimedia (pp. 1221-1224).
14. Jocher, Glenn, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang et al. "ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation." Zenodo (2022).