

Synthetic Data Generation for Computer Vision based Autonomous Landing for Small UAS Package Delivery

Daniel Ding^{*1}, Can Vu^{†1}, Abenezer Taye[‡], Amin Tabrizian[§], Peng Wei[¶], and Zhenhao Zhao^{||}
George Washington University, Washington, DC, 20052

Small unmanned aerial systems (UAS) are showing great promise for commercial package delivery. Zipline and Google Wing have developed and are operating small UAS fleets for delivery in the United States and internationally. Deep learning based computer vision model on UAS platforms for recognition of the objects near landing zone remains an open area of research, such as pedestrians, pets, cars, etc. Deep learning, neural network based computer vision systems supporting autonomous landings require large amounts of data to train and validate the models. Flight tests are often conducted for video data collection of specific landing or delivery scenarios of interest. However, running real-world flight test experiments are often laborious and costly, not to mention possible regulatory constraints depending on the locations of flight test experiments. In addition, it is extremely challenging to use flight tests to capture a wide of range of landing scenes under different seasonal, atmospheric and lighting conditions, such as evening or low-light, fog, light rain, air pollution, etc. Such challenges motivated us to investigate synthetic data generation using high-fidelity simulation frameworks. This paper reports our approach for modeling realistic landing zone environments using Microsoft AirSim, an open-source Unreal Engine-based simulation platform popular for artificial intelligence (AI) research and experimentation. Also reported herein is our process for scenario specification and generation of realistic dynamic and/or static landing scenes with tunable parameters including weather effects, time-of-day lighting, and building/environmental geometry with respect to the landing area. The synthetic flight data is used for YOLOv8 model evaluation to show that the vision model can make accurate obstacle detections in a simulated landing scenario and the standard mean average precision (mAP) metrics are reported for a synthetically generated flight test dataset.

I. Introduction

Package delivery using small unmanned aerial systems (UAS) is experiencing accelerated progress and is closer to becoming reality. The Federal Aviation Administration (FAA) recently granted approval for companies such as Zipline and Wing Aviation to operate commercial UAS in the Dallas-Fort Worth (DFW) area without requiring visual observers, thus enabling beyond-visual line of sight (BVLOS) operations [1–3]. This historic authorization marked a significant shift in the regulatory landscape, aiming to incorporate routine UAS deliveries safely into the national airspace. It allows leading UAS delivery companies to deliver packages while keeping their UAS safely separated using Unmanned Aircraft System Traffic Management (UTM) technology [2]. It also represents a critical step toward routine BVLOS operations in shared airspace, supported by technological advancements in air traffic management.

Wing’s approach focuses on the direct delivery of lightweight packages from businesses to consumers. The company, a subsidiary of Alphabet, has developed a fleet of autonomous drones designed to transport small items quickly and efficiently, catering to on-demand needs [4]. As of October 2024, the company serves 26 cities and towns around the DFW metroplex, with an average flight time from stores to customers of 3 minutes and 24 seconds [5]. In contrast, Zipline is renowned for its pioneering work in medical supply delivery, particularly in African nations such as Rwanda and Ghana. The company operates one of the world’s largest autonomous UAS delivery networks, with service areas

^{*}Research Intern, George Washington University.

[†]Ph.D. Student, Department of Mechanical and Aerospace Engineering, George Washington University, AIAA Student Member.

[‡]Ph.D. Student, Department of Mechanical and Aerospace Engineering, George Washington University, AIAA Student Member.

[§]Ph.D. Student, Department of Computer Science, George Washington University, AIAA Student Member.

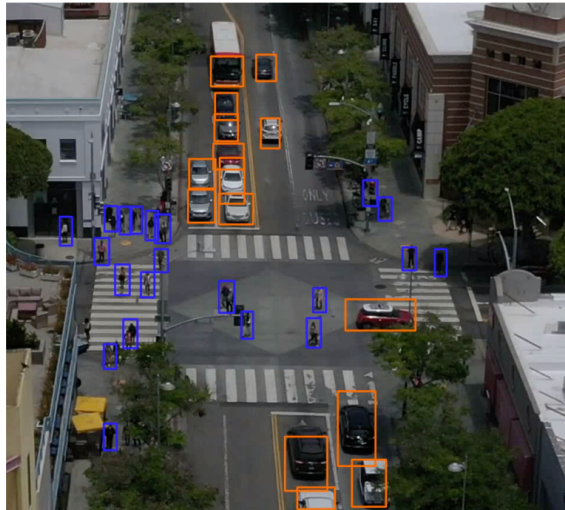
[¶]Associate Professor, Department of Mechanical and Aerospace Engineering, George Washington University, AIAA Associate Fellow.

^{||}Ph.D. Student, Department of Biomedical Engineering, George Washington University, AIAA Student Member.

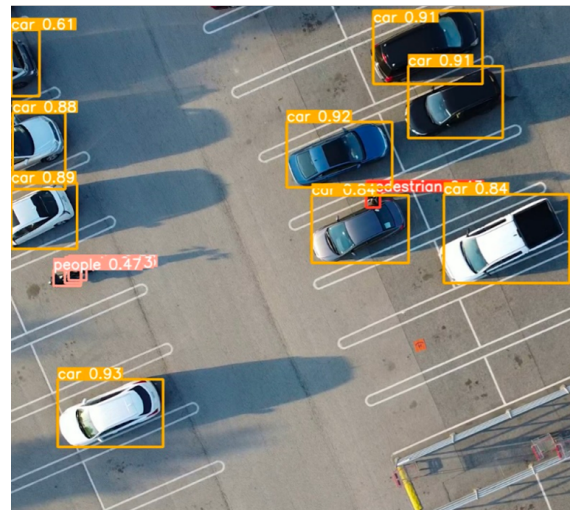
¹These authors contributed equally to this work.

encompassing the United States, Japan and other countries. Zipline’s delivery system is versatile, serving not only healthcare but also retail and logistics sectors by integrating with existing e-commerce and parcel carrier services [3].

As BVLOS UAS delivery operations become increasingly prevalent, ensuring the safety and precision of UAS landings is important. Accurate vision-based landing systems play a crucial role in enabling reliable autonomous deliveries, especially in complex environments where GPS may be unreliable. To achieve a higher level of autonomy during an off-nominal landing, a UAS needs to recognize the landing scene configuration with moving or static obstacles, such as people, pets, cars, or bicycles, in close proximity of the designated landing pad.



(a) An urban environment aerial scene with pedestrian and car detection made by a RetinaNet based computer vision model [6].



(b) A retail parking lot landing scene with pedestrian and car detection made by a YOLOv5 based computer vision model [7].

Fig. 1 Two different aerial scenes from prior work with obstacle detections within expected operational environments of small UAS delivery service.

II. Related Works

In the field of vision-based autonomous landing for UAS, prior research from our team has primarily focused on developing deep learning based computer vision models for the landing pad and objects near the landing pad. Yang et al. [6] developed a deep learning based computer vision model using RetinaNet for pedestrian detection from small UAS during dynamic aerial missions. Their work demonstrated high accuracy in detecting pedestrians and vehicles in the environment despite the relatively small size of the objects from a bird’s eye view. Zhao et al. [7] extended this work by comparing the RetinaNet and You Only Look Once (YOLOv5) architectures for real-time landing pad and obstacle detection, as well as implementing a safety checking algorithm using the obstacle’s bounding box and landing pad coordinates. It was shown that YOLOv5 and RetinaNet yielded similar accuracy; however, YOLOv5 was selected for its higher inference speed as the object detector for this work. Furthermore, ArUco markers were introduced for more precise landing pad identification. The YOLOv5 model was pre-trained on the COCO [8] dataset and fine-tuned on the VisDrone [9] dataset. The YOLOv5 model was trained on a workstation running Ubuntu 20.04 with an Intel Xeon W-3245 @ 4.60 GHz (32 cores), 128GB RAM, and an RTX 3080 GPU (10GB) and evaluated on an Nvidia Jetson Xavier NX. It was shown that the YOLOv5 model could successfully detect the landing pad and pedestrians through real-world data testing, and the integrated safety checking algorithm successfully checked for the safety of the landing zone by ensuring that objects crossing into the landing pad area will signal danger to the UAS.

Some studies focused on autonomous landing systems for UAS on moving platforms. One study employed gimbal-controlled monocular vision systems combined with inertial measurement units to stabilize the camera’s view of a moving platform, enhancing the landing accuracy through monocular ranging techniques [10]. Another approach integrated ArUco fiducial markers for pose estimation, allowing UAS to autonomously detect, follow, and land on mobile ground platforms. The UAS leveraged a vision-based trajectory planning system implemented on a single board

computer [11]. Additionally, a coarse-to-fine visual landing system utilizes specially designed QR code markers and the YOLO framework to improve positional accuracy, incorporating Kalman filtering to fuse data from visual and inertial sensors [12].

Real-world data collection can be a costly and difficult task with many challenges including safety concerns, dataset coverage, environmental variability, and potential legal/regulatory constraints. With the popularity of deep learning-based computer vision approaches to applications within the UAS domain, the ability to generate large amounts of synthetic data is critical in supporting the development of safe, fully-autonomous UAS. Vietz et al. [13] reported targeted synthetic data generation as a path to closing performance gaps with Convolutional Neural Network (CNN) based vision systems in the bicycle detection use-case. By building targeted synthetic training data, it was shown that the detection performance could be improved for identified cognition gaps of the CNN vision model; however, the use of synthetic data comes at a slight cost to the mean detection probability. The authors of [13] reported that the cost to the detection probability was due to their generated data not being realistic enough to improve overall detection of bicycles as well as bias in the validation dataset. Lee et al. [14] reported a method for synthetically producing photo-realistic image datasets from a simulation of 3D-scanned physical spaces, which are linked in a digital twin configuration, to facilitate safe navigation of UAS. These investigations showed that there was promise in utilizing synthetically generated datasets for deep learning as an alternative to more costly real-world data collection methods. It was reported that the simulated datasets did not include weather configurations such as fog and snow due to poor simulation performance and lack of realistic image samples for comparison. Verma et al. [15] proposed the use of game engines as a platform for realistic scenario simulations for the aerospace domain. Their paper described four critical requirements considerations when using game engines as simulation platforms for aerospace autonomous flight use cases and concluded that modern game engines contain the necessary foundational capabilities for offering high-fidelity, hyper-realistic scene rendering of diverse landing scene configurations where real-world data collection would be difficult and/or cumbersome; however, the authors identified the lack of a ready-made solution capable of producing simulations. These are areas this paper intends to address. Our work using AirSim in the autonomous UAS landing domain provides a simulation platform for easily generating realistic synthetic UAS landing scenarios with tunable environmental condition parameters, such as rain and fog, for evaluation of computer vision model performance.

III. Microsoft AirSim

Microsoft AirSim [16] (Aerial Informatics and Robotics Simulation) is an open-sourced robotics simulation platform developed by Microsoft Research 2017 and is inspired by the development of reinforcement learning algorithms for autonomous agents that can operate in the real world. AirSim aims to solve two problems: the need for large quantities of data sets for training and the ability to debug in a simulator. It utilizes current game engine rendering, physics and perception computation technologies to create realistic, real-world simulations.

The simulator architecture is designed to emphasize modularity and extensibility. The core components include environment models, physics engine, sensor models, public API layers, and an interface layer for vehicle firmware designed to be independently used with minimal dependencies outside of AirSim and easily extensible. These components are connected as shown in Figure 2. To produce highly-detailed simulation data for the companion computer to process, AirSim leverages the rendering technologies of platforms such as the Unreal Engine [17]. The AirSim codebase is implemented as a plugin for the Unreal Engine that can be dropped into any Unreal project. Unreal Engine offers capabilities to generate reasonably faithful reconstructions of real-world scenes. Due to the powerful simulation capabilities offered by AirSim and Unreal Engine, our work aims to utilize this simulator platform to generate realistic UAS landing scenarios that can be used for both training and evaluating our computer vision model toward the goal of safe, autonomous UAS landing.

To implement AirSim in our simulation workflow, we integrate the AirSim plugin into a custom Unreal Engine 4.27 environment. Visual Studio 2022 serves as the primary development environment, configured with the latest .NET Framework SDK and Windows SDK. The AirSim repository is cloned and built using its provided scripts, after which the Plugins directory is inserted into the Unreal project structure.

Integration requires modifying the project's .uproject configuration to enable the AirSim plugin and include necessary dependencies. To ensure compatibility and asset recognition, project configuration files such as DefaultGame.ini are also updated. Once the Unreal project is built and launched in the DebugGame Editor configuration, the environment is configured to use the AirSimGameMode to initialize the drone simulation. These steps enable the creation of custom 3D scenes suitable for simulating UAS landing operations under varied and dynamic conditions. By leveraging AirSim's extensibility and Unreal Engine's photorealistic rendering, we simulate realistic aerial perspectives, obstacle interactions,

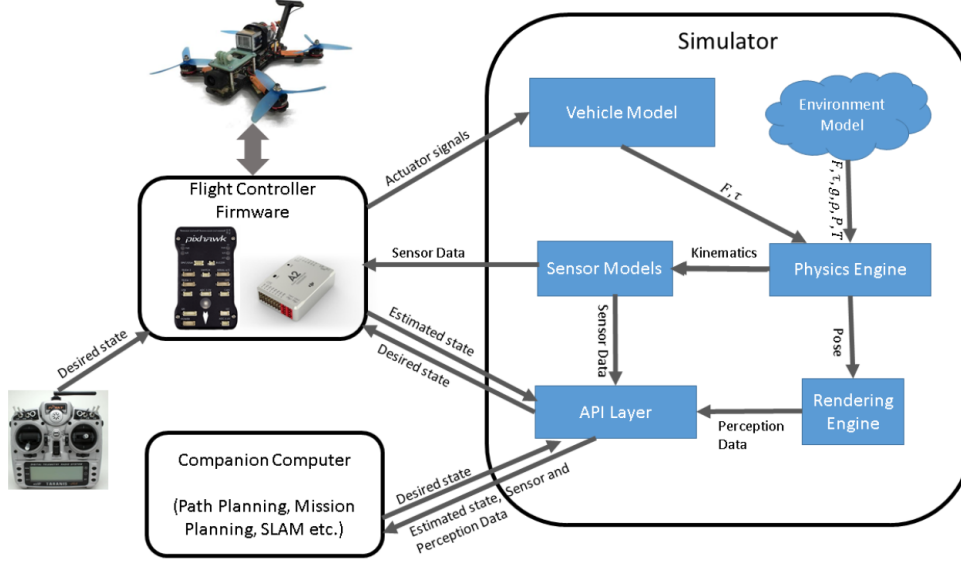


Fig. 2 Architecture and data flow of the AirSim simulation framework [16]. The simulator integrates key components such as the vehicle model, physics engine, sensor models, rendering engine, and environment model to emulate realistic UAS behavior.

and landing pad scenarios. This setup allows the generation of diverse datasets and enables rigorous, repeatable testing of object detection and decision-making algorithms in controlled, high-fidelity virtual environments.

IV. Landing Scenario Specification and Simulation Using AirSim

A. Scenario Specification

The development of custom environments, including UAS parameters, is facilitated by Unreal Engine’s built-in tools and its marketplace in the Epic Games Launcher. The Unreal Engine marketplace in the Epic Games Launcher offers a diverse range of pre-built environments that can be installed into a project. The installed pre-built environments will automatically be set up in the content browser and environment. The marketplace also provides asset packs containing various objects that can be imported into custom environments. These assets will appear in the Content Browser and are dragged into the environment.

AirSim offers two approaches for vehicle control: manual and programmatic. In manual control, users operate the UAS through a remote controller (RC), simulating real-time vehicle control. Alternatively, programmatic control utilizes APIs to interact with the Unreal Environment. When AirSim starts, it opens a communication port that connects to the code written in programming languages such as Python, C++, or Java. This enables remote procedure calls (RPCs) to control elements within the simulated environment. To implement programmatic control, users access the *PythonClient* folder in VS, where the user codes the desired actions or behaviors of the UAS. The actions (or behaviors) performed by the UAS, such as takeoff, landing, and navigation to specified three-dimensional Cartesian coordinates at specific speeds, can be written in code such as python, C++, or JAVA. To execute these coded actions, AirSim must be actively running within Unreal Engine. Once initialized, the user can run the code in the VS Code IDE, triggering the programmatic commands to interact with the Unreal environment.

The UAS settings are located in the **settings.json** file and can be modified to the user’s needs. The settings are edited through VS Code and allows the modification of parameters such as camera positioning, camera types, and capture settings. The camera types available for data collection include depth perspective, depth planar, infrared, etc. These settings change the views that are displayed in the three sub-windows and during recording.

Atmospheric conditions of the environment are added through the weather API that is implemented into any AirSim environment, enabling the simulation of diverse weather effects such as various intensities of rain, fog, snow, and falling leaves. This API enhances environmental diversity and provides the capability for more comprehensive data collection.

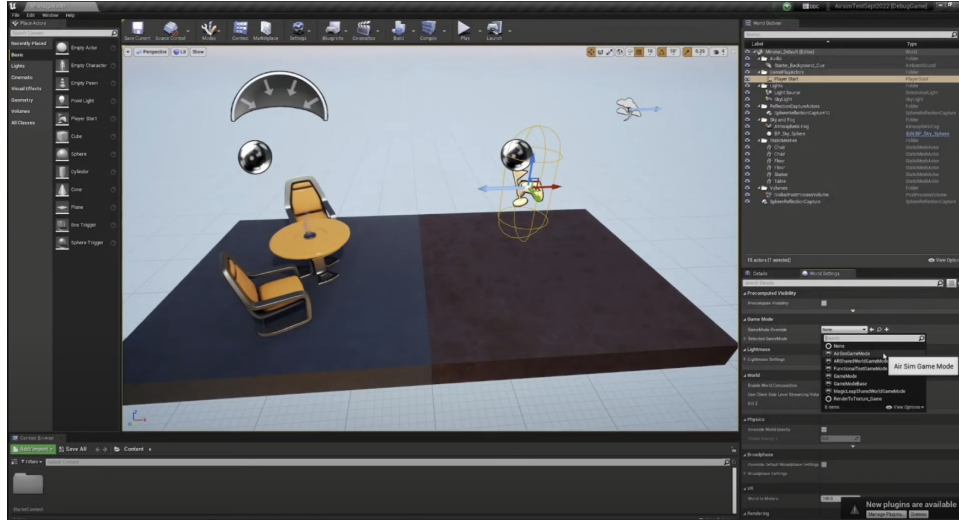


Fig. 3 Default Unreal Engine environment with GameMode Override to AirSim and PlayerStart Icon

Weather adjustments can be executed programmatically via VS Code or interactively within the Unreal environment. Within the simulation environment, a control panel can be opened containing sliders that allow users to manually adjust weather conditions in real time.

To simulate different times of day within an Unreal Engine environment, the **DirectionalLight** setting can be adjusted. This object setting is a simulation of the sun's position within the environment. By rotating the **DirectionalLight** object, the "sun" can be positioned to represent various times, and the Refresh Material button in the **SkySphereBlueprint** should be selected to update the sky's appearance. For nighttime simulation, the **DirectionalLight** should be positioned beneath the environment or set to non-visible. Additionally, setting the **Zenith Color**, **Horizon Color**, and **Cloud Color** in the **SkySphereBlueprint** to black will create a darkened atmosphere. If residual light persists, a **plane object** can be placed beneath the map to block any unwanted lighting from below. **PointLights** can then be added as needed to achieve the desired ambient lighting.

ArUco markers can also be simulated within the environment through this method. By downloading the file for the ArUco markers, they can be easily imported into the Unreal Engine workspace. These markers can then be dragged over a basic **plane** object in the environment, which can be found under the **Place Actors** bar on the left side. Unreal Engine's tools offer users the capability to manipulate objects, like the ArUco markers, through rotations, scaling, and translation within the environment. Users can also diversify the rendered environments by specifying parameters such as foliage density, landscape shapes, and sunlight to vary the time of day.

The integrated recording API is a highly accessible tool within the Unreal Engine environment that enables detailed data capture. Activation of the recording feature in the interface initiates frame-by-frame capture from the UAS's camera, with each frame saved as a .PNG image in a designated folder. The resolution for these captures is adjustable through the settings.json file to suit the use case needs. Alternatively, external screen recording software, such as OBS Studio, may be used to capture the Unreal environment directly. Camera movement can be controlled by entering manual mode and navigating with the WASD and arrow keys. The camera movement can be slowed for more precise camera movement and control. The combination of manual camera control with external recording software, as well as utilization of AirSim APIs, allow for flexible, high-quality recording of simulated aerial views. When creating an environment, objects are categorized into two primary types: static meshes and skeletal (or riggable) meshes. Static meshes are stationary objects within the simulation, such as people sitting on benches. Skeletal or riggable meshes, on the other hand, represent objects that can move within the environment. These meshes require "rigging", which can be done in a 3D graphics software such as Blender. The example scenes in Figure 4, although static images, showcase both static and skeletal meshes. While static meshes can be dragged in and adjusted, movable meshes are much more complex. In the example scenes, non-playable characters (NPCs) are created to simulate a more realistic scene. These moving objects are created by overlaying imported skeletal meshes from the Unreal Marketplace (or third-party sources) onto cars and Third-Person character models provided within Unreal Engine. A spline pathway is then established to define a specific route and speed for the car or character to follow.

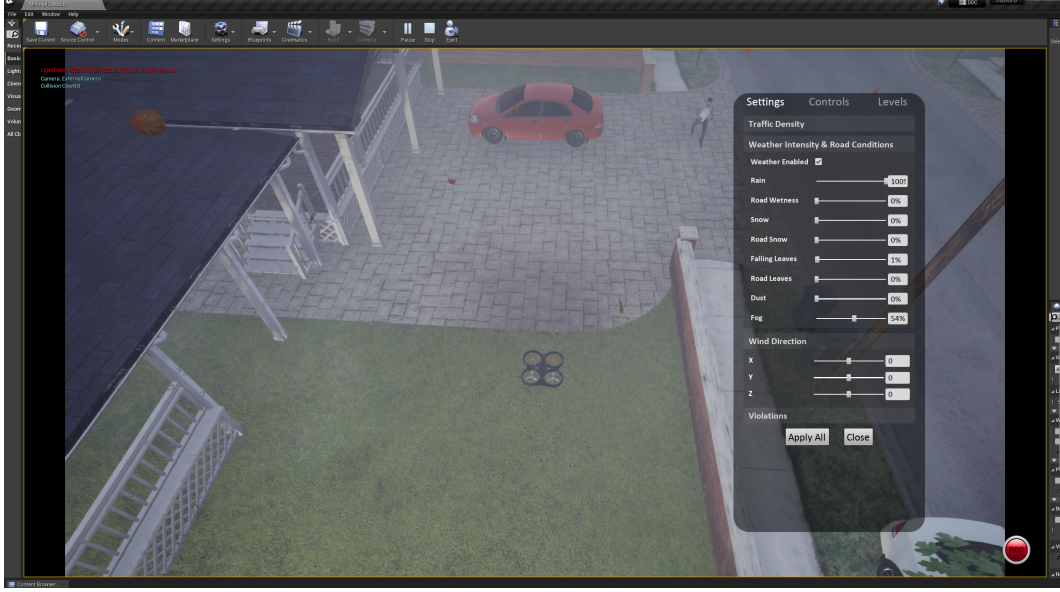


Fig. 4 A rendered environment of a front yard landing zone with weather API implementation. The environmental parameters are adjusted through the sliders shown on the right side of the image. In this scenario, the fog is increased to reduce the visibility of the scene. Other weather conditions such as snow, dust, and rain can be changed as needed to produce the expected atmospheric conditions of the operational environment of the UAS. Car and pedestrian objects are also added to the environment for more realistic rendered scenarios.

These are a few of the features available within AirSim that we leverage in our research to generate realistic synthetic flight data. There are many parameters available to vary, allowing the generation of diverse UAS landing environments for testing and/or training computer vision object detection models.

V. Synthetic Data for Object Detection Model Evaluation

A. YOLOv8 for Aerial-View Object Detection

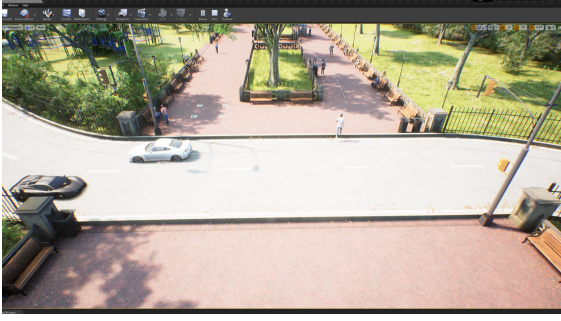
The object detection architecture selected for this work is the Ultralytics YOLOv8 model [18]. Released in 2023, YOLOv8 offers substantial improvements in both accuracy and computational efficiency over YOLOv5, and has become a popular architecture for real-time deployment in embedded or resource-constrained environments such as onboard unmanned aerial systems (UAS). The model architecture retains the backbone, neck, and head structure of its previous iterations.

The backbone is based on an enhanced CSPDarknet convolutional network [19], which extracts hierarchical features from input images. These features are then passed to the neck, which employs a Path Aggregation Network (PANet) [20] to fuse multi-scale feature maps, improving performance on small object detection tasks—such as landing pad recognition in aerial views. Finally, the anchor-free detection head predicts object bounding boxes and class confidences directly, eliminating the need for predefined anchor boxes and simplifying the training process.

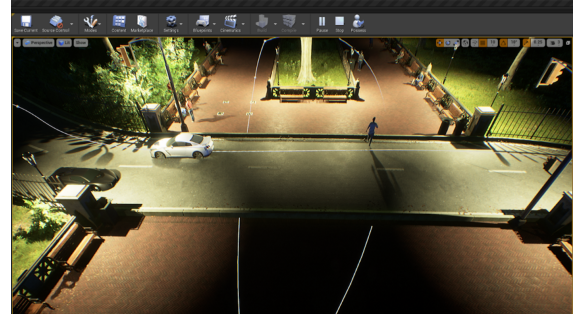
The model is trained using a composite loss function composed of box regression loss, classification loss, and Distribution Focal Loss (DFL), which encourages precise localization of bounding box edges. These components are combined through weighted summation as shown in Equation 1. This structure allows YOLOv8 to efficiently learn object boundaries and class predictions with a high degree of precision, critical for the safety-critical task of autonomous UAS landing site identification.

$$L_{\text{total}} = \lambda_{\text{box}} \cdot L_{\text{box}} + \lambda_{\text{cls}} \cdot L_{\text{cls}} + \lambda_{\text{dff}} \cdot L_{\text{dff}} \quad (1)$$

where L_{box} is the bounding box regression loss, L_{cls} is the binary cross-entropy classification loss, and L_{dff} is the distribution focal loss (dff). The coefficients λ_{box} , λ_{cls} , and λ_{dff} control the weights given to each loss component. These



(a) A simulated park environment during a particularly sunny day. The scene includes multiple car and pedestrian objects, as well as trashcans, benches, and street-lights, for a realistic scene of a busy park during the day.



(b) The same simulated park environment during a nighttime scene. Again, the scene includes multiple user-defined objects as in Figure 6a to create a realistic park scenario. The time of day is adjusted for night time, and the point light sources from the street-lamps add to the realism of the night scene.

Fig. 5 Two simulated environments showing pedestrian and car objects under two different lighting conditions. This shows AirSim’s capabilities to produce realistic, adjustable environments to reflect the expected conditions of the UAS delivery use case. Additionally, ArUco marker objects have been placed just north of the white cars in both scenes to simulate a potential marked landing site.

values are tuned to balance localization accuracy, classification performance, and boundary precision. The YOLOv8s model is a smaller version of the YOLOv8 model, but retains the same overall structure and loss function.

The YOLOv8s model was trained using the Ultralytics framework on the VisDrone dataset. VisDrone is a large-scale benchmark dataset designed for object detection in aerial imagery, containing thousands of high-resolution images and video sequences captured by UAVs across multiple cities in China. The dataset includes diverse urban and suburban environments with varying lighting, weather conditions, and object densities, making it well-suited for training a robust aerial detection model. To assess the viability of using synthetically generated UAS landing flight data for object detection model evaluation, we test this YOLOv8s model trained on the VisDrone dataset with rendered data mirroring real-world scenarios.

Training was conducted for 100 epochs with a batch size of 8 and input resolution of 1280×1280 pixels. The model was initialized with COCO-pretrained weights to accelerate convergence and improve generalization to the aerial-view object detection domain. Optimization was performed using the Adam optimizer with an initial learning rate of 0.01 and cosine learning rate scheduler. The learning rate factor was 0.0001. The loss weights were set to $\lambda_{\text{box}} = 7.5$, $\lambda_{\text{cls}} = 0.5$, and $\lambda_{\text{dfl}} = 1.5$. Data augmentation strategies were applied during training, including mosaic augmentation, horizontal flipping, and RandAugment-based transformations such as color jitter, mixup ($\alpha = 0.2$), and random erasing (with 0.4 probability). A three epoch warm-up training was used, and early stopping 15 epochs to prevent overfitting.

YOLOv8 model training was performed using a workstation with an AMD Ryzen Threadripper PRO 3975WX (32-cores) CPU at 2.2 GHz and two GeForce RTX 3090 GPUs. This workstation is also used for model evaluation using the AirSim generated data.

B. Generating UAS Landing Environments in AirSim

The object and environmental parameters discussed in section IV.A are used to develop and generate the scenes shown in Figures 4 and 5. Figure 4 shows a front lawn landing scene with a foggy weather condition. The scene contains a car object and a pedestrian object that are oriented in a manner that reflects a real-world scenario, i.e. the car is in the driveway and the pedestrian is on the sidewalk. The environmental weather conditions, such as rain, snow, fog, and wind speed, are controllable via the sliders in the control panel on the right side of the window. The sliders allow for the user to finely specify the effects taking place in the environment, allowing for the precise specification of diverse landing environment data. A UAS object is also placed on the front lawn of the simulated environment, with its movements controllable either programmatically via API commands or manually through user input.

Figure 5 presents two rendered potential landing environments in a suburban neighborhood park: a day-time (bright lighting) scene and a night-time (low lighting) scene created by the directional lighting and point light features of the

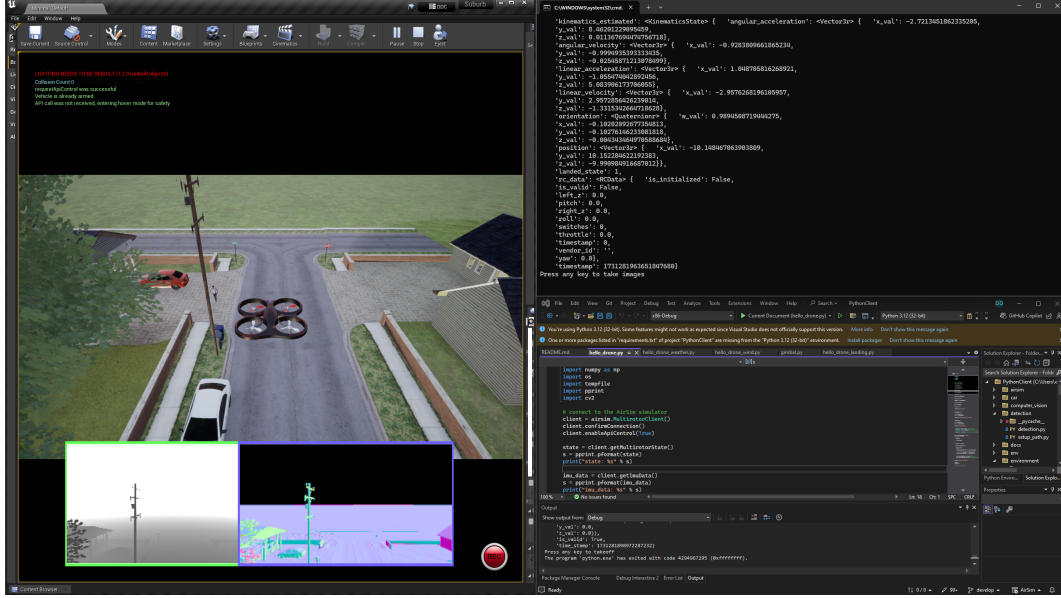


Fig. 6 Execution of the UAS flight code. The forward-facing camera attached to the UAS displays real-time views of the environment through the two sub-window views at the bottom-left of the image.

simulator. Unreal Engine leverages these lighting capabilities, including high-fidelity shadow rendering, to simulate realistic lighting conditions that closely resemble those found in real-world environments. In both scenes, multiple cars and pedestrians objects are placed around the environment on the roads and sidewalks. The objects are placed such that the environment reflects the real-world, i.e. cars are on the road and, pedestrians crossing the road or inside the park pathway. For realistic and dynamic environments, the pedestrians and cars can be set with pre-defined movements over time that mimic real-world behaviors, rather than strictly static objects. Additionally, the ArUco markers can be imported into the scene as an object that can be used to define the landing site of a potential UAS package delivery operation. These examples demonstrate AirSim’s capability to simulate realistic time-of-day scenes by changing the position of the "sun" within the environment relative to the scene of interest including common objects that can be present in the real-world scenario.

The synthetically generated UAS landing flight test data was generated on a personal computer running an Intel(R) Core(TM) i5-10400F CPU at 2.90GHz and 16 GB RAM.

VI. Results and Discussion

This section presents various results of our experiments using synthetic data generation for the evaluation of the YOLOv8 object detection model. Figures 7 and 8 show sample images from our YOLOv8 model inference on a set of four synthetically generated UAS-view landing operation. Figures 7a, 7c, 7e, and 7g* show example frames from an AirSim-generated suburban front yard UAS landing scene. In this scenario, a car is parked on the side of the road and occluded by a tree, while a pedestrian is walking down the sidewalk from top to bottom. The weather conditions are clear with a directional light source placed to the left of the scene. This can be seen by the object shadows extending to the right side of the images. It is shown in the figures that the YOLOv8 model can successfully detected the pedestrian and car objects, while also being able to use ArUco marker detection algorithms to identify the landing site; however, it can be seen in Figure 7c that the pedestrian is not always detected. Furthermore, a tire gets incorrectly detected as a pedestrian in Figure 7a.

Figures 7b, 7d, 7f, and 7h[†] present a UAS landing scenario set in a public suburban park setting. Point light sources placed at streetlamps create a realistic night-time landing scenario that may be difficult to collect in the real-world due to regulatory constraints or cost. In this scene, there are far more objects present than the front yard example. Multiple cars are moving horizontally on the road (from the point-of-view of the UAS), and pedestrians standing still or

*Full Video: <https://www.youtube.com/watch?v=qnbzv3SVIZo>

†Full Video: <https://youtu.be/7dEe5oXEPFE>

moving around the landing site in the park. The UAS starts at an initial altitude significantly higher than the previous example. The YOLOv8 detection model reliably identifies large objects such as cars, even at high altitudes, due to their prominent size. However, it struggles to consistently detect pedestrians in some frames. This limitation is due to the small apparent size of pedestrians at higher altitudes, where distinguishing visual features become difficult to resolve. As the UAS camera get closer to the ground, the detector is able to detect the pedestrians. Additionally, the pedestrian assets imported into Unreal Engine may lack the visual detail and resolution of real-world counterparts, further reducing detection accuracy. This is further made difficult by the lack of clear lighting, with many pedestrians being hidden by the darkness outside of the light. Although the lighting is not ideal in this scenario, the YOLOv8 model is still able to make predictions on the objects and is accurate for the cars, even when there is lack of sufficient lighting.

Additional examples of inference performed on variations of these scenes is given in Figure 8. Figures 8a, 8c, 8e, and 8g [‡] is of the same suburban front yard as in Figure 7, but a hazy/foggy weather condition has been added. In this scenario, our YOLOv8 model is still able to detect the objects, and we can further augment the scene with additional adverse conditions using the tools available in AirSim. Figures 8b, 8d, 8f, and 8h [§] is a day-time variation of the suburban park scene from Figure 7. In these figures, daylight conditions are simulated by positioning a directional light source to the east of the park, relative to the UAS’s point of view. Compared to the lower-intensity lighting used in the front yard environment, this scene is significantly brighter. The increased illumination enhances scene visibility overall, but it can also reduce detection performance by causing color saturation, reducing the contrast between brightly colored objects and the surrounding environment. Such conditions can be incorporated into synthetic data generation to augment the training dataset and enhance the model’s performance in brightly lit or visually saturated environments. Even with the higher-intensity lighting, our YOLOv8 model can still make accurate predictions on the cars and some pedestrians as it descends to the ground.

Figure 9a and Figure 9 summarize the evaluation results of the YOLOv8 model on a labeled AirSim-generated test dataset, specifically the video corresponding to the images in Figures 7a, 7c, 7e, and 7g. The confusion matrix (Figure 9a) indicates that the model performs well in detecting the “car” class, with a high true positive rate and limited misclassifications. However, performance on the pedestrian object class is substantially lower, where many of the ground truth labels for pedestrian were not detected. The precision-recall curve shown in Figure 9 further highlights this disparity. The car class achieves strong performance with a precision of 0.936, whereas pedestrian detection is considerably weaker, with a precision of 0.197. The overall model mAP@0.50 and mAP@0.50-0.95 across all classes are 0.567 and 0.390, respectively, indicating moderate detection performance on the dataset. This can guide future training efforts by emphasizing augmentation or dataset enhancement strategies for object classes that exhibit lower detection accuracy, especially in improving the overall robustness of the model in off-nominal operating conditions.

These results indicate that synthetic data generation via simulators such as AirSim holds significant promise for uncovering weaknesses in object detection models that may not be exposed through real-world testing alone. Real-world flight testing is often constrained by safety, cost of collection, and environmental variability, making it difficult to consistently capture rare or adverse landing scenarios. By synthetically generating data, we gain the ability to create large volumes of diverse landing conditions, including extreme lighting, occlusions, or complex obstacle layouts, at significantly cost and effort. This controlled and repeatable testing environment enables targeted testing of perception models and can accelerate the iterative improvement of UAS safety systems prior to real-world deployment.

VII. Limitations

While Microsoft AirSim offers a powerful simulation platform for generating synthetic data, some limitations arise when using it to model UAS landing scenarios. The sim-to-real gap remains a significant concern. Despite leveraging Unreal Engine’s rendering technologies, the textures, lighting dynamics, and sensor noise in simulated scenes may not fully capture the complexity and variability of real-world environments, which can limit the generalizability to real-world situations.

Additionally, realism in dynamic scenes is constrained in simulated environments. While AirSim supports scripted and interactive objects, the behavior of obstacles, such as pedestrians and vehicles, is typically predefined or lacks the unpredictability seen in real-world operations. This may lead to detection models that are less robust when faced with complex or emergent behavior during actual landings scenarios.

Furthermore, the diversity of environmental context and realism achievable in simulation is constrained by the available assets within the Unreal Engine ecosystem. Generating richly varied landing scenarios—featuring realistic

[‡]Full Video: <https://youtu.be/OQzCPpxNAbc>

[§]Full Video: <https://youtu.be/IyAx5xmeJqE>

terrain, vegetation, clutter, and urban structures—often requires significant manual effort and domain expertise. This limitation becomes relevant when aiming to generalize performance across different geographic regions, where the visual characteristics of common objects may vary. For instance, while vehicles in the United States and Europe serve the same function, subtle differences in their design, size, or lighting features can impact object detection performance and, by extension, system safety and reliability.

Finally, Microsoft AirSim has been deprecated and is no longer under active development or maintenance. This can pose challenges for long-term sustainability, integration with future versions of Unreal Engine, and access to community support, which may limit its continued viability for simulation-based research and development. Access to the original code is still available, though no updates are being made.

VIII. Conclusion

In this paper, we present our efforts at using simulator-based approaches for synthetic data generation of autonomous UAS package delivery landing scenario test cases. It was shown that generating flight data via AirSim yielded realistic environments where the properties of the scene, such as weather, lighting, and object placement, can be systematically controlled. This level of control enables the creation of diverse and challenging test cases that are often difficult, costly, or unsafe to capture in the real world.

By leveraging AirSim, we were able to evaluate the performance of a YOLOv8-based object detection model in detecting obstacles around a landing pad under various simulated conditions. The results demonstrated that synthetic data can reveal weaknesses in model performance, particularly under adverse or rare environmental scenarios. These insights can inform future training strategies by guiding data augmentation and targeted data collection.

Future work in this area can explore the use of synthetic data generation to augment training datasets with rare or safety-critical edge cases that are difficult to capture in real-world settings. This can improve the robustness and generalization of object detection models by exposing them to a broader variety of conditions and scenarios, ultimately enhancing the reliability of autonomous UAS operations. Another area that future efforts can investigate is in the automation of scenario generation at scale with simulator tools. Automating the placement of obstacles, variation of environmental conditions, and trajectory design can streamline the creation of diverse, rich training and evaluation datasets. This could reduce manual effort, enhance reproducibility, and enable systematic exploration of safety-critical scenarios for autonomous UAS landing operations.

Acknowledgments

We thank our Federal Aviation Administration (FAA) colleagues for their continued funding and support for our research.



(a)



(b)



(c)



(d)



(e)



(f)



(g)



(h)

Fig. 7 Example frames from a YOLOv8 model inference on synthetically generated UAS landing test videos. The left column contains frames from a front yard landing scenario under day-light conditions. The right column contains frames from a park landing scenario under night-time conditions. The IoU and confidence thresholds of the detector were set at 0.5 and 0.25, respectively.



(a)



(b)



(c)



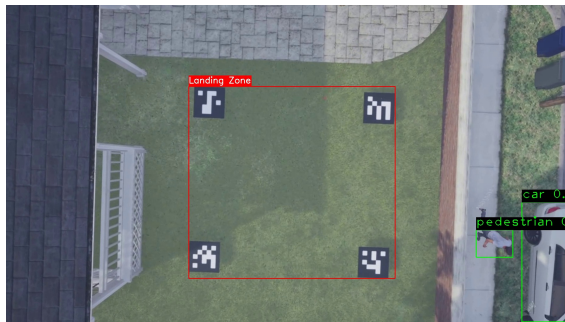
(d)



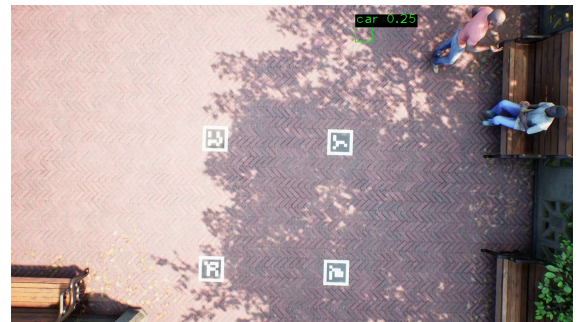
(e)



(f)

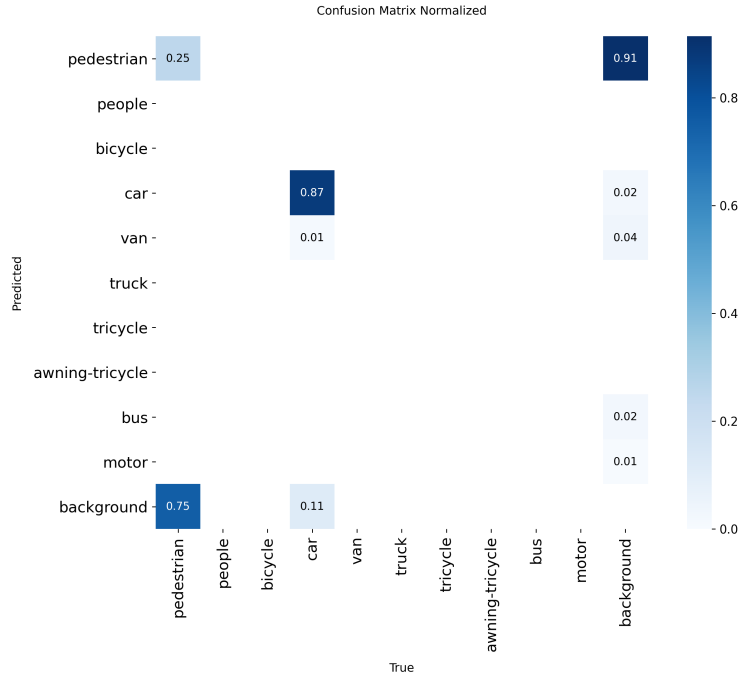


(g)

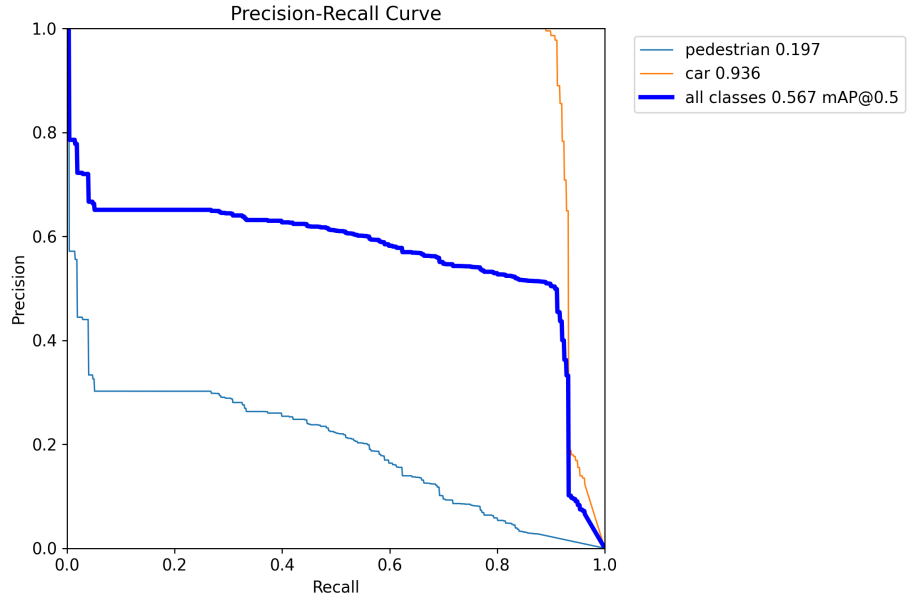


(h)

Fig. 8 Additional example frames from a YOLOv8 model inference on synthetically generated UAS landing test videos. The left column contains frames from the same front yard landing scenario under day-light conditions as 7a, with added with foggy weather. The right column contains frames from a park landing scenario under very bright day-time conditions. Again, the IoU and confidence thresholds of the detector were set at 0.5 and 0.25, respectively.



(a) Confusion matrix from the evaluation of the YOLOv8 model on a labeled synthetically generated test dataset. The dataset was labeled via Roboflow [21], and evaluation was performed using the Ultralytics API.



(b) Precision-Recall curve from the same evaluation of the generated dataset using the Ultralytics API. The precision-recall curve of the car class shows significantly superior performance over the pedestrian class.

Fig. 9 Evaluation results of the YOLOv8 model on a synthetically generated test dataset using AirSim. The test dataset was generated by extracting frames from a simulated UAS landing onto a suburban front yard, as shown in Figures 7a, 7c, 7e, and 7g.

References

- [1] Federal Aviation Administration, “Package Delivery by Drone (Part 135),” , 2023. URL https://www.faa.gov/uas/advanced_operations/package_delivery_drone, accessed: 2024-10-19.
- [2] Federal Aviation Administration, “FAA Makes Drone History in Dallas Area,” , 2024. URL <https://www.faa.gov/newsroom/faa-makes-drone-history-dallas-area>, accessed: 2024-10-19.
- [3] Marquand, Y. L., “FAA authorises Zipline and Wing for BVLOS operations in Dallas,” , 2023. URL <https://www.revolution.aero/news/2024/07/30/faa-authorises-zipline-and-wing-for-bvlos-operations-in-dallas/>, accessed: 2024-10-19.
- [4] Singh, I., “Zipline, Wing enter new drone delivery era with historic FAA waiver,” , 2024. URL <https://dronedj.com/2024/07/30/zipline-wing-drone-delivery-dallas/>, accessed: 2024-10-19.
- [5] Nate Milner, “1 year and 26 cities later: Celebrating our DFW growth,” , 2024. URL <https://blog.wing.com/2024/10/1-year-and-26-cities-later-celebrating.html>.
- [6] Yang, X., Murphy, M., Brittain, M. W., and Wei, P., “Computer Vision for Small UAS Onboard Pedestrian Detection,” *AIAA AVIATION 2020 FORUM*, American Institute of Aeronautics and Astronautics, VIRTUAL EVENT, 2020. <https://doi.org/10.2514/6.2020-3270>, URL <https://arc.aiaa.org/doi/10.2514/6.2020-3270>.
- [7] Zhao, Z., Lee, J., Li, Z., Park, C. H., and Wei, P., “Vision-based Perception with Safety Awareness for UAS Autonomous Landing,” *AIAA SCITECH 2023 Forum*, American Institute of Aeronautics and Astronautics, National Harbor, MD & Online, 2023. <https://doi.org/10.2514/6.2023-0126>, URL <https://arc.aiaa.org/doi/10.2514/6.2023-0126>.
- [8] Lin, T. e. a., “Microsoft COCO: Common Objects in Context,” *Computer Vision - ECCV 2014*, Springer, Cham, 2014. https://doi.org/https://doi.org/10.1007/978-3-319-10602-1_48, URL https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48#citeas.
- [9] Zhu et al., “Vision Meets Drones: A Challenge,” , 2018. URL <https://arxiv.org/abs/1804.07437>, accessed: 2024-10-24.
- [10] Zhao, Z., Han, P., Xu, Y., Xie, W., Zhang, W., Liang, K., and Zeng, Q., “Vision-based Autonomous Landing Control of a Multi-rotor Aerial Vehicle on a Moving Platform with Experimental Validations,” *IFAC-PapersOnLine*, Vol. 55, No. 3, 2022, pp. 1–6. <https://doi.org/https://doi.org/10.1016/j.ifacol.2022.05.001>, URL <https://www.sciencedirect.com/science/article/pii/S2405896322002658>, 16th IFAC Symposium on Large Scale Complex Systems: Theory and Applications LSS 2022.
- [11] Cui, Q., Liu, M., Huang, X., and Gao, M., “Coarse-to-fine visual autonomous unmanned aerial vehicle landing on a moving platform,” *Biomimetic Intelligence and Robotics*, Vol. 3, No. 1, 2023, p. 100088. <https://doi.org/https://doi.org/10.1016/j.birob.2023.100088>, URL <https://www.sciencedirect.com/science/article/pii/S2667379723000025>.
- [12] Morales, J., Castelo, I., Serra, R., Lima, P. U., and Basiri, M., “Vision-Based Autonomous Following of a Moving Platform and Landing for an Unmanned Aerial Vehicle,” *Sensors*, Vol. 23, No. 2, 2023. <https://doi.org/10.3390/s23020829>, URL <https://www.mdpi.com/1424-8220/23/2/829>.
- [13] Vietz, H., Rauch, T., and Weyrich, M., “Synthetic Training Data Generation for Convolutional Neural Networks in Vision Applications,” *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Institute of Electrical and Electronics Engineers, Stuttgart, Germany, 2022. <https://doi.org/10.1109/ETFA52439.2022.9921534>, URL https://ieeexplore.ieee.org/abstract/document/9921534?casa_token=FR7m1_YIZGYAAAAA:wru2U5JzroMDE2R-JhrUxF3QCnPtIJPkWUE0dovp4TkKmNnijPj39qIBUEkBxE9d_c0y_ka.
- [14] Lee, T., McKeever, S., and Courtney, J., “Synthetic Dataset Generation Using Photo-Realistic Simulation with Varied Time and Weather Axes,” *Electronics: Emerging Technologies in Digital Twins*, Vol. 13, No. 8, 2024, pp. 1–17.
- [15] Verma, A., Wendt, C., DeStafeno, N., Aggarwal, C., Baserinia, A., and Rodriguez, A. G., “Using Game Engines as Simulation Training Platforms to Enable Autonomous Flight,” *AIAA AVIATION FORUM and ASCEND 2024*, American Institute of Aeronautics and Astronautics, Las Vegas, Nevada, 2024. <https://doi.org/10.2514/6.2024-4125>, URL <https://arc.aiaa.org/doi/10.2514/6.2024-4125>.
- [16] Shah, S., Dey, D., Lovett, C., and Kapoor, A., “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” , 2017. <https://doi.org/https://doi.org/10.48550/arXiv.1705.05065>, URL <https://arxiv.org/abs/1705.05065>.
- [17] Epic Games, “Unreal Engine 5.5 Documentation,” , 2024. URL <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation>.

- [18] Jocher, G., Chaurasia, A., and Qiu, J., “Ultralytics YOLOv8,” , 2023. URL <https://github.com/ultralytics/ultralytics>.
- [19] Khalili, B., and Smyth, A. W., “SOD-YOLOv8—Enhancing YOLOv8 for Small Object Detection in Aerial Imagery and Traffic Scenes,” *Sensors*, Vol. 24, No. 19, 2024. <https://doi.org/10.3390/s24196209>, URL <https://www.mdpi.com/1424-8220/24/19/6209>.
- [20] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J., “Path Aggregation Network for Instance Segmentation,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768. <https://doi.org/10.1109/CVPR.2018.00913>.
- [21] Dwyer, B., Nelson, J., Hansen, T., et al., “Roboflow (version 1.0) [software],” [Online]. Available: <https://roboflow.com>, 2024. Computer vision.