# GENERATING MULTI-MODAL DRONE DETECTION DATA IN UNREAL ENGINE

by

Süleyman Emre Demir

B.S., Computer Engineering, Yıldız Technical University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2024

# ACKNOWLEDGEMENTS

I want to extend my heartfelt thanks to my thesis advisor, Hüseyin Birkan Yılmaz, and to all my professors for their constant support and invaluable contributions throughout these difficult times.

I am profoundly grateful to my research team and friends at NetLab in Boğaziçi University's Computer Engineering Department, with particular appreciation to Yusuf Özben for their significant input.

# ABSTRACT


## GENERATING MULTI-MODAL DRONE DETECTION DATA IN UNREAL ENGINE

In recent years, the rapid advancement of drone technology has resulted in significant challenges for security and surveillance systems. Effective drone detection methods are crucial for various applications, including national security, private property protection, and safe airspace management. This thesis addresses the need for comprehensive datasets required to train robust drone detection models by leveraging simulation environments. Our approach utilizes Unreal Engine and AirSim plugin in conjunction with the AirSim GUI, a simulation environment created to generate multi-modal datasets that include both visual and audio data of drones in diverse scenarios.

The primary objective of this research is to create a cost-effective and scalable method for generating high-quality datasets to improve the performance of drone detection models. We simulate a drone in a custom-built environment within Unreal Engine, capturing images and audio clips to form the dataset. The AirSim GUI allows for precise control over the simulation parameters, including the ability to spawn drones at specific coordinates, integrate various types of cameras, define flight paths, and record audio. We implemented an audio generation process that includes both real-time and offline capabilities, ensuring comprehensive coverage of possible detection scenarios. Our results demonstrate that the synthetic data generated through this method enhances the performance of drone detection models, particularly in situations where real-world data is scarce.

# ÖZET

## UNREAL ENGINE'DE ÇOK MODLU DRONE TESPİT VERİSİ ÜRETİMİ

Son yıllarda, drone teknolojisinin hızla gelişmesi, güvenlik ve gözetim sistemleri için önemli zorluklar ortaya koymuştur. Etkili drone tespit yöntemleri, ulusal güvenlik, özel mülk koruması ve güvenli hava sahası yönetimi gibi çeşitli uygulamalar için kritik öneme sahiptir. Bu tez, başarılı drone tespit modellerini eğitmek için gerekli olan kapsamlı veri setlerine olan ihtiyacı, simülasyon ortamlarını kullanarak ele almaktadır. Yaklaşımımız, çeşitli senaryolarda dronların hem görsel hem de ses verilerini içeren çok modlu veri setleri oluşturmak için geliştirdiğimiz AirSim GUI ile birlikte Unreal Engine ve AirSim eklentisini kullanmaktadır.

Bu araştırmanın temel amacı, drone tespit modellerinin performansını artırmak için yüksek kaliteli veri setleri oluşturmanın maliyet etkin ve ölçeklenebilir bir yöntemini geliştirmektir. Unreal Engine içinde oluşturulmuş bir ortamda bir dronunu simüle ediyor ve veri setini oluşturmak için görüntüler ve ses klipleri kaydediyoruz. AirSim GUI, simülasyon parametreleri üzerinde hassas kontrol sağlar ve belirli koordinatlarda drone oluşturma, çeşitli kamera türlerini entegre etme, uçuş yollarını tanımlama ve ses kaydetme gibi özellikler içermektedir. Çalışmamızda, tespit senaryolarının kapsamlı bir şekilde ele alınmasını sağlamak için hem gerçek zamanlı hem de çevrimdışı yetenekleri içeren bir ses üretim süreci uyguladık. Sonuçlarımız, bu yöntemle üretilen sentetik verilerin, özellikle gerçek dünya verilerinin yetersiz olduğu durumlarda, drone tespit modellerinin performansını artırdığını göstermektedir.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| $nMFCC$ | number of MFCCs |

# LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| 2D | Two Dimensional |
| 3D | Three Dimensional |
| AAA | Triple-A |
| AUC-ROC | Area under the receiver operating characteristic curve |
| AP | Average Precision |
| AR | Augmented Reality |
| API | Application Programming Interface |
| BN | Batch Normalization |
| CBL | Two |
| CNN | Convolutional Neural Network |
| COCO | Common Objects in Context |
| Conv | Convolution |
| CPU | Central Processing Unit |
| CSP | Cross Stage Partial |
| CRNN | Convolutional Recurrent Neural Network |
| DvB | Drone vs Bird |
| dB | Decibel |
| FNR | False Negative Rate |
| FP | False Positive |
| FPR | False Positive Rate |
| FFT | Fast Fourier Transform |
| GPU | Graphics Processing Unit |
| GPS | Global Positioning System |
| GUI | Graphical User Interface |
| HITL | Hardware-in-the-loop |
| IoU | Intersection Over Union |
| KHz | Kilo-hertz |
| LSTM | Long short-term memory |

| | |
|---|---|
| mAP | Mean Average Precision |
| MFCC | Mel-frequency cepstral coefficients |
| MPEG-4 | Moving Pictures Experts Group 4 |
| MS-COCO | Microsoft Common Objects in Context |
| ReLU | Rectified Linear Unit |
| RGB | Red Green Blue |
| RNN | Recurrent Neural Network |
| ROC | Receiver Operating Characteristic |
| R-Unit | Recurrent Unit |
| SITL | Software-in-the-loop |
| SPP | Spatial Pyramid Pooling |
| SVM | Support Vector Machine |
| TP | True Positive |
| TPR | True Positive Rate |
| UAV | Unmanned Aerial Vehicle |
| VGG | Visual Geometry Group |
| VOC | Visual Object Classes |
| VR | Virtual Reality |
| YOLO | You Only Look Once |

# 1. INTRODUCTION

The use of Unmanned Aerial Vehicles (UAVs) or drones has seen a significant rise in recent years, revolutionizing numerous industries and applications. Initially developed for military purposes, drones have now penetrated civilian and commercial sectors, offering solutions in fields such as agriculture, logistics, disaster management, and surveillance. Their ability to access remote and hazardous areas, coupled with advancements in technology, has made drones indispensable tools for enhancing efficiency, safety, and data collection capabilities.

Drones have been used for unauthorized surveillance, smuggling, and even acts of terrorism. Equipped with cameras and other sensors, drones enable covert operations, compromising privacy and enabling industrial espionage. Instances of using drones to smuggle into restricted areas, such as prisons and border zones, highlight serious security risks. Furthermore, incidents involving weaponized drones or the delivery of explosives pose significant challenges for counter-terrorism efforts. These threats highlight the urgent need for advanced detection technologies to prevent drone misuse and protect public safety.

Various methodologies have been developed for the detection of drones, each leveraging distinct technological principles to achieve robust identification. Visual detection methods utilize high-resolution cameras and advanced image processing algorithms to discern drone shapes, colors, and movement patterns against varying backgrounds, thereby facilitating effective identification under optimal lighting conditions. Acoustic detection approaches analyze the unique sound signatures emitted by drones, distinguishing them from ambient noise through sophisticated signal processing techniques. This method offers a reliable detection mechanism suitable for nighttime operations or environments with visual obstructions. Radar-based systems employ radio frequency waves to detect drones by capturing reflections from their surfaces, enabling precise determination of drone position, speed, and size. Integration of these diverse detec-

tion modalities into hybrid systems enhances overall detection accuracy and reliability, ensuring comprehensive surveillance capabilities essential for mitigating the risks associated with unauthorized drone activities across diverse operational settings. However, the integration of different modalities is not a straightforward task and requires advanced techniques and extensive testing.

Training data is essential for the development of effective detection techniques for drones, yet collecting a comprehensive dataset presents several challenges. Firstly, obtaining diverse and representative drone data across various environmental conditions, flight behaviors, and drone types can be logistically complex and time-consuming. Ensuring sufficient variability in drone characteristics, such as size, shape, and color, is crucial for robust model generalization. Moreover, real-world drone datasets often require meticulous annotation of drone instances and background clutter to facilitate accurate model training. Additionally, ethical considerations regarding the capture of drone data, particularly in sensitive or regulated airspace, necessitate adherence to legal frameworks and privacy guidelines. It is crucial to tackle these challenges to develop reliable detection models that can accurately identify drones in complex and changing environments.

To address the dataset challenge for drone detection research, we propose an approach for generating comprehensive datasets encompassing both visual and audio data. In our solution, we leverage Unreal Engine as a simulation environment and integrate the Airsim plugin for drone simulation. We have developed an Airsim GUI that interfaces with the Airsim plugin within Unreal Engine, facilitating the creation of datasets. Through the Airsim GUI, users can configure drones, set up cameras for visual data capture, define parameters for audio recording, and establish autonomous drone paths. This setup enables systematic generation of datasets that include synchronized visual and audio recordings, providing researchers with a diverse range of data to train and evaluate machine learning models designed for both visual and acoustic drone detection in dynamic environments.

## 1.1. Problem Definition

The thesis addresses the challenges of creating multi modal datasets essential for training and testing drone detection models, focusing on image and audio data. Significant hurdles include the high costs associated with acquiring drones and equipment, as well as the time-intensive setup and testing procedures. Additionally, the collection of synchronized image and audio data and the manual annotation process impose further complexities and costs on research projects. To address these challenges effectively, there is a critical need for a practical solution, a simulation system capable of efficiently generating synchronized datasets comprising visual and acoustic data. Such a system would facilitate the comprehensive training and evaluation of drone detection models in a cost-effective manner. Moreover, the developed system will enable testing the fusion of strategies for multi modal data effectively.

## 1.2. Key Contributions

The contributions of the thesis can be summarized as:

(i) A virtual drone simulation environment, integrating multiple data modalities, has been developed to streamline the testing of drone detection algorithms.

(ii) Airsim GUI, designed to be adaptable according to specific requirements, has been developed to interface with the simulation environment and gather datasets.

(iii) Offline audio generation has been integrated into the Airsim GUI to record and generate audio signals from multiple sources based on pre-recorded drone path data.

(iv) Image and audio-based drone detection techniques have been applied in various use cases, demonstrating their effectiveness in multiple scenarios such as in addressing the limitations posed by the low availability of real-world data.

## 1.3. Thesis Outline

To begin with, Chapter 3 covers a comprehensive review of the existing literature related to our topic. This chapter encapsulates the various drone detection datasets and methodologies that have been explored in previous research.

In Chapter 4, we delve into the overall architecture of our simulation and dataset generation process. This chapter provides a detailed overview of the technologies utilized, their configurations, the step-by-step workflow of the Airsim GUI for dataset generation, and the audio parameters, including offline generation capabilities.

Chapter 5 explores practical applications of the datasets generated through the Airsim GUI. This chapter presents performance metrics and a drone detection model that utilizes an image dataset enhanced by our simulated data. Additionally, we introduce an audio-based detection model using the simulation audio dataset and a model for estimating drone distance through audio data.

Lastly, Chapter 6 encapsulates the primary contributions of our research and discusses potential future directions for this work.

# 2. BACKGROUND INFORMATION

## 2.1. Machine Learning

Machine learning is a subset of artificial intelligence that focuses on the development of algorithms and statistical models that enable computers to perform specific tasks without explicit instructions. Instead, these systems rely on patterns and inference drawn from data. The core idea of machine learning is to create models that can generalize from a training dataset to make predictions or decisions based on new data [1].

There are several types of machine learning such as supervised learning, unsupervised learning and reinforcement learning. Supervised learning trains a model with a dataset that is labeled, which means that each training example is paired with an output label. Common algorithms include linear regression, logistic regression, support vector machines (SVM), and neural networks. Unsupervised learning involves training a model on data without labeled responses. The goal is to infer the underlying pattern that is present within a set of data points. Common algorithms include clustering methods like K-means and dimensionality reduction techniques like principal component analysis [2]. Reinforcement learning involves training a model to make a sequence of decisions by rewarding or punishing the model based on the outcomes of its actions. This type of learning is commonly used in robotics, security, environment-based games, and other areas where decision-making is crucial [3].

## 2.2. Artificial Neural Networks

Artificial neural networks are a subset of artificial intelligence and machine learning and are inspired by the structure and function of the human brain [4]. They consist of interconnected nodes and organized into layers. These layers include an input layer, one or more hidden layers, and an output layer.

The nodes in each layer are connected to nodes in the subsequent layer, forming a network.

The primary function of an artificial neural network is to map input data to the appropriate output. This is achieved through a process called training, where the network learns the parameters of the network from a set of labeled data. During training, the network adjusts the weights of the connections between nodes to minimize the error in its predictions. This is typically done using a technique called back-propagation, which calculates the gradient of the loss function and updates the weights accordingly [5, 6].

Artificial neural networks have proven to be effective in a wide range of applications, including image and speech recognition and natural language processing. Their ability to learn and generalize from data makes them a powerful tool in the field of artificial intelligence if there is an available labeled dataset [7].

### 2.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of artificial neural network designed specifically for processing structured grid data, such as images. Introduced by Yann LeCun and his colleagues in the late 1980s, CNNs have become the standard for most computer vision tasks due to their superior performance in image-related applications [8, 9].

The key innovation of CNNs lies in their convolutional layers, which evaluate a series of filters (also known as kernels) on the input data. Each filter convolves around the input image, detecting various features such as edges, textures, and shapes. This process is followed by pooling layers that down-sample the spatial dimensions, reducing the computational load and the number of parameters while preserving the important features.

CNNs typically consist of multiple convolutional and pooling layers, that is followed by multiple fully connected node layers that perform the final classification or regression tasks. In general, the hierarchical structure of CNNs allows them to capture low-level features in the initial layers and high-level features in the deeper layers.

One of the major advantages of CNNs is their ability to learn spatial hierarchies of features. This makes them particularly well-suited for tasks such as image recognition, object detection, and image segmentation. Over the years, various architectures of CNNs, such as VGG [10], ResNet [11], YOLO [12], and Inception [13], have been developed, each improving on the previous designs in terms of accuracy and efficiency.

### 2.4. Data Needs of CNNs

One of the critical factors influencing the performance of CNNs, is the availability of large and diverse datasets. These models require extensive training data to learn the underlying patterns and generalize well to new, unseen data. For instance, image classification tasks benefit significantly from large annotated datasets like ImageNet [14], which contains millions of labeled images across thousands of categories.

In the context of neural networks, more data helps in two significant ways:

- Improving Accuracy: Larger datasets provide more examples for the network to learn from, reducing overfitting and improving generalization.
- Uncovering Complex Patterns: More data allows the network to capture subtle and complex patterns that smaller datasets might miss.

CNNs require substantial amounts of labeled images to achieve high performance. Additionally, CNNs benefit from data augmentation techniques, which artificially expand the training dataset by applying random transformations to the input images. This helps the model learn to recognize objects under various conditions, improving robustness and accuracy.

In summary, the success of neural networks and CNNs heavily depends on the quantity and quality of the training data. Advances in data collection, labeling, and augmentation have been crucial in driving the progress and capabilities of these models.

# 3. RELATED WORKS

## 3.1. Image and Audio-Based Drone Detection Datasets

In recent years, the development of image datasets specifically for drone detection has gained significant attention due to the growing prevalence of drones and the need for effective detection mechanisms. Unlike datasets that contain images captured by drones, these datasets focus on images of drones taken from the ground or other aerial platforms.

One of the notable contributions in this area is the "Anti-UAV" dataset introduced by Jiang, Nan, et al (2020) [15]. The Anti-UAV dataset consists of more than 300 video pairs that contain 585.9k manually annotated bounding boxes in various environments and from different viewpoints, including urban, rural, and coastal areas. This dataset aims to facilitate the training and evaluation of drone detection algorithms by providing a diverse set of challenging scenarios.


Figure 3.1. Sample image from Anti-UAV dataset.

The “Real World Object Detection Dataset for Quadcopter Unmanned Aerial Vehicle Detection” by Maciej Ł. Pawełczyk et al. (2020) [16] introduces a comprehensive dataset designed to train machine learning algorithms for the specific task of drone detection using industrial camera feeds. This dataset addresses the growing need for affordable and effective drone detection systems that can operate on low-performance hardware. The dataset comprises 56,821 images and 55,539 bounding boxes, all hand-labeled from real-world footage. The images include a wide variety of drone models and flight scenarios, enhancing the dataset’s robustness and applicability to diverse detection tasks. The authors also explored semi-automated labeling techniques to improve efficiency. This dataset expands existing image classification and object detection datasets, such as ImageNet [14], MS-COCO [17], PASCAL VOC [18], and Anti-UAV [15], by providing a specialized and extensive collection of drone images. The dataset was used to train and evaluate both haar cascades and deep neural networks, demonstrating the feasibility of large-scale drone detection systems based on machine learning.

The “Drone-vs-Bird Detection Challenge” dataset, introduced by Rozantsev et al. (2017) [**?**], addresses a crucial aspect of aerial surveillance: distinguishing between drones and birds. This dataset is specifically designed to aid in the development and evaluation of algorithms capable of differentiating drones from birds, which is a significant challenge due to their similar visual characteristics when observed from a distance. The dataset comprises thousands of annotated images and video sequences that capture various drone and bird models in flight. These images are taken under different environmental conditions, including varying lighting and weather scenarios, providing a diverse and challenging set of data for testing detection algorithms.

Figure 3.2. Frame taken from a video sequence in Drone vs Bird Dataset where both drone and multiple birds exists.

The use of audio data for drone detection has emerged as a complementary approach to visual methods, particularly in scenarios where visibility is limited.

In "Audio Based Drone Detection and Identification using Deep Learning" [19] a specialized dataset was created to train deep learning models for drone detection and identification using audio signals. This dataset includes over 1300 publicly accessible drone audio clips [20] augmented with noise from publicly available datasets [21, 22] to simulate real-world environments. The recordings were captured using an iPhone's integrated microphone at a 44.1KHz sampling rate, stored in MPEG-4 audio format (m4a), and segmented into one-second intervals for optimized model training and real-time deployment. Data labeling differentiated between drone types (Bebop, Mambo) and categorized audio clips into drone vs. non-drone for detection purposes, ensuring balanced representation across labels.

### 3.2. Image and Audio-Based Drone Detection Techniques

Using cameras and advancements in computer vision and deep learning, extensive research has been conducted for drone detection.

A recent study [23] addressed these challenges by proposing a one-shot detector, You Only Look Once version 5 (YOLOv5). This model was trained using pre-trained weights and data augmentation, achieving a mean average precision (mAP) of 90.40%, which is a 21.57% improvement over the previous YOLOv4-based model. The study demonstrated the effectiveness of YOLOv5 in real-time drone detection using highly configured devices such as GPUs.

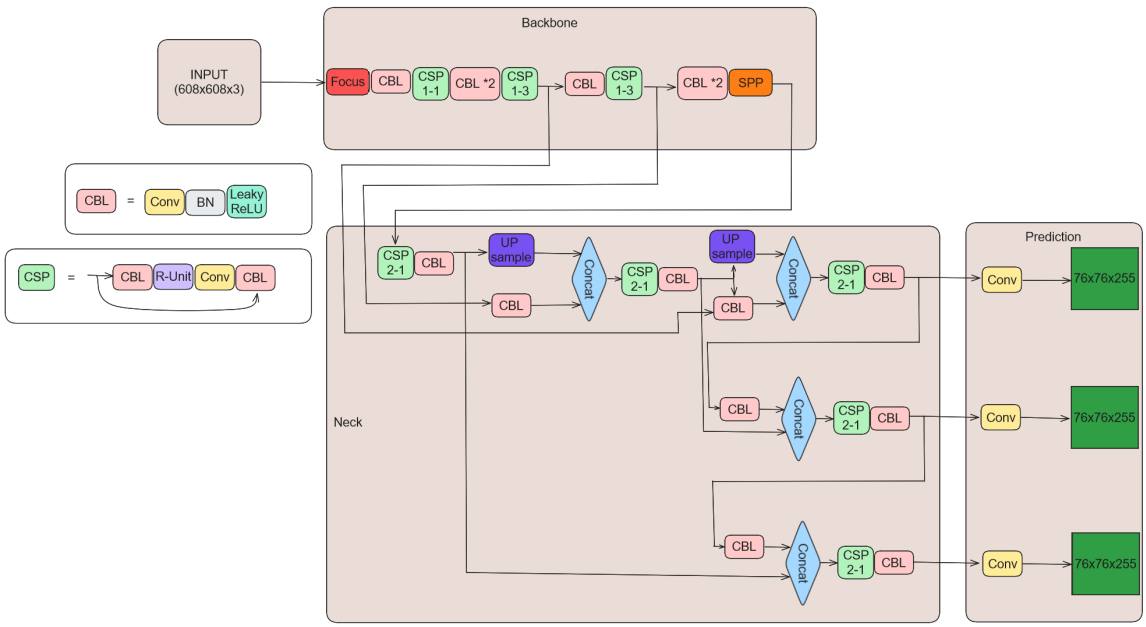


Figure 3.3. YOLOv5 network architecture (adapted from [24]).

The authors of [25] employed the drone-vs-bird dataset to devise a solution tailored for static cameras. They approached the drone detection challenge by splitting it into two distinct phases: detection of moving objects and their subsequent classification. For detecting moving objects, they utilized a background subtraction technique, while MobileNetV2 was employed for classifying the detected objects into drones, birds,

or backgrounds. Their evaluation revealed an F1-score of 0.742 with an Intersection over Union (IoU) threshold of 0.3.

The authors of [26] utilized newer version of YOLO named YOLOv8, and improved on the existing model to enhance its performance. Initially, using the standard small YOLOv8 model (YOLOv8s), they achieved a mean average precision (mAP) score of 85.2%. Through their optimizations, they were able to improve this score significantly, reaching a mAP of 95.3%.

[27] explores a machine learning-based framework for detecting drones using their unique acoustic signatures. Similar to [28], the system captures environmental sounds, processes them into digital format, and normalizes the data. Short-term and mid-term analyses extract meaningful features from audio frames, which are then used to create signature vectors. These vectors are classified by Support Vector Machines (SVM) to detect the presence of drones. The framework demonstrates effectiveness in recognizing drone sounds and provides a valuable approach to enhancing security in critical areas through acoustic monitoring.

Another work [29] introduces a real-time multi-camera system designed to detect, track, and localize multiple moving drones in 3D space. A key innovation is its target re-identification process, which fuses information from various cameras based on drone trajectories and relative locations. The system detects drones using motion-based blob detection and tracks their 2D locations within individual camera frames via a geometry and camera-based model. By cross-correlating these tracks among cameras, the system integrates the 2D positions into a global 3D position for all tracked drones. In outdoor tests with two drones and three cameras, the system achieved an average positional error of 8%, demonstrating its effectiveness in accurately tracking and re-identifying drones.

Acoustic-based drone detection has gained traction due to its ability to detect drones in visually hurdled environments [30].

In [19], researchers introduced methods for drone detection and identification using audio data and various deep learning models such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Convolutional Recurrent Neural Network (CRNN). Their approach not only detected the presence of drones but also identified the type of drone based on their unique audio signatures. Empirical validation supported the effectiveness of using deep learning techniques, corroborating previous literature. Through multiple experiments, they concluded that CNNs outperformed RNNs and CRNNs in accuracy, F1 score, CPU training and testing time, precision, and recall. However, the performance difference between CNNs and CRNNs was minimal, suggesting that CRNNs might be more practical due to their faster speed and stable outcomes.

Another study [31] explored the design of an automatic multi-sensor drone detection system incorporating video, audio, and thermal infrared sensors. The inclusion of a thermal infrared camera, despite its slightly lower resolution, proved to be as effective as visible range cameras for drone detection. The study also examined detector performance relative to sensor-to-target distance and demonstrated that sensor fusion enhances system robustness, reducing false detection cases. To address the lack of public datasets, the researchers introduced a novel dataset containing 650 annotated infrared and visible videos of drones, birds, airplanes, and helicopters, complemented by an audio dataset featuring drones, helicopters, and background noise.

### 3.3. Drone Detection in Simulation Environments

Simulation environments for drone detection have emerged as an effective tool for developing and testing detection algorithms in a controlled and reproducible setting. These environments allow researchers to create diverse scenarios that would be challenging or impractical to replicate in the real world, enabling more comprehensive evaluation of detection methods. Moreover, obtaining the ground truth is much more easier and reliable compared to experiment-based datasets.

Table 3.1. Performance comparison of related works.

| **Paper** | **Model** | **Data Type** | **Precision** | **Recall** | **F1-Score** | **Goal** |
|---|---|---|---|---|---|---|
| [23] | YOLOv5 | 2395 Image | 0.918 | 0.875 | 0.896 | Drone vs bird |
| [25] | MobileNetV2 | 24075 Image | 0.701 | 0.788 | 0.742 | Drone vs bird |
| [26] | Optimized YOLOv8 | 2850 Image | 0.97 | 0.895 | - | Drone detection |
| [31] | YOLOv2 | 37519 Image | 0.808 | 0.7636 | 0.7849 | Visible camera drone detection |
| [31] | YOLOv2 | 37428 Image | 0.8169 | 0.7179 | 0.7601 | Infra camera drone detection |
| [19] | CNN | 1300 Audio | 0.9624 | 0.9560 | 0.9590 | Binary Drone Detection |
| [19] | CNN | 1300 Audio | 0.9275 | 0.9263 | 0.9263 | Drone Identification |
| [31] | LSTM-RNN | 90 Audio | 0.9354 | 0.9293 | 0.9323 | Drone, helicopter, background identification |

In [32], researchers generate synthetic data for training deep learning-based drone detection system using Unreal Engine [33] and AirSim [34]. Results indicate that training exclusively on synthetic data yielded a decent performance on synthetically generated test sets but limited validity and performance in real-world scenarios. Adding a small amount (0.2-1%) of real data to synthetic data through fine-tuning or mixed-training significantly improved mean Average Precision (mAP) and reduced False Negative Rates (FNRs) on real-world tests.

In [35], researchers created a custom environment in Unreal Engine and utilized AirSim to capture photorealistic images for generating a synthetic dataset. They equipped the drones with two cameras using AirSim's camera model to match the specifications of the ZED camera. Simulating two drones, they observed one from the perspective of the other and generated segmentation through AirSim. The dataset was then used to train a neural network.

# 4. SIMULATION ENVIRONMENT

## 4.1. Architecture

We created a simulation environment that can be used to collect multi-modal data such as images and audio. Users can create experiments with desired components and the environment where datasets with ground truth labels can be generated easily.

Our simulation environment consists of three main parts: a user interface named AirSim GUI, Unreal Engine as the game engine, and a file system. These components interact with each other based on user actions, such as generating audio or image datasets. All actions are triggered by the AirSim GUI. In this architecture, users can change any underlying component, such as the drone model or environment version in Unreal Engine, without affecting the dataset creation process.

In the scope of this thesis, the architecture has two main dataset creation capabilities: image and audio generation. To create an image dataset, AirSim GUI sends requests to AirSim plugin in Unreal Engine and triggers the data collection process. AirSim GUI uses AirSim Python client to coordinate the tasks. As shown in Figure 4.1, while the AirSim Plugin is needed for image recording, it is not required for audio recording. This feature is particularly useful as it allows us to create audio data using pre-recorded paths. After an experiment, all simulation data is saved to a user-defined directory in the file system. This simulation data includes images, audio clips, vehicle logs, and camera logs. The vehicle logs consist of the drone's position and velocity at each timestamp. The camera logs include the timestamp, image type, and saved image path.

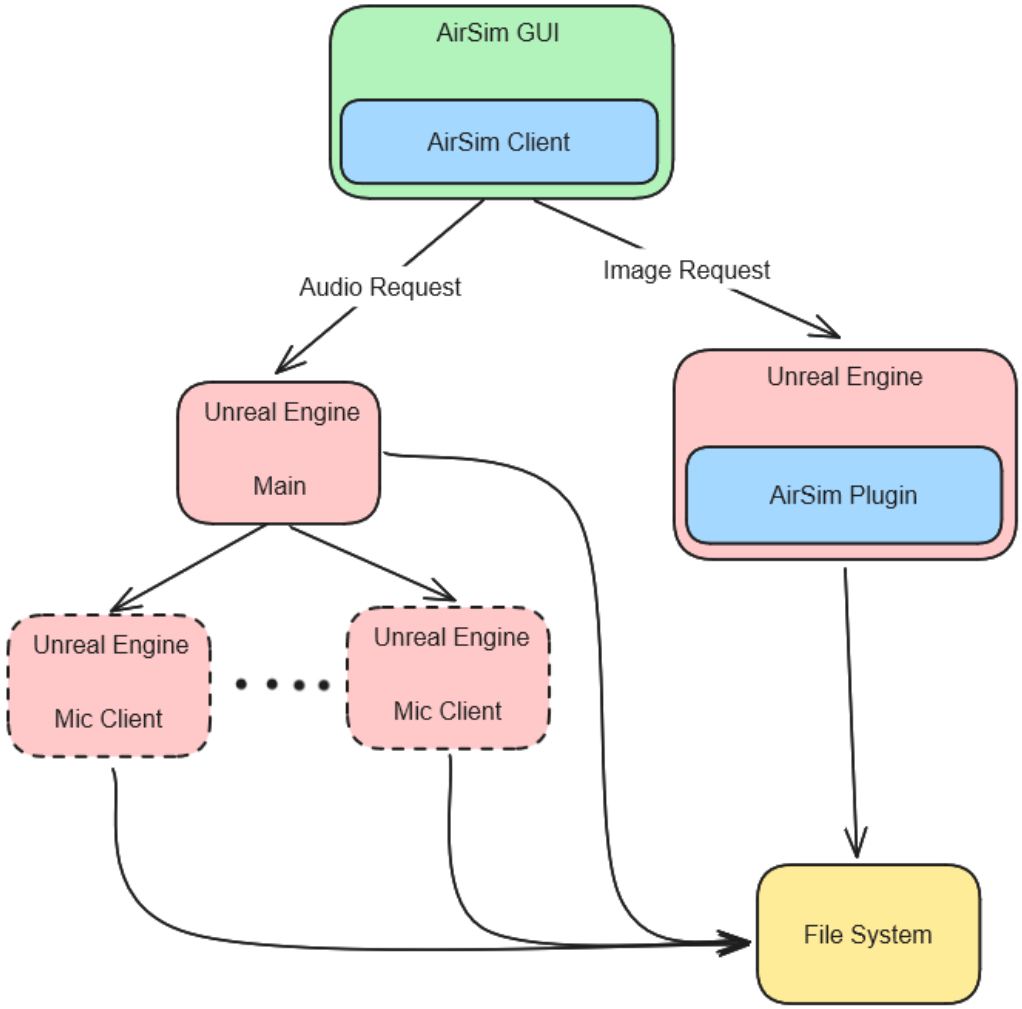


Figure 4.1. The simulation architecture.

## 4.2. Technologies

In this section, we will detail the technologies utilized in the simulator.

### 4.2.1. Unreal Engine 4

Unreal Engine [33] is a powerful and versatile game engine developed by Epic Games. It is widely used for creating high-quality, immersive games and simulations due to its robust features and advanced capabilities. It is chosen as the simulation

environment mainly because of its AirSim plugin compatibility. We explained some of its main features and differences to other game engines in this section.

Unreal Engine is a complete suite of development tools designed for anyone working with real-time technology. It offers a high degree of flexibility and control over the development process, providing a comprehensive set of tools for 3D modeling, animation, lighting, physics, and more. Some of its core features include:

- Real-Time Rendering: Unreal Engine's real-time rendering capabilities allow developers to create photorealistic visuals with dynamic lighting and shadows.
- Blueprints: A visual scripting system that enables developers to create gameplay mechanics and complex behaviors without writing code.
- C++ Support: For developers who prefer traditional coding, Unreal Engine provides extensive support for C++, allowing for deep customization and performance optimization.
- Physics and Destruction: Advanced physics simulation and destruction systems enable realistic interactions and environmental effects.
- Animation Tools: Comprehensive tools for creating and editing animations, including the Animation Blueprint system for procedural animations.
- Marketplace: A vast library of assets, plugins, and tools available for purchase or free download, accelerating development and reducing the need for creating everything from scratch.

Unreal Engine stands out from other game engines, such as Unity and CryEngine, in several ways:

- Visual Quality: Unreal Engine is renowned for its high visual fidelity and realistic graphics, often preferred for AAA game development and high-end simulations.
- Blueprint System: The Blueprints visual scripting system is unique to Unreal Engine, providing an accessible way for designers and artists to implement functionality without extensive programming knowledge.

- Built-in Tools: Unreal Engine comes with a wide array of built-in tools and features, reducing the need for third-party plugins and extensions.
- Performance: Unreal Engine is optimized for performance, capable of handling complex scenes and high-end graphics with efficient resource management.
- Source Code Access: Unreal Engine provides full source code access, allowing developers to modify and extend the engine to meet specific project needs.

### 4.2.2. AirSim

AirSim [34] is an open-source, cross-platform simulator developed by Microsoft Research. It is designed to provide a realistic and high-fidelity simulation environment for autonomous vehicles, including drones and cars. AirSim supports both software-in-the-loop (SITL) and hardware-in-the-loop (HITL) simulations, making it an invaluable tool for research and development in areas such as computer vision, robotics, and machine learning.

AirSim leverages Unreal Engine's advanced rendering and physics capabilities to create realistic simulation environments. The integration of AirSim with Unreal Engine involves several key components and steps:

(i) Installation and Setup: To use AirSim with Unreal Engine, you must first install Unreal Engine and the AirSim plugin. The plugin can be downloaded from the AirSim repository [36] and added to an Unreal Engine project. This involves copying the plugin files into the project's plugin directory and enabling the plugin within the Unreal Engine Editor.

(ii) Creating a Simulation Environment: Once the plugin is installed, users can create custom simulation environments within Unreal Engine. This involves designing and configuring the environment, including terrain, buildings, obstacles, and other elements relevant to the simulation. Unreal Engine's powerful editor tools make it easy to create detailed and interactive environments.

(iii) Configuring AirSim Settings: AirSim provides a configuration file (settings.json)

that allows users to customize various aspects of the simulation, such as the type of vehicle, sensor configurations, and simulation parameters. This file can be edited to match the specific requirements of the simulation scenario.

(iv) Running the Simulation: After setting up the environment and configuration, users can run the simulation directly within Unreal Engine. AirSim interacts with Unreal Engine to simulate the vehicle's dynamics, capture sensor data, and provide a realistic virtual testing ground.

AirSim provides a comprehensive Python API, enabling users to control the simulation and interact with the simulated vehicles programmatically. This allows for automation, data collection, and integration with machine learning workflows. The Python API can be used to perform various tasks such as:

- Connecting to the Simulation: To interact with the simulation, users can establish a connection using the AirSim Python client. This involves importing the AirSim Python library and creating a client object that communicates with the AirSim server running in Unreal Engine.
- Controlling the Vehicle: The Python API allows users to send control commands to the simulated vehicle. For drones, this includes commands for takeoff, landing, and waypoint navigation.
- Capturing Sensor Data: AirSim supports various sensors, including cameras, lidar, and GPS. The Python API can be used to capture and process sensor data in real-time.
- Data Collection and Analysis: The Python API facilitates the collection of large datasets for training and evaluating machine learning models. Users can automate data collection processes, label data, and perform analysis on the collected data.

### 4.2.3. PyQt5

PyQt [37] is a set of Python bindings for the Qt application framework, enabling the creation of cross-platform applications with graphical user interfaces (GUIs). PyQt

combines the simplicity and flexibility of Python with the comprehensive features and performance of the Qt framework. It is widely used for developing desktop applications that require robust and responsive user interfaces.

PyQt provides a wide range of tools and components for building GUIs, including:

- Widgets: PyQt offers a rich set of standard widgets, such as buttons, labels, text fields, and combo boxes, which can be easily customized and extended.
- Layouts: PyQt includes powerful layout managers to arrange widgets in a flexible and consistent manner. These include horizontal and vertical boxes, grids, and form layouts.
- Signals and Slots: This mechanism allows for easy event handling and communication between objects, making it simple to respond to user interactions.
- Styles and Themes: PyQt supports various styles and themes, allowing developers to create visually appealing interfaces that are consistent across different platforms.
- Graphics View Framework: This provides a framework for managing and interacting with custom 2D graphics items, supporting advanced graphics effects and transformations.
- Internationalization: PyQt includes tools for translating applications into different languages, making it easy to develop multilingual applications.

To create the AirSim GUI, we used Qt Designer, which is a powerful tool for designing and building graphical user interfaces (GUIs) with the Qt application framework. It provides a visual environment for creating and arranging widgets, setting properties, and defining the layout of your application. Qt Designer allows developers to create complex interfaces quickly and efficiently without writing extensive code manually.

## 4.3. Simulation Level

In this section, we will discuss the design and implementation of our simulation environment using Unreal Engine. The area chosen for the simulation replicates specific locations on a university campus, providing a realistic backdrop for our drone detection tests.

### 4.3.1. Environment Selection

We selected a portion of the university campus for our simulation, covering notable landmarks and buildings such as the Aptullah Kuran library, computer engineering building, and surrounding areas. This area spans approximately 11,500 square meters, ensuring a comprehensive simulation space. A birdeye view of the chosen area is depicted in Figure 4.2, while a detailed model of the main library is shown in Figure 4.3 and its real life counterpart in Figure 4.4.


Figure 4.2. Birdeye view of the selected simulation environment.

Figure 4.3. Aptullah Kuran library in simulation.

Figure 4.4. Aptullah Kuran library in Google Street View [38].

### 4.3.2. Two-Phase Development Approach

Our level design process is divided into two distinct phases: material creation and material integration.

- Material Creation:
  - Using a DJI Mavic Air 2 drone, we captured high-resolution images of the selected buildings and their surroundings.
  - These images were processed and combined in Blender to create accurate 3D models. The models were scaled to a 1:1 ratio to ensure realism in the simulation environment.
- Material Integration:
  - The created 3D models were exported from Blender and imported into Unreal Engine.
  - In Unreal Engine, we assembled these models to reconstruct the selected campus area, maintaining the 1:1 scale for consistency. Figure 4.5 showcases various views of the completed simulation environment.

### 4.3.3. Additional Environmental Features

To enhance the realism of our simulation, we incorporated dynamic elements:

- Birds: Using a free bird blueprint available in Unreal Engine marketplace, we added birds to the environment. A bounding box object was used to define the flying area for the birds, and multiple birds could be added within this box. We mapped the actions to add birds to specific keyboard commands "4", "5", and "6" which adds 10, 20 and 30 birds respectively.
- Day-Night Cycle and Weather: To simulate different times of the day, we implemented a functionality to switch between daytime and nighttime environments using the "7" and "8" keys. This was achieved through blueprints, adding another layer of realism to our simulation.

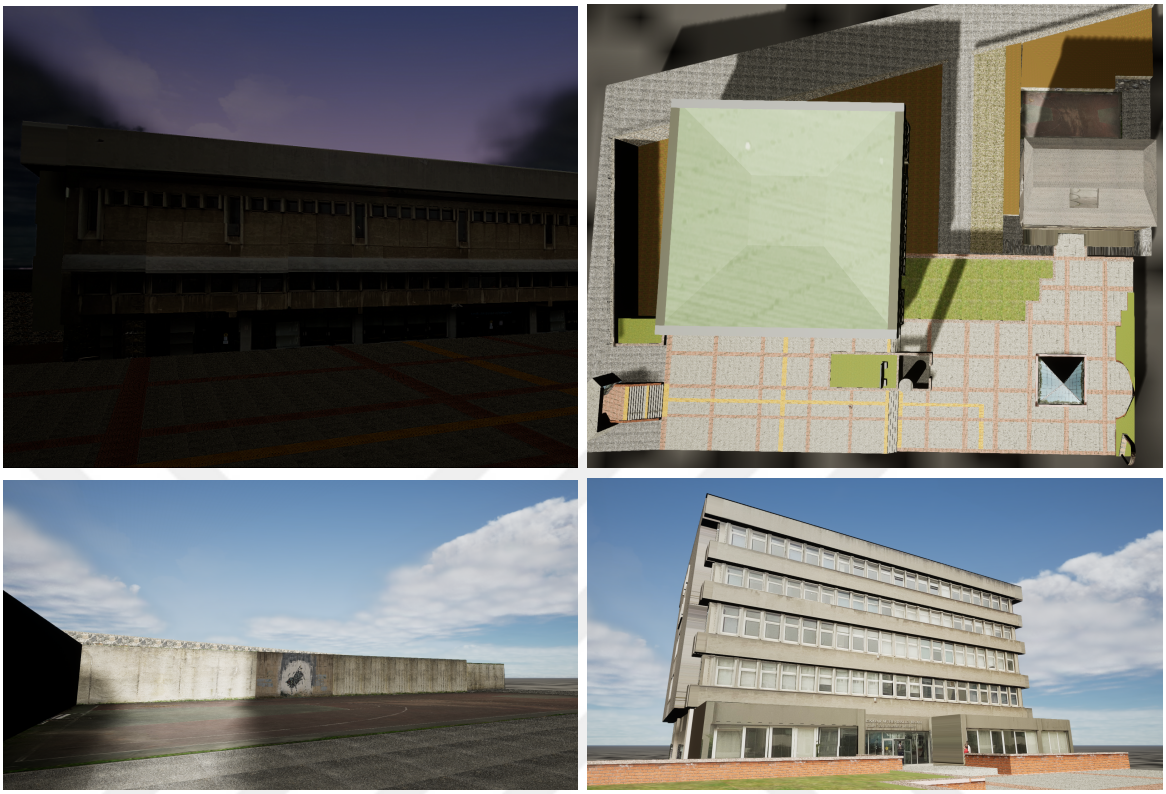

Figure 4.5. Images from the simulation environment under different conditions. Top left image: Aptullah Kuran library at night. Top right image: Top down view of the environment. Bottom left image: The basketball court. Bottom right image: Computer Engineering building.

By following these steps, we created a detailed and dynamic simulation environment in Unreal Engine, tailored to our specific needs for testing drone detection systems.

## 4.4. AirSim GUI Capabilities

### 4.4.1. Main Capabilities

AirSim GUI provides comprehensive controls and features for managing drone simulations and recording data, including images and audio. This section explores the

various capabilities of the AirSim GUI, highlighting its functionality and versatility in supporting research objectives.

4.4.1.1. Spawning Drones. AirSim GUI allows users to spawn drones at specified coordinates (x, y, z). This capability provides precise control over the initial positioning of drones within the simulation environment, enabling tailored scenarios for testing and data collection.

4.4.1.2. Camera Integration. Users can define cameras through the AirSim settings file. AirSim GUI allows users to add these cameras into the simulation environment. Cameras can be added to an external location or attached directly to drones, facilitating a wide range of recording configurations. The types of cameras available include:

- Scene: Captures standard RGB images of the environment, similar to what a human eye would perceive. Ideal for visual documentation and creating datasets for computer vision tasks like object detection and classification.
- Depth Planar: Captures depth information using a planar projection, where the distance from the camera to objects in the scene is recorded. Useful for applications requiring distance measurement and 3D reconstruction.
- Depth Perspective: Similar to Depth Planar, but captures depth information with a perspective projection, which more accurately mimics human vision. Suitable for augmented reality (AR) and virtual reality (VR) applications, where depth perception is critical.
- Depth Vis: Visualizes depth information by encoding it into a color image, where different colors represent different distances. Helps in quickly understanding the depth distribution within a scene, useful for debugging and visualization.
- Disparity Normalized: Captures disparity maps, which represent the difference in image location of an object seen by two cameras. Commonly used in stereo vision systems to estimate depth and for applications in autonomous driving and robotics.

- Segmentation: Produces images where each object in the scene is assigned a unique color or label. Essential for training segmentation models in computer vision, where distinguishing between different objects is necessary.
- Surface Normals: Captures images where each pixel represents the surface normal at that point, indicating the direction perpendicular to the surface. Useful in rendering and 3D modeling, providing information about the geometry and orientation of surfaces.
- Infrared: Simulates infrared imaging, capturing heat signatures rather than visible light. Critical for applications in surveillance, search and rescue operations, and night-time navigation.
- Optical Flow: Measures the motion of objects between consecutive frames, capturing the direction and speed of movement. Important for motion analysis, video stabilization, and dynamic scene understanding.
- Optical Flow Vis: Visualizes optical flow information, encoding motion vectors into a color-coded image. Useful for debugging and analyzing the motion captured in the scene, providing an intuitive way to understand optical flow data.

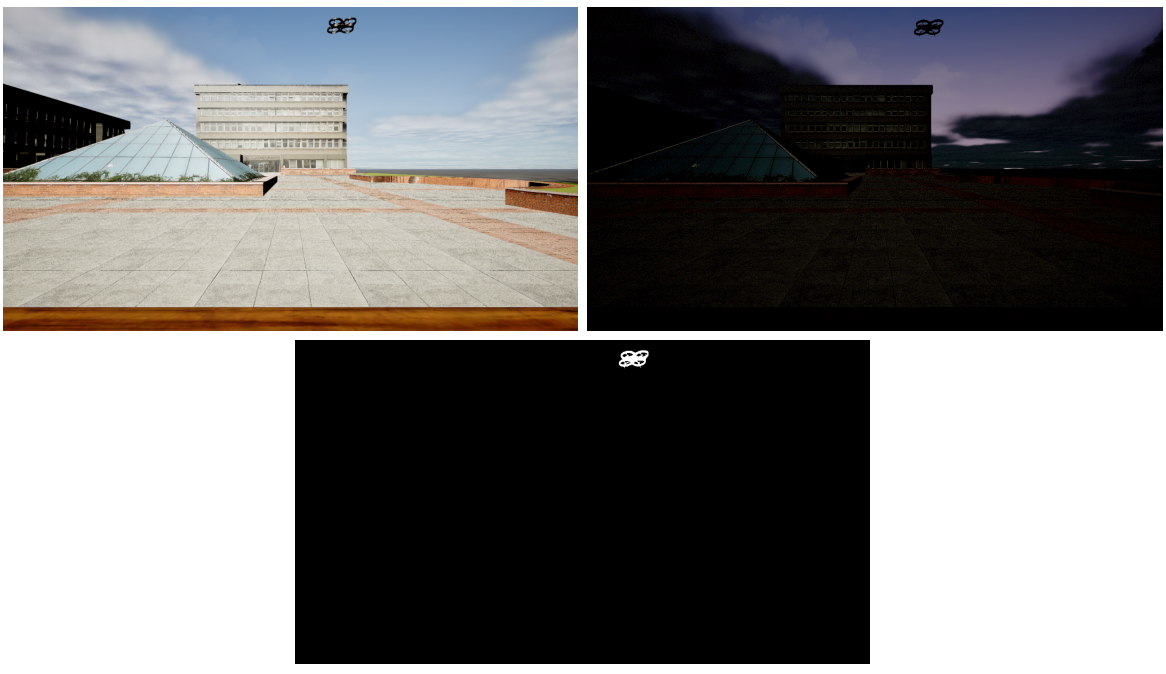

Figure 4.6. Top Left: Scene camera, day time. Top Right: Scene camera, night time. Bottom: Segmentation camera.

These camera options offer diverse imaging capabilities, from capturing realistic scenes to specialized views like depth perception and infrared imaging.

4.4.1.3. Camera Preview. AirSim GUI supports real-time previewing of cameras at any point in the simulation, whether before or during recording. This feature ensures that camera placements and settings can be verified on-the-fly, enhancing the accuracy and quality of the recorded data.

4.4.1.4. Path Management. Using a top-down map interface, users can set and edit drone paths within the AirSim GUI. This visual approach simplifies the process of defining complex flight patterns and trajectories. Users can select $x$ and $y$ coordinates by clicking and $z$ coordinate by scroll wheel.

The GUI provides options to save created paths for future use or load previously saved paths. This functionality enables repeatable and consistent simulations, essential for systematic testing and data collection.

4.4.1.5. Recording. The AirSim GUI facilitates the recording of images through the integrated cameras. Multiple cameras can record simultaneously, allowing for comprehensive data capture from various perspectives. Additionally, the GUI includes an option to save segmented images, where the drone is segmented as white and all other elements as black. This segmentation is particularly useful for tasks requiring precise drone localization and analysis.

Users can enable drone path following either at the start of the recording or at any chosen point during the simulation. This ensures that the drone's movements can be accurately synchronized with the recording process.

The AirSim GUI includes an option to start audio generation following the recording. This process initiates multiple Unreal Engine instances, based on the specified con-

figurations, to capture and simulate realistic drone sounds. In addition to that, AirSim GUI supports offline audio generation. This feature allows users to generate audio for a previously run simulation, providing flexibility in post-processing and analysis.

After a recording, simulation data is saved to a user-defined directory in the file system. This simulation data includes images, audio clips, vehicle logs, and camera logs. The data is organized into three main folders: "camera_logs", "vehicle_logs", and "audio_recordings". Camera logs and vehicle logs folder contains multiple text files, where the number of files corresponds to the number of cameras and drones used in the experiment. The text files in "vehicle_logs" folder contain logs of the drone's position and velocity at each timestamp. The text files in "camera_logs" folder includes logs with timestamps, image types, and saved image paths. The "audio_recordings" folder stores audio clips recorded during the simulation for each microphone. Additionally, for each camera, a separate folder is created in the main directory to store the recorded images. This folder structure ensures that all simulation data is organized and easily accessible.

### 4.4.2. Audio Data Generation Details and Offline Usage

Airsim GUI has the capability to collect audio data. Users can either provide previously recorded drone position logs or record audio automatically after recording images. Also, users need to provide simulation executable and airsim settings file.

4.4.2.1. Audio in Unreal Engine. Audio plays a crucial role in creating immersive and realistic environments in virtual simulations. It enhances the experience by providing contextual cues, depth, and spatial awareness. Unreal Engine's audio system provides a comprehensive and flexible framework for simulating realistic sound environments. The engine's detailed attenuation settings, spatialization capabilities, and support for audio effects ensure that sound behaves naturally, enhancing the overall realism of virtual environments.

Sound simulation in virtual environments involves accurately replicating how sound waves propagate, reflect, and interact with various surfaces. Key concepts in sound simulation include attenuation, path loss, and reverberation.

Attenuation refers to the reduction in sound intensity as it travels through a medium. Path loss accounts for the reduction in signal strength as it travels through a medium. Reverberation is the persistence of sound in a space after the original sound is produced, caused by multiple reflections.

Unreal Engine provides extensive tools and features for handling audio. It supports 3D spatialization, attenuation, and various audio effects to enhance the simulation. The engine uses a combination of sound cues, sound classes, and sound attenuation settings to manage audio.

- Sound Cues: Sound cues are assets that define how sounds are played in the game. They can be simple (playing a single sound) or complex (mixing multiple sounds, applying effects, or creating randomized playback).
- Sound Classes: Sound classes allow developers to group sounds and apply common properties, such as volume adjustments or audio effects, to all sounds within the class.
- Audio Effects: Unreal Engine supports various audio effects, including reverb, echo, and EQ adjustments. These effects can be applied to individual sounds or entire sound classes.
- Spatialization: Spatialization is the process of simulating 3D sound, making it appear as though sounds are coming from specific locations in the virtual environment. Unreal Engine uses HRTF (Head-Related Transfer Function) algorithms to achieve realistic spatialization.

Sound attenuation in Unreal Engine is a crucial aspect of creating realistic audio. Attenuation settings determine how sound diminishes with distance, simulating the natural behavior of sound in the real world.

Unreal Engine provides various models and parameters for configuring sound attenuation.

Unreal Engine supports several distance models, including linear, logarithmic, inverse, log reverse, and natural sound. These models define how sound intensity decreases with distance.

- Linear Model: Sound intensity decreases linearly with distance.
- Logarithmic Model: Sound intensity decreases logarithmically with distance.
- Inverse Model: Sound intensity decreases according to the inverse square law. The changes in volume are more exaggerated but similar to that of the logarithmic curve.
- Log Reverse Model: A custom model where sound intensity follows a logarithmic increase rather than a decrease, used for specific audio effects.
- Natural Sound Model: Mimics the complex attenuation characteristics of real-world sounds, combining multiple factors such as frequency-dependent attenuation and environmental interactions.

Unreal Engine allows for various attenuation shapes that define the volume within which sound attenuation is applied. These shapes help to create realistic audio experiences by simulating how sound propagates in different environments. Common attenuation shapes include:

- Sphere: Attenuation radiates uniformly in all directions from the sound source, suitable for omnidirectional sounds.
- Capsule: Attenuation follows a capsule shape, useful for elongated sound sources such as a river or wind along a path.
- Box: Attenuation is confined within a rectangular box, ideal for sounds originating from or confined to a specific area, such as a room.
- Cone: Attenuation is directional, simulating how sound propagates from a source that has a focused direction, like a megaphone or a spotlight.

Unreal Engine also allows developers to customize attenuation settings, such as the minimum and maximum distances for attenuation, falloff distance, and the shape of the attenuation curve. These settings provide fine-grained control over how sound behaves in the simulation environment.

Unreal Engine can simulate occlusion and obstruction, where sound is blocked or muffled by objects in the environment. This adds an extra layer of realism, as sounds dynamically change based on the listener's position and surrounding objects.

4.4.2.2. Simulation Audio Parameters. This section details the specific audio parameters used in Unreal Engine to simulate the DJI Mavic Air 2 drone sound. The decisions behind each parameter setting are also discussed to provide insights into their relevance and impact on the simulation's authenticity.

The audio for the DJI Mavic Air 2 drone was captured using a smartphone while the drone was airborne but stationary. This recording was then integrated into the Unreal Engine's drone model to simulate realistic drone sounds during the simulation.

The sphere shape was chosen for drone audio attenuation, as illustrated in Figure 4.7. This shape provides a natural and uniform attenuation of sound in all directions, mimicking how sound propagates from a point source in real-world conditions.

As an attenuation function, we chose natural sound. Using the natural sound attenuation function ensures that the drone audio diminishes in a manner similar to real-world acoustic behavior. This choice enhances the realism of the simulation by providing an authentic audio.

We enabled spatialization since it allows the audio to be perceived in 3D space, giving the recording a sense of direction and distance of the sound source. This is vital for accurately simulating the drone's position relative to the microphones.

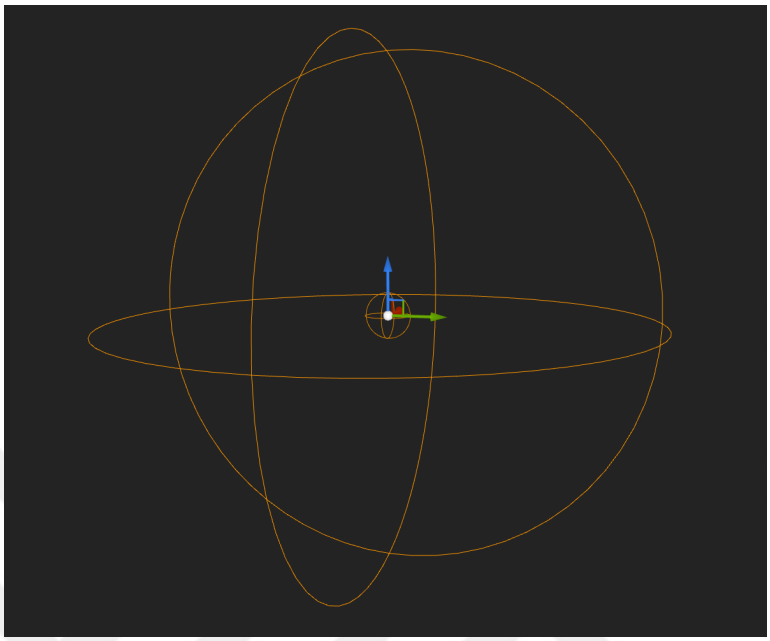

Figure 4.7. Drone audio attenuation shape as sphere.

We set the attenuation at max to $-60\,\mathrm{dB}$. This ensures that the drone sound can be heard from a significant distance, but it diminishes enough to simulate realistic distance-based audio attenuation.

An inner radius of 100 units (1 meter) establishes a zone around the drone where the sound remains unchanged. This setting represents the immediate vicinity of the drone, where audio should be at its most intense. The falloff distance of 10000 units (100 meters) defines the range over which the sound attenuates from its maximum intensity to the minimum threshold. This large radius ensures that the drone sound gradually diminishes, providing a realistic audio decay over distance.

We enabled occlusion which allows the simulation to account for objects obstructing the sound path, providing a more accurate representation of how sound behaves in complex environments. We set the occlusion volume attenuation to 0.9 and occlusion interpolation time to 0.1. Setting the occlusion volume attenuation to 0.9 means that occluded sounds are reduced in volume, simulating the effect of physical barriers between the sound source and the listener. An occlusion interpolation time of 0.1 sec-

onds ensures that changes in occlusion are applied swiftly, maintaining the realism of dynamic environments where objects frequently move in and out of the sound path.

We also enabled complex collision for occlusion to ensure that detailed and accurate collision detection is used to determine occlusion, enhancing the fidelity of the audio simulation. The difference between simple and complex collision is shown in Figure 4.8.

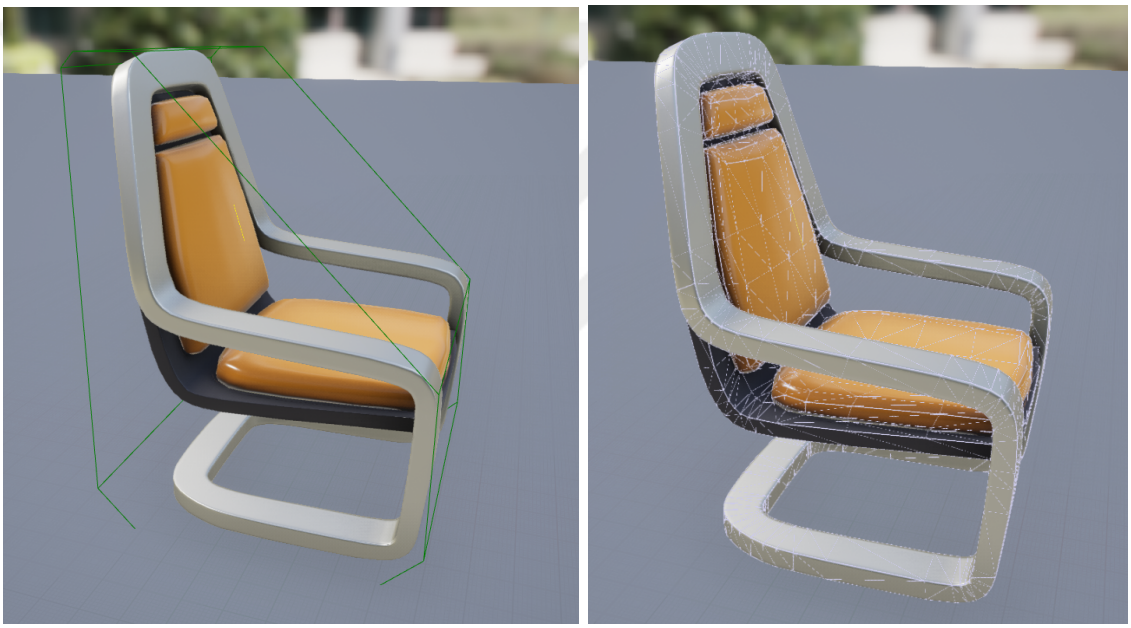

Figure 4.8. Difference between a simple collision (left) and a complex collision (right).

We activated randomization on pitch and volume to min 0.95 and max 1.05. Randomizing the pitch within a range of 0.95 to 1.05 introduces slight variations in the drone sound, preventing it from sounding monotonous and artificial. These minor pitch changes enhance the realism of the audio. Similarly, randomizing the volume within the same range adds subtle variations in loudness, mimicking natural fluctuations in drone noise and further contributing to the authenticity of the simulation.

The careful selection of these audio parameters ensures that our simulation provides a realistic audio. By accurately representing the drone's sound characteristics

and incorporating realistic attenuation, occlusion, reverb, and randomization settings, we create a robust environment for developing and testing audio-based drone detection models. These parameters not only enhance the simulation's fidelity but also contribute to the effectiveness of the detection algorithms that rely on precise and authentic audio data.

4.4.2.3. Audio Collection Steps. To capture audio from multiple microphones, we initiate several instances of Unreal Engine, corresponding to the number of microphones being used. For example, if the user intends to record audio from four microphones, four instances of Unreal Engine are launched. One of these instances serves as the main instance, while the others follow its commands. Running Unreal Engine in low resolution optimizes overall performance without compromising audio quality.

Each Unreal Engine instance creates a recording object and positions it at the coordinates of a designated microphone. This setup ensures that audio is captured accurately from each microphone's location.

The process for handling drone path logs depends on whether the user wishes to record audio after capturing image data or independently. If the user opts to record audio following image data collection, the main instance reads the last recorded drone path logs. If the user chooses to record audio independently, they must provide previously recorded drone path logs.

Upon reading the drone path logs, the main instance initiates a countdown and communicates the drone's starting position to the other instances. This synchronization ensures that all instances are ready to begin recording simultaneously.

Following the countdown, the main instance issues drone movement commands to all instances. Each command comprises the $x$, $y$, and $z$ coordinates of the drone and the time required for the drone to reach these coordinates. This approach guarantees precise timing and spatial accuracy for the audio recordings.

Each instance records its audio data independently, ensuring that all microphones capture sound accurately and in sync with the drone's movements. The audio files are then saved to a predefined recording folder for subsequent processing and analysis.

# 5. EXAMPLE USE CASES AND PERFORMANCE EVALUATION

To work with in our use cases, we created six different drone routes which can be seen in Figure 5.1 that includes measurement table locations. Each route was recorded from three different table position where each table consists of two cameras and two microphones. Both cameras and microphones are two meters apart from each other. We collected both daytime and nighttime data and also added 10 birds to the environment to make it more difficult to detect drones via image. Recorded images are in 1280x720 resolution. The created dataset takes up approximately 14 GB of storage. Our final scenarios for each route are shown in Table 5.1

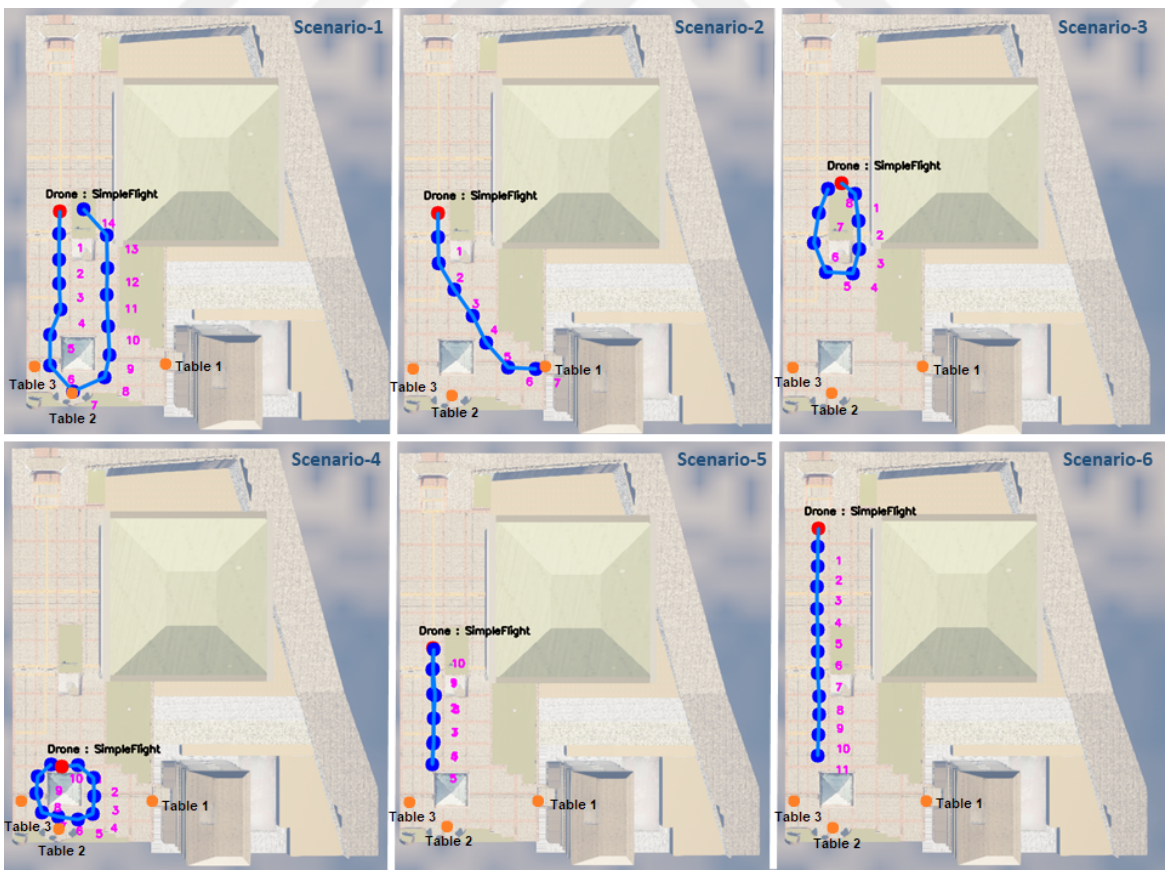


Figure 5.1. Dataset scenarios.

Table 5.1. Scenarios for each route.

| **Position** | **Time** | **Weather** | **Bird Count** |
|---|---|---|---|
| **Table 1** | Day | Clear | 10 |
| **Table 1** | Night | Clear | 10 |
| **Table 2** | Day | Clear | 10 |
| **Table 2** | Night | Clear | 10 |
| **Table 3** | Day | Clear | 10 |
| **Table 3** | Night | Clear | 10 |

We worked on three different use cases: augmenting drone image datasets with simulation data, drone detection via audio data, and drone distance estimation via audio data.

## 5.1. Performance Metrics

In evaluating the performance of object detection models, several key metrics are commonly used. These metrics provide insights into the accuracy and robustness of the model, particularly in distinguishing between true positive detections and false positives.

Precision measures the accuracy of the positive predictions made by the model. It is defined as

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}. \tag{5.1}$$

Recall (also known as sensitivity or true positive rate) measures the model's ability to correctly identify all relevant objects when the true state is "positive". It is defined as

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}. \tag{5.2}$$

Intersection over Union (IoU) is a fundamental metric used to evaluate the accuracy of an object detector on a particular dataset. IoU is defined as the area of overlap between the predicted bounding box and the ground truth bounding box divided by the area of their union. The formula for IoU is defined as

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}. \tag{5.3}$$

Mean Average Precision (mAP) is a comprehensive metric used to evaluate the precision and recall across different thresholds. It considers the model's performance at multiple intersection over union (IoU) thresholds. To evaluate mAP:

(i) Set IoU Thresholds: Multiple IoU thresholds are set, such as 0.5, 0.55, 0.6, ..., 0.95.

(ii) Compute Precision and Recall: For each threshold, precision and recall are computed.

(iii) Average Precision (AP): For each IoU threshold, the precision-recall curve is plotted. The AP is the area under this curve.

(iv) Calculate Mean Average Precision (mAP): The mean of the average precision values calculated at these IoU thresholds is mAP.

mAP50 measures the mean average precision at an IoU threshold of 0.50. This means that a detection is considered correct if the IoU between the predicted bounding box and the ground truth is at least 0.50.

mAP50-95, often simply referred to as mAP, is the average of the mean average precision values calculated at IoU thresholds ranging from 0.50 to 0.95, in increments of 0.05. This provides a more rigorous assessment of the model's performance across varying levels of overlap and is defined as

$$mAP50 - 95 = \frac{1}{11} \sum_{t=0.5}^{0.95} mAPt. \tag{5.4}$$

Average Precision (AP) is calculated by plotting the precision-recall curve and computing the area under this curve. The precision-recall curve is a graphical representation that shows the trade-off between precision and recall for different threshold values.

The Receiver Operating Characteristic (ROC) Curve is a graphical plot used to evaluate the performance of a binary classifier. It illustrates the model's ability to discriminate between positive and negative classes by plotting the true positive rate (recall) against the false positive rate (FPR) at various threshold settings.

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}. \tag{5.5}$$

The Area Under the ROC curve (AUC-ROC) provides a single scalar value to summarize the performance of the model. A model with a perfect performance has an AUC-ROC of 1.0, while a model with random performance has an AUC-ROC of 0.5.

The ROC curve is particularly useful for comparing different models and choosing the best one based on its ability to balance sensitivity and specificity. It helps in understanding the trade-offs between true positive rate and false positive rate, enabling a comprehensive evaluation of the model's performance.

## 5.2. Augmenting Drone Image Datasets with Simulation Data

For the drone detection, we used YOLOv8, the latest iteration of the You Only Look Once (YOLO) models developed by Ultralytics [39]. YOLOv8 continues the tradition of its predecessors, offering state-of-the-art performance in object detection tasks with several enhancements such as speed, accuracy, and ease of use.

Ultralytics provides several different YOLOv8 models, each designed to balance performance and computational efficiency according to the needs of different applications. The available YOLOv8 models and their accuracy on the COCO [17] dataset is shown in Table 5.2. We used YOLOv8n because detection speed was an important

part of our use case and the ability to run on entry-level devices increases usability when we don't have a powerful hardware.

Table 5.2. YOLOv8 object detection model performance comparison. [40]

| **Model** | **Size (Pixels)** | **mAP mAP50-95** | **Speed (ms) (A100 TensorRT)** | **Params (M)** |
|---|---|---|---|---|
| **YOLOv8n** | 640 | 37.3 | 0.99 | 3.2 |
| **YOLOv8s** | 640 | 44.9 | 1.20 | 11.2 |
| **YOLOv8m** | 640 | 50.2 | 1.83 | 25.9 |
| **YOLOv8l** | 640 | 52.9 | 2.39 | 43.7 |
| **YOLOv8x** | 640 | 53.9 | 3.53 | 68.2 |

We utilized the Drone vs Bird dataset, a benchmark dataset specifically designed to address the challenges of distinguishing drones from birds in visual data. This dataset includes 69 video clips that capture various scenarios where drones and birds appear in similar contexts. The primary aim of forming this dataset is to provide a comprehensive collection of annotated visual data that can be used to train and evaluate drone detection and classification algorithms.

From the 69 video clips provided in the Drone vs Bird dataset, we extracted a total of 95 645 images. Among these images, 75 092 of them contains drones, while the remaining images either contains birds or they are background images without any relevant objects. This extraction process is crucial to create a diverse and comprehensive set of images for training and evaluating our models.

To investigate the impact of simulation data on the training process, especially when the amount of real-world data is limited, we created four distinct custom datasets. Data augmentation is achieved by the help of our simulator, which also enables balancing the data where the real data is imbalanced.

These training datasets allowed us to evaluate the performance of the model under different training conditions. The specific configurations of these datasets are detailed in Table 5.3.

Table 5.3. Training data used in models where real data is from Drone vs Bird.

| Dataset | Total | | Drone | | Background | |
|---|---|---|---|---|---|---|
| | **Real** | **Simulation** | **Real** | **Simulation** | **Real** | **Simulation** |
| **DvB-250** | 250 | 0 | 204 | 0 | 46 | 0 |
| **DvB-250 + Sim** | 250 | 3000 | 204 | 1398 | 46 | 1602 |
| **DvB-500** | 500 | 0 | 369 | 0 | 131 | 0 |
| **DvB-500 + Sim** | 500 | 3000 | 369 | 1398 | 131 | 1602 |

We used the same test dataset to evaluate the performance of the same YOLOv8n model that is trained by using different datasets. Test dataset consists of only Drone vs Bird images. Total of 9563 images were used for testing where 7564 of them included drones and 1999 of them were backgrounds.

The experimental setup involved training the same model using the four dataset variations. The model was trained to detect and classify drones in the images, with the performance measured using standard metrics such as precision, recall (a.k.a. true positive rate) and mean average precision (mAP). The training and evaluation processes were conducted using Nvidia GTX 1070 GPU. Test results are shown in Table 5.4

Table 5.4. Test results.

| Training Dataset | Precision | Recall | mAP50 | mAP50-95 |
|---|---|---|---|---|
| **DvB-250** | 0.707 | 0.594 | 0.637 | 0.294 |
| **DvB-250 + Sim** | 0.75 | 0.645 | 0.688 | 0.326 |
| **DvB-500** | 0.843 | 0.675 | 0.732 | 0.366 |
| **DvB-500 + Sim** | 0.859 | 0.685 | 0.752 | 0.393 |

Comparison of ROC curves for different IoU threshold between models are shown in Figure 5.2 and Figure 5.3

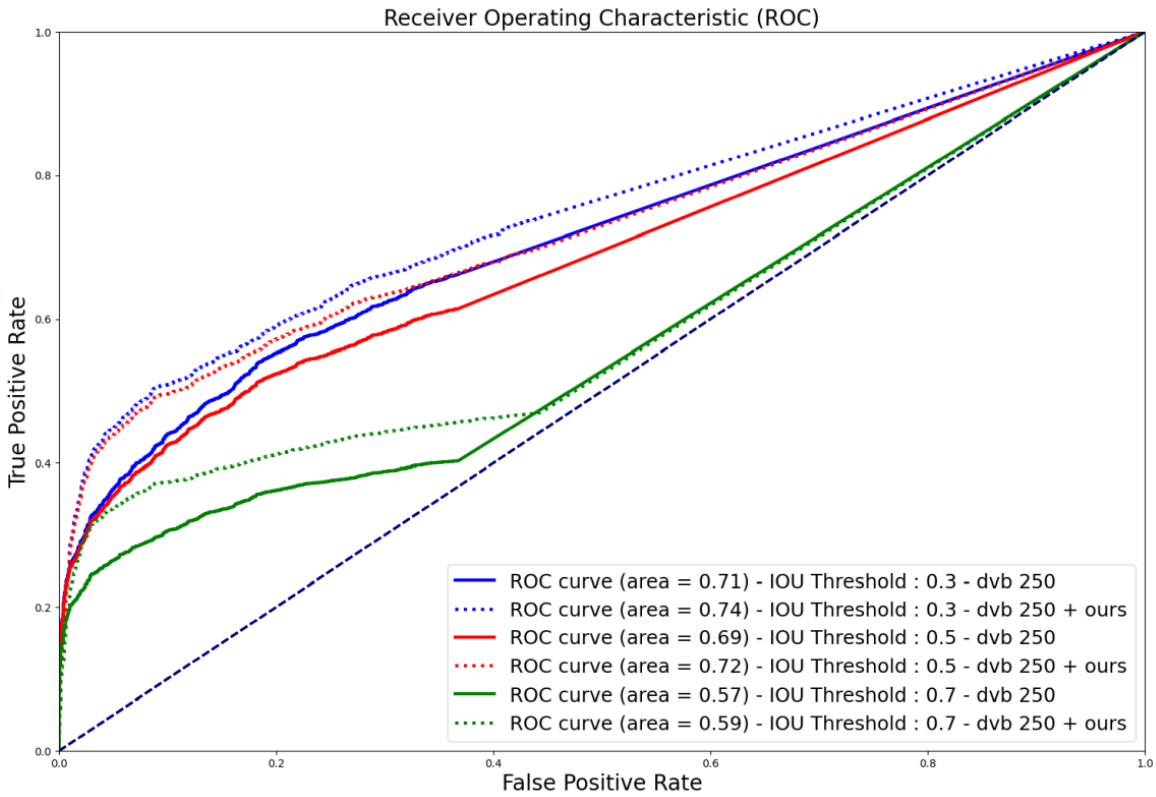


Figure 5.2. ROC curve change of models for 250 Drone vs Bird Data.

In Figure 5.2, The plot contains several ROC curves, each representing different IOU thresholds (0.3, 0.5, 0.7) and datasets (DvB 250 alone versus DvB 250 combined with our simulation data). The combination of our simulation data with DvB 250 improves the model's performance across all IOU thresholds. The improvements are more pronounced at higher IOU thresholds, suggesting that the simulation data helps the model distinguish between drones and background more effectively.

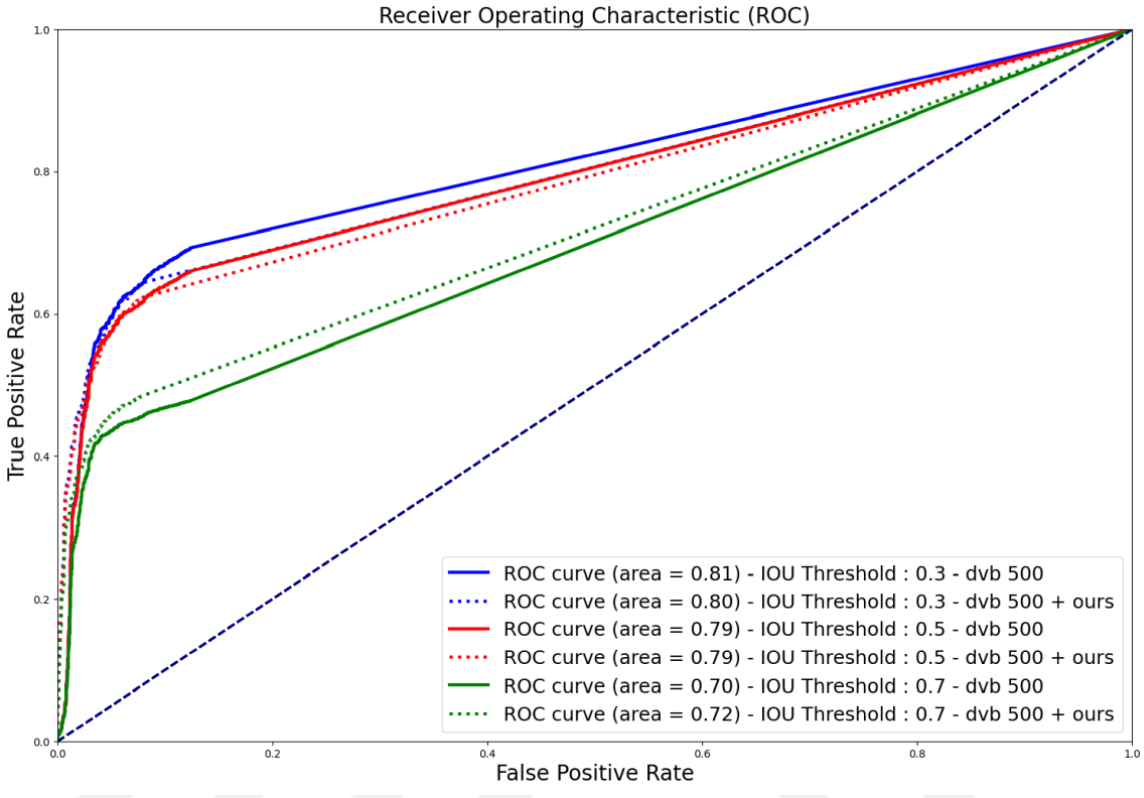


Figure 5.3. ROC curve change of models for 500 Drone vs Bird Data.

In Figure 5.3, The plot contains several ROC curves, each representing different IOU thresholds (0.3, 0.5, 0.7) and datasets (DvB 500 alone versus DvB 500 combined with our simulation data). While the addition of our simulation data slightly improves the model's performance for IoU threshold of 0.7, as we decrease IoU it loses its effect.

Overall, incorporating our simulation data generally improves the model's performance, especially when dealing with lower amounts of real-world data (DvB 250). As we can observe in Tables 5.5 and 5.6, higher IoU thresholds tend to show better performance improvement, indicating that our simulation data is more effective when a more severe matching criterion is applied. Since higher AUC values indicate better performance, values suggest that our models are capable of distinguishing between drones and other objects or background with reasonable accuracy.

Table 5.5. Percentage change of TPR with respect to our data (DvB-250 + Sim) on different FPR values.

| | **IOU 0.3** | **IOU 0.5** | **IOU 0.7** |
|---|---|---|---|
| **FPR 0.1** | +15.94% | +16.61% | +22.4% |
| **FPR 0.2** | +6.87% | +9.08% | +13.65% |
| **FPR 0.3** | +6.55% | +8.91% | +14.04% |

Table 5.6. Percentage change of TPR with respect to our data (DvB-500 + Sim) on different FPR values.

| | **IOU 0.3** | **IOU 0.5** | **IOU 0.7** |
|---|---|---|---|
| **FPR 0.1** | -2.76% | -1.33% | +6.1% |
| **FPR 0.2** | -2.86% | -2.44% | +5.53% |
| **FPR 0.3** | -3.33% | -2.16% | +4.46% |

## 5.3. Audio Based Drone Detection

Audio-based drone detection is necessary to ensure the security of restricted airspaces when the visibility is hurdled. By analyzing the unique acoustic signatures produced by drones, classification models can determine the presence of drones even when they are not visible to the naked eye or cameras. Utilizing the audio modality is particularly useful in situations where visual obstruction or low visibility conditions exist.

For this use case, we created an audio dataset using data collected from our simulation scenarios in digital twin of the environment. Specifically, we used audio recordings from the first five routes for training, resulting in 30 audio clips, and route six for testing. Each audio recording is divided into one-second clips to increase the

granularity of the data. To simulate a more realistic environment, we added random background noise from the UrbanSound8k dataset [41]. This dataset contains 8,732 labeled sound excerpts (less than 4 seconds) from ten classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gunshot, jackhammer, siren, and street music.

Given that our simulation scenarios used two microphones, we added the same background audio to both microphones but varied the decibel levels to introduce diversity in the recordings. This approach helps the model learn to differentiate between drone sounds and background noise more effectively.

A mel spectrogram is a powerful visual representation for audio analysis. It represents the short-term power spectrum of a sound, with the frequency axis transformed to a mel scale, which approximates the human ear's response more closely than the linear frequency scale. By converting audio signals into mel spectrograms, we can capture the essential features of drone sounds, making it easier for machine learning models to detect the existence of drones.

For each one-second audio clip, we generated a mel spectrogram and labeled it as either drone or background. We created an equal amount of data using only background audio, ensuring a balanced dataset for training, which can be easily ensured with our simulation.

We used the YOLOv8n-cls classification model from Ultralytics, a state-of-the-art model designed for efficient and accurate classification tasks. The available YOLOv8 classification models and their accuracy on the ImageNet [14] dataset is shown in Table 5.7

Our training dataset consisted of 636 mel spectrograms labeled as drone and 636 labeled as background, all derived from the first five routes. For validation, we used 70 mel spectrograms of drones and 70 of background from the same routes.

Table 5.7. YOLOv8 classification model performance comparison [42].

| Model | Size (Pixels) | acc top1 | acc top5 | Speed (ms) (A100 TensorRT) | Params (M) |
|---|---|---|---|---|---|
| **YOLOv8n-cls** | 224 | 69.0 | 88.3 | 0.31 | 2.7 |
| **YOLOv8s-cls** | 224 | 73.8 | 91.7 | 0.35 | 6.4 |
| **YOLOv8m-cls** | 224 | 76.8 | 93.5 | 0.62 | 17.0 |
| **YOLOv8l-cls** | 224 | 76.8 | 93.5 | 0.87 | 37.5 |
| **YOLOv8x-cls** | 224 | 79.0 | 94.6 | 1.01 | 57.4 |

For testing, we used 174 mel spectrograms labeled as drone and 174 labeled as background from route six. Example mel histogram of a one drone audio clip with background and a background without drone can be seen in Figure 5.4

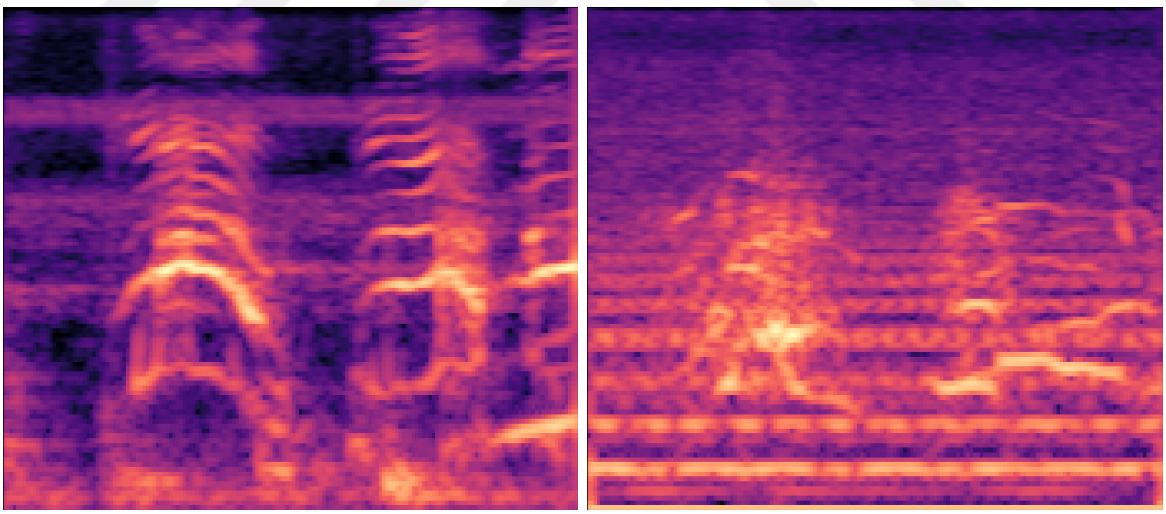

Figure 5.4. Mel spectrograms used in training. Left: Random background audio. Right: Drone and random background audio.

The model was tested on audio data from the sixth route using the left microphone, right microphone, and a combination of both microphones. We conducted two sets of tests: one with the background noise in the right microphone 6dB less than the left, and another with the background noise in the right microphone 2dB less than

the left. On both tests, background noise on left microphone is equal. The results in Tables 5.8, 5.9, 5.10 demonstrate that our audio-based drone detection model performs reasonably well in different noise conditions. When the background noise was 6dB less noise-hurdled in the right microphone, the combined accuracy measured from table 3 was the highest at 89.66%. In contrast, with a 2dB less noise-hurdled right microphone, the combined accuracy was slightly lower at 87.93%.

Table 5.8. Confusion matrices of drone detection from measurement table 1 using a single microphone and two microphones.

| | | **-6dB (Right Mic)** | | **-2dB (Right Mic)** | |
|---|---|---|---|---|---|
| **Using Left Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 13 | 1 | 13 | 1 |
| | **Negatives** | 16 | 28 | 16 | 28 |
| **Using Right Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 14 | 0 | 13 | 0 |
| | **Negatives** | 15 | 29 | 16 | 29 |
| **Using Both Microphones** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 14 | 0 | 13 | 0 |
| | **Negatives** | 15 | 29 | 16 | 29 |

The performance difference between using individual microphones and combining them suggests that leveraging multiple audio sources can improve detection accuracy. However, the noise level disparity between microphones also plays a role in the model's performance.

Overall, the use of mel spectrograms and the Yolov8n-cls model proved effective in distinguishing drone sounds from background noise, showcasing the potential for reliable audio-based drone detection in various environmental conditions.

Table 5.9. Confusion matrices of drone detection from measurement table 2 using a single microphone and two microphones.

| | | **-6dB (Right Mic)** | | **-2dB (Right Mic)** | |
|---|---|---|---|---|---|
| **Using Left Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 13 | 0 | 13 | 0 |
| | **Negatives** | 16 | 29 | 16 | 29 |
| **Using Right Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 17 | 1 | 16 | 1 |
| | **Negatives** | 12 | 28 | 13 | 28 |
| **Using Both Microphones** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 17 | 0 | 16 | 0 |
| | **Negatives** | 12 | 29 | 13 | 29 |

Table 5.10. Confusion matrices of drone detection from measurement table 3 using a single microphone and two microphones.

| | | **-6dB (Right Mic)** | | **-2dB (Right Mic)** | |
|---|---|---|---|---|---|
| **Using Left Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 17 | 1 | 17 | 1 |
| | **Negatives** | 12 | 28 | 12 | 28 |
| **Using Right Microphone** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 22 | 0 | 21 | 0 |
| | **Negatives** | 7 | 29 | 8 | 29 |
| **Using Both Microphones** | | **Actual** | | **Actual** | |
| | | **Positives** | **Negatives** | **Positives** | **Negatives** |
| **Predicted** | **Positives** | 23 | 0 | 22 | 0 |
| | **Negatives** | 6 | 29 | 7 | 29 |

## 5.4. Audio Based Drone Distance Estimation

In this use case, we explore the methodology and results of our audio-based drone distance estimation using data generated from our simulation environment. Our goal is to estimate the distance of a drone from a fixed point using audio recordings from left and right microphones. We used first five routes for training and the sixth route for testing.

To create a dataset which we can feed into a neural network, the audio recordings from our simulation were processed into 1-second clips. Each clip was then analyzed using Mel-Frequency Cepstral Coefficients (MFCCs), a common feature extraction technique in audio processing. MFCCs are useful for this task because they represent the short-term power spectrum of a sound, capturing essential characteristics of the audio signal that are relevant for distinguishing different sources and estimating distances. By converting the time-domain signal into a frequency-domain representation, MFCCs provide a robust set of features for machine learning models to learn from.

MFCCs have the ability to capture the spectral properties of the audio signal. The process involves dividing the audio signal into overlapping audio frames and using a window function on each frame. The Fourier transform is then used to convert each frame into the frequency domain. The resulting power spectrum is filtered through a set of triangular band-pass filters spaced according to the mel scale, which approximates the human ear's response to different frequencies. Finally, the logarithm of the filter bank energies is computed, and a discrete cosine transform (DCT) is applied to obtain the MFCCs.

For our specific application, we set the number of MFCCs to 13. Since the sampling rate for the simulation audio is 48000Hz, this resulted in a feature matrix of size 94x13 for each 1-second audio clip and these 94 MFCCs can be calculated as

$$nMFCC = \frac{audiolength * sampling}{window} \sim \frac{audiolength * sampling}{hop} = \frac{1 * 48000}{512}, \quad (5.6)$$

where the unit of *audiolength* is seconds. To compute MFCC, Fast Fourier Transform (FFT) is used and its window length is 2048 and number of samples between successive frames is 512.

To improve the robustness of our distance estimation model, we combined the features from the left and right microphones. This concatenation led to a final input feature size of 94x26, integrating spatial audio information from both microphones.

To generate the labels for our training and testing datasets, we utilized the drone position logs from our simulation environment. For each 1-second audio clip, we calculated the average position of the drone. Using this average position, we computed the distance from the center point between the two microphones to the drone's position. This distance served as the ground truth value for our model.

We created a CNN model shown in Figure 5.5 and trained it for 1000 epochs using the dataset created from the first 5 route.

Figures 5.6, 5.7 and 5.8 represents the evaluation of the model's performance based on 1-second intervals from a 29-second audio clip. The graphs show how the model's performance varies over time. The left side of each graph represents the start of the audio clip, and the right side represents the end.

Figure 5.6 shows the percentage error of the drone distance estimation. The color bar at the bottom indicates the range of percentage error values, with green representing lower errors and red representing higher errors. Gray areas indicate the minimum and maximum 10% values for the percentage error.

In Figure 5.6, measurement table 1 shows dominantly green areas, indicating generally low percentage errors across most intervals. There are no yellow and red areas, suggesting that high errors are rare. Measurement table 2 shows a mix of green, yellow, and red patches, indicating more variability in percentage errors.

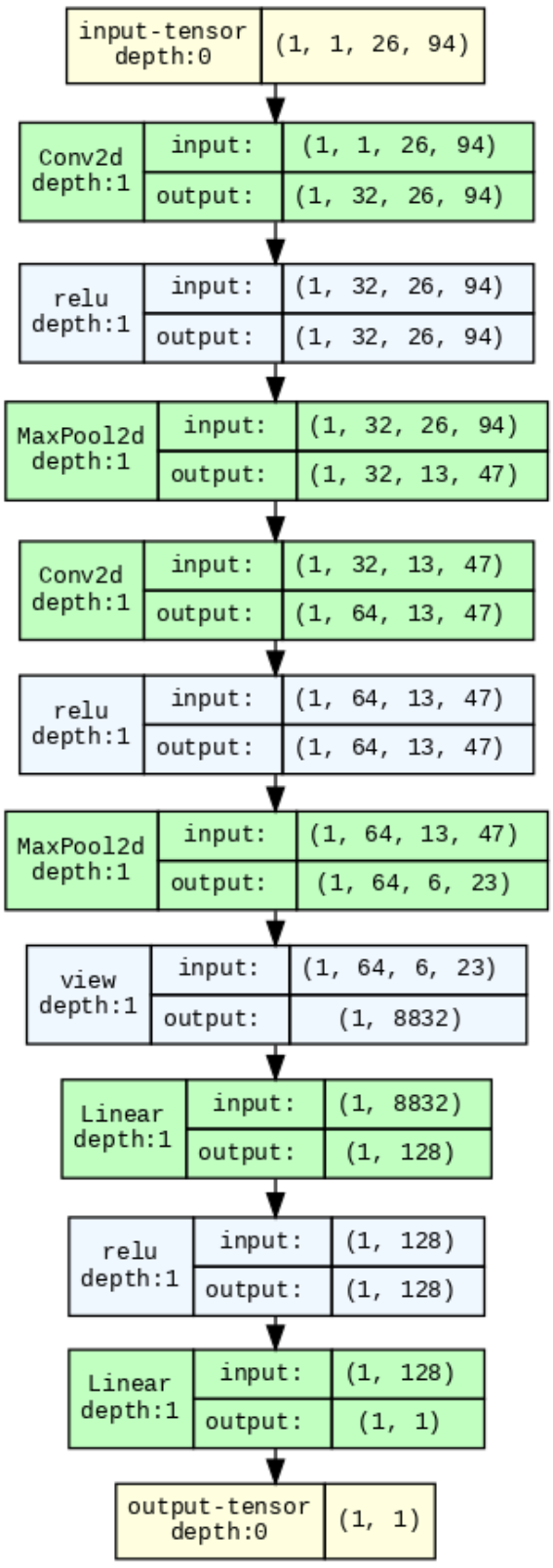


Figure 5.5. Distance estimation CNN model.

Table 5.11. Drone distance estimation statistics for the absolute difference and percentage error between ground truth and estimations. For statistics, we trimmed rows that in the min 10% and max 10% percentage error.

| **Measurement Table 1 (RMSE = 2.18)** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Absolute Difference (m)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 1.79 | 1.26 | 0.08 | 1.04 | 1.46 | 2.48 | 5.09 |
| **Percentage Error (%)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 3.10 | 1.78 | 0.19 | 1.99 | 3.26 | 3.86 | 7.79 |
| **Measurement Table 2 (RMSE = 2.83)** | | | | | | | |
| **Absolute Difference (m)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 2.04 | 1.36 | 0.21 | 0.90 | 1.84 | 2.79 | 5.17 |
| **Percentage Error (%)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 6.27 | 5.45 | 0.36 | 1.90 | 4.87 | 8.59 | 19.87 |
| **Measurement Table 3 (RMSE = 2.88)** | | | | | | | |
| **Absolute Difference (m)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 1.48 | 1.37 | 0.05 | 0.46 | 0.90 | 2.18 | 4.97 |
| **Percentage Error (%)** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| | 3.99 | 3.81 | 0.44 | 1.11 | 3.11 | 5.33 | 16.06 |

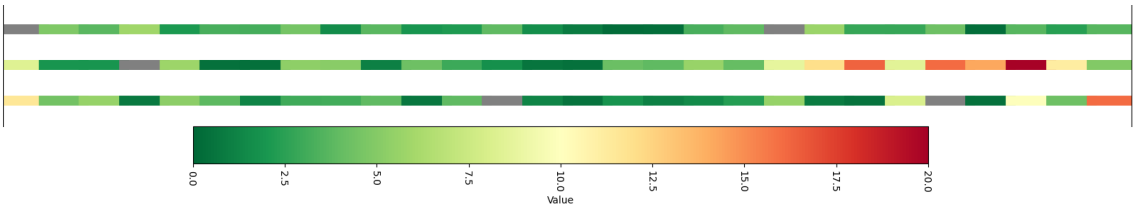


Figure 5.6. Percentage Error based on 1 second intervals for measurement tables. Min and Max 10% shown in gray. Top : Table 1, Middle : Table 2, Bottom : Table 3.

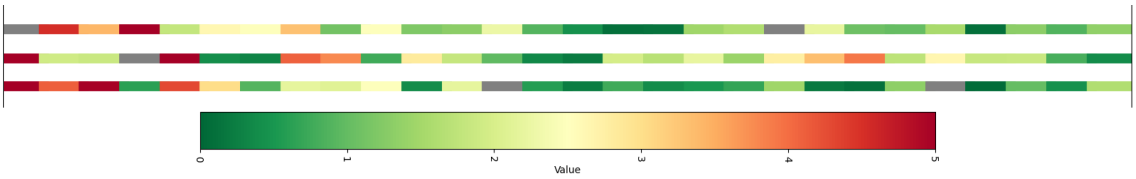


Figure 5.7. Absolute Difference based on 1 second intervals for measurement tables. Min and Max 10% shown in gray. Top : Table 1, Middle : Table 2, Bottom : Table 3.

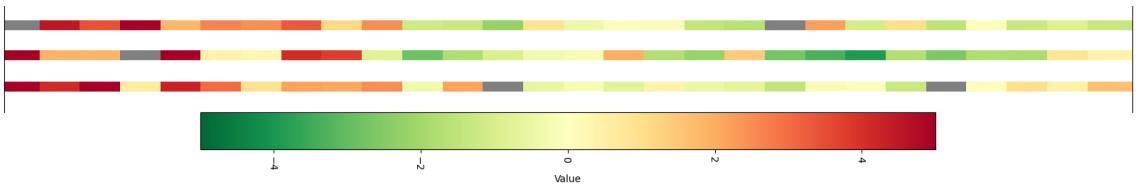


Figure 5.8. Difference based on 1 second intervals for measurement tables. Min and Max 10% shown in gray. Top : Table 1, Middle : Table 2, Bottom : Table 3.

There are more frequent yellow and red areas towards the end compared to measurement table 1, suggesting periods with higher errors. Measurement table 3 is mostly green with few yellow and red areas. The errors are generally low, with high errors being infrequent. Overall, measurement table 1 exhibits the best performance with the lowest and most stable percentage errors. Measurement table 2's performance is less consistent than measurement table 1's, indicating a higher degree of variability in the model's accuracy. Measurement table 3 also exhibits good performance with relatively low percentage errors, comparable to measurement table 1.

Figure 5.7 displays the absolute difference between the predicted and actual drone distances. The color bar at the bottom indicates the range of absolute difference values, with green representing smaller differences and red representing larger differences. Gray areas indicate the minimum and maximum 10% values for the absolute difference. In Figure 5.7, measurement tables 1 and 3 show similar performance with predominantly small absolute differences, indicating more stable and better model accuracy. Measurement table 2 has larger and more variable absolute differences, indicating less accurate performance compared to measurement tables 1 and 3.

Figure 5.8 represents the difference between the predicted and actual drone distances. The color bar at the bottom indicates the range of difference values, with green meaning model estimated drones location to be farther than the actual and red representing the opposite. Yellow represents perfect estimation. In Figure 5.8, at the start all three measurement tables exhibit noticeable red patches, indicating all measurement tables estimated drone to be closer than it actually is. This suggests that the model initially struggles to accurately predict the drone's distance when it is farther away. As the drone approaches, the performance improves, as evidenced by the increasing prevalence of yellow areas.

Measurement table 3 maintains the highest accuracy and stability throughout, especially as the drone gets closer. Measurement table 1, while initially less accurate, demonstrates substantial stability over time. Measurement table 2, exhibits the most variability and larger differences, indicating less reliable performance compared to the other two tables. Towards the end, green patch shows that it started to guess drones position to be farther than it actually was. The trend across all tables highlights the model's increased accuracy as the drone approaches the recording point.

According to the results shown in Table 5.11, table 1 shows a relatively low mean absolute difference (1.79 m) with a standard deviation of 1.26 m, indicating consistent performance. The maximum difference is 5.09 m. The mean percentage error is the lowest among the tables at 3.10%, with a maximum of 7.79%. This suggests that table 1 provides relatively accurate estimations.

Table 2 has the highest mean absolute difference (2.04 m) and standard deviation (1.36m), with a maximum of 5.17 m. This indicates greater variability in estimation accuracy. The mean percentage error is significantly higher at 6.27%, with a maximum of 19.87%. This suggests that Table 2's estimations are less accurate and more variable.

Table 3 has the lowest mean absolute difference (1.48m) and a standard deviation of 1.37m, with a maximum of 4.97m. This indicates relatively consistent and accurate

estimations. The mean percentage error is moderate at 3.99%, with a maximum of 16.06%. While the errors are higher than table 1, they are lower and more consistent than table 2.

These results suggest that the environmental conditions and spatial configuration around table 1 are most favorable for accurate drone distance estimations, while table 2 poses more challenges for the model.

# 6. CONTRIBUTIONS AND FUTURE WORK

## 6.1. Summary of Contributions

The key contributions of this thesis are as follows:

- To address the challenges and costs associated with developing drone detection systems for research, we created Airsim GUI, which integrates with Unreal Engine and AirSim. This tool allows researchers to build drone detection systems in a simulated environment and generate both image and audio datasets.
- We demonstrated that drone simulation image data from our system can enhance model performance, particularly when real-world data is limited.
- Using our proposed dataset generation system, we produced an image and audio dataset. This dataset was utilized to augment existing drone image datasets, detect drone through audio data, and estimate drone distance using audio data.

## 6.2. Future Work

There are more complex scenarios, different drone types, and various environmental conditions to enhance the dataset's diversity.

We can use other camera types such as depth camera and infrared camera to create datasets and compare their performance to a real datasets. We plan to include weather scenarios to the system such as fog and snow. We also plan to include different drone types.

# REFERENCES


1. Bishop, C. M. and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*, Vol. 4, Springer, New York, 2006.

2. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

3. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018.

4. McCulloch, W. S. and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *The Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, 1943.

5. Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR, Hamilton, ON, 1998.

6. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning Representations by Back-Propagating Errors", *nature*, Vol. 323, No. 6088, pp. 533–536, 1986.

7. Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet Classification With Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems*, Vol. 25, 2012.

8. LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, Vol. 1, No. 4, pp. 541–551, 1989.

9. LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

10. Simonyan, K. and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition", *arXiv preprint arXiv:1409.1556*, 2014.

11. He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, 2016.

12. Redmon, J., S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, Las Vegas, NV, USA, 2016.

13. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper With Convolutions", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, Boston, MA, USA, 2015.

14. Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database", *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, Miami, FL, USA, 2009.

15. Jiang, N., K. Wang, X. Peng, X. Yu, Q. Wang, J. Xing, G. Li, Q. Ye, J. Jiao and Z. Han, "Anti-UAV: A Large-scale Benchmark for Vision-based UAV Tracking", *IEEE Transactions on Multimedia*, Vol. 25, pp. 486–500, 2021.

16. Pawełczyk, M. and M. Wojtyra, "Real world object detection dataset for quadcopter unmanned aerial vehicle detection", *IEEE Access*, Vol. 8, pp. 174394–174409, 2020.

17. Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft COCO: Common Objects in Context", *European Conference on Computer Vision*, pp. 740–755, Springer, Zurich, Switzerland, 2014.

18. Everingham, M., S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn and A. Zis-

serman, "The Pascal Visual Object Classes Challenge: A Retrospective", *International Journal of Computer Vision*, Vol. 111, pp. 98–136, 2015.

19. Al-Emadi, S., A. Al-Ali, A. Mohammad and A. Al-Ali, "Audio Based Drone Detection and Identification Using Deep Learning", *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 459–464, IEEE, Tangier, Morocco, 2019.

20. Al-Emadi, S., "Saraalemadi/droneaudiodataset", 2018, `https://github.com/saraalemadi/DroneAudioDataset`, accessed on 11 July 2024.

21. Piczak, K. J., "ESC: Dataset For Environmental Sound Classification", *Proceedings of the 23rd ACM International Conference on Multimedia*, pp. 1015–1018, Brisbane, Australia, 2015.

22. Warden, P., "Speech Commands: A Dataset For Limited-Vocabulary Speech Recognition", *arXiv preprint arXiv:1804.03209*, 2018.

23. Aydin, B. and S. Singha, "Drone Detection Using YOLOv5", *Eng*, Vol. 4, No. 1, pp. 416–433, 2023.

24. Zhang, Y., Z. Guo, J. Wu, Y. Tian, H. Tang and X. Guo, "Real-Time Vehicle Detection Based on Improved YOLOv5", *Sustainability*, Vol. 14, No. 19, p. 12274, 2022.

25. Seidaliyeva, U., D. Akhmetov, L. Ilipbayeva and E. T. Matson, "Real-Time and Accurate Drone Detection in a Video with a Static Background", *Sensors*, Vol. 20, No. 14, p. 3856, 2020.

26. Zhai, X., Z. Huang, T. Li, H. Liu and S. Wang, "YOLO-Drone: An Optimized YOLOv8 Network for Tiny UAV Object Detection", *Electronics*, Vol. 12, No. 17, p. 3664, 2023.

27. Bernardini, A., F. Mangiatordi, E. Pallotti and L. Capodiferro, "Drone Detection by Acoustic Signature Identification", *Electronic Imaging*, Vol. 29, pp. 60–64, 2017.

28. Mezei, J. and A. Molnár, "Drone Sound Detection By Correlation", *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 509–518, IEEE, 2016.

29. Srigrarom, S., N. J. L. Sie, H. Cheng, K. H. Chew, M. Lee and P. Ratsamee, "Multi-Camera Multi-Drone Detection, Tracking and Localization With Trajectory-Based Re-Identification", *2021 Second International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, pp. 1–6, IEEE, 2021.

30. Mezei, J., V. Fiaska and A. Molnár, "Drone Sound Detection", *2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 333–338, IEEE, 2015.

31. Svanström, F., C. Englund and F. Alonso-Fernandez, "Real-Time Drone Detection and Tracking With Visible, Thermal and Acoustic Sensors", *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 7265–7272, IEEE, Milano, Italy, 2021.

32. Dieter, T. R., A. Weinmann, S. Jäger and E. Brucherseifer, "Quantifying the Simulation–Reality Gap for Deep Learning-Based Drone Detection", *Electronics*, Vol. 12, No. 10, p. 2197, 2023.

33. Games, E., "Unreal Engine", `https://www.unrealengine.com/en-US/`, accessed on 11 July 2024.

34. Shah, S., D. Dey, C. Lovett and A. Kapoor, "Airsim: High-fidelity Visual and Physical Simulation For Autonomous Vehicles", *Field and Service Robotics: Results of the 11th International Conference*, pp. 621–635, Springer, Zurich, Switzerland, 2018.

35. Carrio, A., J. Tordesillas, S. Vemprala, S. Saripalli, P. Campoy and J. P. How, "Onboard Detection and Localization of Drones Using Depth Maps", *IEEE Access*, Vol. 8, pp. 30480–30490, 2020.

36. Microsoft, "AirSim", `https://github.com/microsoft/AirSim`, accessed on 11 July 2024.

37. Computing, R., "PyQt", `https://www.riverbankcomputing.com/software/`, accessed on 11 July 2024.

38. Google, "Google Street View of Boğaziçi University", `https://maps.app.goo.gl/1b9im3iLQAuC1py49`, accessed on 11 July 2024.

39. Jocher, G., A. Chaurasia and J. Qiu, "Ultralytics YOLOv8", 2023, `https://github.com/ultralytics/ultralytics`, accessed on January 06, 2024.

40. Ultralytics, "Object Detection", 2023, `https://docs.ultralytics.com/tasks/detect/`, accessed on January 06, 2024.

41. Salamon, J., C. Jacoby and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research", *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 1041–1044, Orlando, FL, USA, 2014.

42. Ultralytics, "Image Classification", 2023, `https://docs.ultralytics.com/tasks/classify/`, accessed on January 06, 2024.