Obstacle Detection for Drone Flight Path

A Project Report
Presented to
The Faculty of the Department of Applied Data Science
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Data Analytics
By

Pagrut Tanvi

Sharma Vansh

Sodha Akshay

Somaiya Harsh

May 9, 2025

Copyright © 2025

Pagrut Tanvi, Sharma Vansh, Sodha Akshay, Somaiya Harsh

ALL RIGHTS RESERVED

APPROVED FOR DEPARTMENT OF APPLIED DATA SCIENCE

_____

Dr. Simon Shim, Project Advisor

_____

Dr. Lee C. Chang, Project Advisor

ABSTRACT

The project focuses on enhancing obstacle detection for the navigation of drones by developing sophisticated models. The project will design a system that will allow a drone to safely avoid obstacles. This project will apply YOLOv8, Faster R-CNN using Detectron2, RetinaNet, and RT-DETR to find which model performs better in detecting obstacles in drone flight paths under variable environments. YOLOv11 gave a precision of 92% and gave the best results among the four models for detecting objects. The novelty of this research is understanding how each model enhances obstacle detection and improves drone safety in dynamic situations.

To achieve this goal, the project will begin by collecting drone flight data across urban areas, such as buildings, trees, and ultra-thin structures such as power lines. Data pre-processing entails cleaning and normalizing the data to ensure consistency. Detailed annotations will be carried out to label the various categories of obstacles. This data will be used to train four models to perform obstacle detection. These models were evaluated using mAP, precision, and recall.

This project will develop advanced obstacle detection models for drone flight paths in diverse environments. The system will be improved to handle the identification of obstacles for safer and more efficient navigation in various climatic conditions. These results will improve drone operations across diverse industries by offering robust solutions for handling dynamic scenarios having huge categories of objects to be classified.

This project will have major applications in logistics, especially where drones will fly autonomously, taking routes for delivery and maintaining the capability for obstruction avoidance. Such a system can be used to perform surveillance by monitoring areas in urban and rural environments by helping in the navigation of drones around structures such as buildings and trees with no interference.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

1. **Introduction** **8**
   - 1.1 Project Background and Executive Summary 8
   - 1.2 Project Requirements 13
   - 1.3 Project Deliverables 15
   - 1.4 Technology and Solution Survey 20
   - 1.5 Literature Survey of Existing Research 33
2. **Data and Project Management Plan** **40**
   - 2.1 Data Management Plan 40
   - 2.2 Project Development Methodology 43
   - 2.3 Project Organization Plan 46
   - 2.4 Project Resource Requirements and Plan 46
   - 2.5 Project Schedule 50
3. **Data Engineering** **55**
   - 3.1 Data Process 55
   - 3.2 Data Collection 59
   - 3.3 Data Pre-processing 74
   - 3.4 Data Transformation 86
   - 3.5 Data Preparation 97
   - 3.6 Data Statistics 104
4. **Model Development** **114**
   - 4.1 Model Proposals 114
   - 4.2 Model Supports 130
   - 4.3 Model Comparison and Justification 135
   - 4.4 Model Evaluation Methods 139
5. **Data Analytics System** **143**
   - 5.1 System Requirements Analysis 143
   - 5.2 System Design 151
   - 5.3 Intelligent Solution 158
6. **System Evaluation and Visualization** **167**
   - 6.1 Analysis of Model Execution and Evaluation Results 167
   - 6.2 Achievements and Constraints 177
   - 6.3 Quality Evaluation of Model Functions and Performance 190
   - 6.4 Evaluation of Models vs. Requirements 207
   - 6.5 Project Information Visualization 213
7. **Conclusion** **224**
   - 7.1 Summary 224
   - 7.2 Benefits and Shortcoming 224
   - 7.3 Potential System and Model Applications 225
   - 7.4 Experience and Lessons Learned 226
   - 7.5 Recommendations for Future Work 227
   - 7.6 Contributions and Impacts on Society 227

References **229**
Appendices **257**
Appendix A – System Testing **257**
Appendix B – Project Data Source and Management Store **262**
Appendix C – Project Program Source Library, Presentation, and Demonstration **262**

Introduction

Project Background and Executive Summary

Project Background

Drones are now essential in many industries, including surveillance, infrastructure inspection, logistics, and agriculture. They have, in fact, significantly improved operational efficiency in these areas due to their speed and capacity for autonomous tasks over wide areas. Obstacle environment avoidance, however, is one of the major problems drones encounter. These could be dynamic things like cars, animals, or even other drones, or they could be static things like trees or buildings. Drones can easily collide if there is no efficient obstacle-detection system in place. This can result in operational failures, drone damage, and occasionally human injury.

When flying in a complex, cluttered environment, like an urban landscape, dense forest, or industrial site, obstacle detection becomes especially challenging. Generally speaking, conventional obstacle detection systems are unable to accurately identify and classify obstacles, particularly when flying quickly in the presence of small obstacles like power lines or other drones. This restriction is made worse by the fact that the majority of drones must function independently, making the ability to process various weather conditions a necessary feature.

In light of these difficulties, this work aims to develop a reliable obstacle-detection system that uses cutting-edge models for drone-based systems. To enable the safe operation of drones without the need for human intervention, the project will use computer vision to develop a variety of faster and more accurate obstacle detection techniques. The system will be able to detect, classify, and track obstacles in a variety of environments by implementing models like YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. This will allow unmanned aerial vehicles to alter their flight paths on their own.

Needs and Importance

Drone deployments in increasingly complex environments, such as urban logistics, agriculture, surveillance, and disaster relief, will inevitably increase the need for obstacle detection. There are a lot of obstacles to avoid when working autonomously, ranging from big, static objects like buildings and trees to tiny, dynamic ones like power lines and cars. The majority of modern systems have a common flaw that makes them potentially dangerous to use, particularly in cluttered or rapidly changing environments. Due to hardware constraints like onboard processing power and battery life, this entails allowing drones to process high-resolution video feeds, identify obstacles, accurately classify them, and make fast navigational adjustments while retaining a low computational load. The development of a robust AI-powered system must address the three requirements of scalability, efficiency, and security. It truly unlocks their potential for widespread and dependable application by creating opportunities for autonomous drone use in much more complex and high-stakes scenarios in addition to reducing collisions.

Targeted Project Problem

The primary issue addressed in this work is how to fly drones safely and effectively over both static and dynamic obstacles. Indeed, the majority of traditional obstacle detection systems encounter difficulties when objects are small, have little contrast, or are even moving quickly. Power lines are a common example of these challenges; they are thin and typically have low contrast when flying over cities, making them challenging for the system to detect. In addition, tree limbs and other forest foliage might impair eyesight, which could result in an incoming collision.

Complexity is increased by dynamic impediments like moving cars, animals, or even other drones. To prevent potential accidents, drones must be able to both detect and predict the movement path of obstacles. Even though most systems now perform to some extent, they

are typically far from adequate, particularly under various circumstances. In situations where decisions must be taken in a matter of seconds, even a small delay in obstacle recognition could spell disaster.

With the help of these machine learning models, the project offers a system that can quickly and accurately identify collisions. As a result, the project will be able to provide a comprehensive solution that incorporates many obstacle kinds and scenarios employing models such as Faster R-CNN for high-precision detection in a crowded environment and YOLOv11 for obstacle identification. In the end, this would significantly lower the chance of a collision, making drone operations much safer and more seamless.

Motivation and Goal

The growing needs of many autonomous drone applications in the surveillance, logistics, and agricultural sectors for efficiency and safety are what drove this initiative. Drones' capacity to autonomously recognize and avoid obstacles is becoming increasingly important since they are being flown in increasingly complicated surroundings, such as between buildings, electrical lines, and streetlights in urban settings or around trees, fences, and other obstacles in rural ones. Without sufficient precautions to successfully avoid obstacles, poor obstacle detection could cause a drone mission to fail; this could require expensive repairs or replacement. More significantly, autonomous navigation's dependability determines whether a mission is successful or poses an operational danger in high-risk settings like disaster assistance and infrastructure inspection. Therefore, by increasing obstacle detection accuracy, this study aims to produce improved safety, efficiency, and diversity of drone operations in several industries.

The primary goal of this project is to increase the accuracy of obstacle detection in a variety of environmental situations by utilizing cutting-edge deep learning models that can identify small or rapidly moving low-contrast objects, such as tree branches or power wires.

The likelihood of crashes during flight will be greatly decreased by this increase in precision. Enabling processing so that drones can recognize and react to barriers in dynamic situations, like densely populated cities or forests, is another crucial objective. Finally, it should be adaptable and scalable, able to be used in a variety of contexts without requiring a lot of retraining, from crowded urban areas to rural areas with sparse populations. Because of its increased flexibility, the system can be applied to a wide range of fields with only slight adjustments.

Planned Project Approaches and Method

The project will first conduct a thorough data gathering and preparation phase in order to address the problem of obstacle identification in drone flight pathways. The Drone Depth and Obstacle Segmentation dataset, or DDOS, will be the primary source of data. It includes extensive drone flight data over a variety of environmental scenarios, including parks, urban neighborhoods, and rural locations. This dataset is ideal for training models because it includes a variety of barriers in various environmental settings. To guarantee correct labeling for important obstacle categories, such as thin structural barriers like cables and wires and large-size obstacles like buildings, trees, and meshes, the project will involve extensive annotation and augmentation. It would increase the model's capacity to generalize across different situations and contribute to the development of a more varied training basis.

Going with the work, the succeeding section elaborates on how this development and training of a model using machine learning or deep learning models devised for obstacle detection will be accomplished; the next step will be real implementation and testing of these models in the detection and classification of obstacles. High-speed inference, which is needed in obstacle detection for drone navigation through dynamic and unpredictable environments, will be done using YOLOv11. This is a model of choice due to its high speeds. Detectron2 shall form a very important block in giving highly accurate obstacle detection and

segmentation, especially where the scenes involve overlapping objects or intricately challenging environments. To offer a balance between real-time performance and accuracy, the proposed system will implement RT-DETR v3 to robustly detect multiple obstacles with adaptability. Finally, Faster R-CNN will be implemented to achieve very good performance, especially for detecting small or overlapping obstacles. Therefore, safe and precise navigation is guaranteed in complex and cluttered scenarios.

Testing and attempting to optimize the performances will be the project's last phase. Every model that is to be produced is rigorously tested using a variety of scenarios to make sure it performs effectively and accurately in all kinds of environmental situations. By fine-tuning the models for performance, it will smoothly integrate with drone operations, where obstacle avoidance depends on response times. Additionally, a closer examination of the model's outputs will be conducted using visualization tools to gain insight into its performance and areas that require improvement. This will allow the models to be fine-tuned, and the final models will have met the project's objectives for processing in a complex environment and high accuracy.

Project Contributions and Applications

Autonomous Navigation. This research increases drones' capacity to navigate challenging terrain with minimal human supervision.

Processing different obstacles in various scenarios. It makes a drone responsive to changing surroundings by enabling it to recognize and avoid obstacles in a variety of situations.

Safety Improvement. Reduces the possibility of drones colliding with objects like trees, buildings, or power wires, increasing drone safety.

Drone Autonomy. Increases the ability of drones to perform intricate tasks on their own without assistance from humans, advancing the field of autonomous drone technology.

The expected project applications could be as follows:

Aerial Surveillance. With better obstacle avoidance, it increases the possibilities for drone surveillance in urban security, wildlife tracking, and environmental monitoring.

Logistics and Delivery. Delivery services will become more efficient as a result of drones' ability to securely navigate metropolitan areas and avoid obstructions when delivering packages.

Agriculture. Helps farmers inspect crops and evaluate fields by enabling drones to navigate a complicated agricultural landscape.

Disaster Relief. Drones could be used more effectively in search and rescue operations since they can navigate through hazardous locations and debris on their own.

Infrastructure Inspection. Guarantees more safety and, thus, greater accuracy by making sure that drones may examine objects like wind turbines, electricity lines, and bridges without colliding with them.

Environmental Research. Allows safe navigation of rugged terrain that is hard to tackle with drones, hence helping researchers catch valuable data in remote or hazardous environments.

Project Requirements

Functional Requirements

From massive buildings and mesh structures to the more difficult small and thin structures like wires, poles, and branches, the obstacle detection system must accurately identify and classify a variety of obstacle classes. It is crucial for ensuring that a drone can navigate safely in a variety of environments, including parks, forests, and urban areas. Among its features are the ability for the drone to detect obstacles instantly and act quickly. A system must be robust if it can function consistently within ±5%, have at least a minimum accuracy threshold, and be unaffected by different weather conditions, such as rain, fog, or

intense sunlight. Confusion matrices and F1 scores will be used to evaluate performance; the goal F1 score for all obstacle classes should be at least 0.85. The detection system should be able to scale and operate flexibly in changing environmental conditions without requiring extensive retraining in an effort to further guarantee this adaptability. This would significantly increase the likelihood that drones would fly autonomously and avoid obstacles with little assistance from humans. It is necessary to deploy the application on a cloud platform, like Google Cloud Storage.

AI-Powered Requirements

Obstacle detection is empowered with sophisticated AI models, including YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. These ensure the detection of obstacles from large-size buildings and vehicles to small-size wires and tree branches effectively. Each model has different strengths-for example, the high detection speed of YOLOv11, the finer segmentation by Detectron2 in complicated scenes, real-time performance with robust accuracy in RT-DETR v3, and the very precise small or overlapped obstacle detecting ability of Faster R-CNN. Together, they come to create one system that will allow your drones to safely navigate almost any type of environment through the combined strength of each. Performance metrics will be concentrated on achieving at least 80% recall for smaller and harder-to-detect obstacles.

These AI models are designed to work effectively under various weather conditions, including rain, fog, and fluctuating lighting scenarios. The system will be optimized to process data in real time, thus enabling drones to make instantaneous decisions when faced with detected obstacles. Performance tests shall be carried out for consistency in different environmental conditions and strive to maintain accuracy within ±5%. Besides, the efficiency of processing will be further optimized to avoid excessive resource consumption on board the

drone, hence enhancing safety and efficiency in operation. This ensures that drones can reliably and efficiently navigate through challenging dynamic environments.

Data Requirements

It is evident that to ensure the training of AI models to detect obstacles under all possible environmental conditions, a highly strong and diverse dataset will be needed for the project. This will majorly be done on the DDOS-Drone Depth and Obstacle Segmentation dataset, which includes at least 10,000 frames with objects of interest labeled in different settings of the environment. The dataset would be in need of rich image features on RGB data, depth maps, segmentation masks, and surface normals, with metadata on drone positioning, weather, and timestamps that give the model context for training. In increasing the model's robustness, the dataset variation may also be enhanced through data augmentation such as image rotation, scaling, and brightness. The augmentation shall prevent overfitting and provide an increase of at least 50% in the size of the dataset for the models to generalize correctly to real-world scenarios. The data should be divided in such a way that each dataset is balanced and holds an appropriate and proportionate representation of obstacle type and environmental condition: 70% for training, 20% for validation, and 10% for testing.

Project Deliverables

Deliverables include key stages of the project, from the proposal to the final report. The list of project deliverables is shown in Table 1.

Project Proposal

The main objective of the project is to design an obstacle detection system powered by AI that can fly more safely and independently. For this purpose, the system can employ state-of-the-art deep learning models like YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN to detect and classify obstacles such as trees, buildings, wires, and moving cars. Because the system would work in many various conditions due to the DDOS dataset

included, there can be safe and effective methods for flying drones in cities and rural areas. On a high scale, drone navigation systems tend to be more reliable and generally scale well for different types of industry areas, such as surveillance, agriculture, or logistics.

Introduction

Integrating modern AI models, especially those used in drone navigation, the Obstacle Detection System rethinks autonomous flight. Equipped with models like YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN, among others, it will be capable of detecting static and dynamic obstacles even under adverse weather and light conditions. Key demands, which ensure increased drone safety and operational effectiveness, form part of this project's contributions. It will encompass everything from data gathering and labeling to augmentation, to applying strong machine learning models that would ensure high performance in challenging cases-from dense forests to city streets.

Project Management Tool and Work Breakdown Structure

The project will be managed using Jira software, which will be used to assign tasks, track work activity, and manage workflows. A project work breakdown structure that emphasizes project deliverables, activities, and milestones shall be prepared. These milestones include data preparation and collection, model training, evaluation, and system deployment. Furthermore, the specific responsibilities of each team member will be identified in detail.

Effort Estimates, Pert, and Gantt Chart Creation

Every project deliverable will have an effort estimate. The creation of the project timeline visualization will use both Gantt and PERT charts. The Gantt chart will show the length of the task, its dependencies, and its progress in relation to tasks like system testing, model development, and data preprocessing. Major project milestones, task interdependencies, and critical path activities will be highlighted in the PERT chart. All these

elements will be considered for the successful completion of the project within the given time.

Data and Project Management Plan

The paper will also outline the hardware and software requirements, methods of preprocessing, and the deadline for different tasks. It will explain how to collect data from the DDOS dataset and then annotate it. The development of Gantt and PERT charts showing the progress of phases with resources and deliverables associated with them will also be explained. The final paper will include a comprehensive plan concerning data flow and project management.

Data Collection and Data Exploration

In general, the collection would come from the DDOS dataset since it contains a variety of pictures of the drone flight path under different environmental conditions. Considering that photos will be annotated to detect obstacles such as trees, buildings, wires, and cars during preprocessing. To ensure equal distribution of obstacle types in the dataset, it should be divided into training, validation, and test datasets. Different augmentations will be used to improve exploratory data analysis. Rotation, brightness changes, and noise addition will be applied, increasing the dataset's size by over 50% and enhancing the model's capacity for generalization.

Data Pipeline

The data pipeline to be delivered in this project provides an end-to-end workflow automation that will seamlessly integrate all the phases of data processing, such as ingestion, annotation, pre-processing, transformation, and model training. Using avant-garde tools and cloud platforms like Google Cloud Storage, Apache Beam, and Cloud Composer, the pipeline will assure the efficient handling of imagery captured by drones and their metadata. It will be annotated based on SAM, bringing complete precision in obstacle identification; it will also

be preprocessed by resizing, normalizing, and augmenting the data, along with structured transformations to make the data ready for training models. This pipeline will be scalable, efficient, and real-time, providing a ready-to-use obstacle detection model system that can show high-speed, accurate performance by navigating drones in dynamic environments.

Prototype

A prototype of the obstacle detection system will be designed in such a way that advanced models of AI, namely YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN, will be used. It shall be trained in the detection and classification of obstacles in real time during operations. These models will be continuously optimized for various weather and lighting conditions in order to negotiate them for the prototype. This would also help in demonstrating the system's capability for a wider range of obstacle detection, in both static and dynamic environments.

Data Engineering

Integration of trained AI models with the DDOS dataset would involve core data engineering. Ensuring that the data is cleaned, preprocessed, and stored using scalable cloud-based solutions like Google Cloud Storage will help ensure the system is prepared for big datasets during model training. The continuous collection of feedback from system performance metrics through the data engineering process will enable iterative improvements in the models and system as a whole.

Model Development and Testing

The model development and testing will be done by embedding previously trained YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN models, optimized using the DDOS dataset. The models will be taught to recognize different types of barriers. The testing of the model will be done using metrics from accuracy, precision, recall, and F1 score. These metrics will help determine the degree to which the system can handle the various

environmental situations without danger. In addition, model testing should be flexible to carry out in various settings.

Development Applications

The system that has been developed as part of this project would have various uses, including making the process of navigation and operating an autonomous drone safer and enabling such operations in very diverse settings: from cities to forests to rural landscapes. The obstacle detection system could find a wide range of applications, including agriculture for monitoring crops, logistics for the delivery of packages, and disaster relief for search and rescue activities.

Project Presentation

The PowerPoint presentation provides an overview of the project and how the research is being applied to solve it.

Group Project Report

The Group Project Report will focus on an overview of the project's goals, methodologies, challenges, and findings. In addition to emphasizing AI model selection, dataset preparation, model training, and system testing, it should discuss the contributions of every team member assigned to the project. It should also discuss the performance evaluation of each AI model by contrasting how well each model detects obstacles of different types and under different conditions.

Table 1

Project Deliverables and Timeline

Deliverable	Description	Due Date
Project Proposal	Propose Paper For the project	09/03/2024
Project Breakdown	Creation Of WBS In Jira	09/10/2024
Project Plan	Creation Of Tasks, Subtask And Task Assignment, Gantt Chart And Pert Chart	09/17/2024
Data Collection	Collection Of The Data	09/30/2024
Data Engineering	Data Cleaning And Data Preprocessing	10/14/2024
Prototype	Model Building	10/28/2024
Testing	Evaluation And Validation of The Models	11/10/2024
Development Application	Design Automation	11/20/2024
Final Model	Integration And Deployment of The Model	11/30/2024
Project Report	Documentation of The Project	12/10/2024

Note. The above table illustrates the project deliverables

Technology and Solution Survey

Singh, R. et al, (2021) applied the Detectron2 and EfficientDet methodologies for helmet detection; thus, it will be interesting to look at how such models function in object detection tasks characterized by both precision and scalability. The modular framework in Detectron2 is designed for instance segmentation and object detection; handling overlapping objects and complex visual situations is at the core of effectively distinguishing helmets against challenging backgrounds. While EfficientDet brings efficiency with BiFPN and compound scaling, it allows for solid multi-scale object detection while optimizing resources. The work in this paper will guide us on how to conduct the project by emphasizing accuracy

versus computational efficiency trade-offs for leveraging these models in drone obstacle detection. This approach ensures that the selected models effectively address the various challenges provided by different obstacle types and flying conditions.

Abdallah, A. B. at al, (2022) presented the ability of Detectron2 to perform instance segmentation, especially in complex scenarios such as differentiating olive trees from their shadows. The robustness of Detectron2 in segmenting and classifying overlapping or ambiguous objects correctly speaks to its robustness for complex detection tasks. Key strategies are discussed in the paper that are crucial to enhance model accuracy, including advanced hyperparameter tuning and augmentation techniques. These insights are directly applicable to our project, where obstacles like wires or tree branches may appear in cluttered or low-visibility settings. Based on the methods discussed in this paper, we can fine-tune Detectron2 for precise obstacle detection that will be reliable in a variety of challenging drone flight conditions.

R. C. V et al, (2024) presented in great detail the multi-modal functionalities of Grounding DINO and highlighted how it could seamlessly integrate textual and visual information to enhance object detection and classification. The transformer-based architecture of the grounding DINO is amazingly clear in grounding textual prompts over their specific visual regions for conducting highly accurate obstacle annotation and detection. This research output on prompt engineering and model scalability will significantly boost our project by automating most of the tasks used to annotate images captured with a drone. In the process, this makes it very certain that Grounding DINO effectively classifies obstacles under various situations with high precision using efficient techniques, hence boosting both accuracy and context in our detection.

Rahim, A. et al, (2023) undertook in-depth analyses to ground the multi-modal competence of DINO, mainly focusing on how the model integrates both visual and textual

data into superior object detection and classification. With a transformer-based architecture, Grounding DINO performs exemplary in grounding textual prompts to specific visual regions and enables very accurate obstacle annotation and detection. The results of this study regarding prompt engineering and model scalability will help our project extensively in simplifying the process of annotation of images captured by drones. This would, in turn, assure us that Grounding DINO identifies and classifies obstacles in difficult scenarios with much efficiency, thereby giving a big boost to the accuracy with contextual understanding of our detecting system.

E. L. Thin Jun, M. -L. Tham and B. -H. Kwan (2024) compared RT-DETR and YOLOv8 in their performance of urban aerial object detection to give a comprehensive comparison study of their respective strengths and shortcomings. RT-DETR presents superior performance with its Transformer-based architecture in real-time detection with high precision. YOLOv8 provides a fast inference speed together with competitive accuracy by the design in CNN. The contribution of this study is on providing further insights into speed versus accuracy for drone navigation metrics, such as mAP and FPS. These findings are critical in guiding our project on model selection and optimization to ensure the effective detection of obstacles while keeping responses in real-time in an urban dynamic environment.

In this paper, Wang, S., Xia, C., Lv, F., & Shi, Y. (2024) developed an extended version of the transformer-based architecture, RT-DETRv3 with hierarchical dense positive supervision, additional auxiliary branches, and improved attention mechanisms. Further precision improvements are allowed by ensuring a low latency so that the system can run in real time for resulting systems. Extensive architecture optimization and a discussion on computational efficiency will be of great value to our project, targeting real-time obstacle detection using resource-constrained models on drones. In implementing these, we make sure

to maintain a highly precise and responsive detection system on onboard drone hardware despite computational constraints.

Deng, Z. et al. 2017, proposed a Coupled Region-Based Convolutional Neural Network to detect vehicles from aerial images. Optimized by training on data sets of urban and rural variable settings, this model was fit for recognition by the size and orientation of these usually mis-sized vehicles. This had outperformed earlier conventional methods and increased vehicle detection accuracy by 4.8%. It outperforms state-of-the-art very high-performance systems and hence would fulfill the demands of various urban surveillance applications like monitoring of congested traffic conditions. On using the proposed system with large test areas and high density in the number of vehicles, scaling issues have been mentioned. However, it was an important step toward precise studies in aerial imagery vehicle detection.

Hassan, S. A., Rahim, T., and Shin, S. A. (2019) proposed a real-time UAV detection system by combining YOLOv4 with other deep learning models. As this system is intended for practical applications, the main focus of this model was on finding an optimal trade-off between speed and accuracy. This system was more appropriate for disaster management applications and real-time UAV tracking, where it achieved an accuracy of 92.3% operating at 28 frames per second. The long-range detection of tiny UAVs was the failure point of the system, which could be used as a defense against its use in a wide-ranging surveillance area. Considering these points, the model's balance between speed and accuracy is highly efficient for UAV detection in a dynamic environment.

Kucukayan, G., and Karacan, H. 2024, came up with the YOLO-IHD for indoor drone navigation, an improved YOLO model. As far as the detection of target objects is concerned, this system, which aimed at human detection indoors, in offices, and warehouses, was found to be more effective by 7% than the standard models. It is helpful in several dynamic barrier

settings, with people where there is an assurance of safety and security. As good as this model is, it performs exceptionally well at low light, increasing its robustness for real-world indoor situations. Performance will degrade when the area becomes highly cluttered due to object overlap.

Dafrallah, S., and Akhloufi, M. (2024) studied the use of multi-modal data in the detection of malicious UAVs by fusing data from thermal, radar, and visual sources. Their system outperformed a single-modality system with an accuracy of 89.7% in distinguishing between benign and malicious UAVs. High-risk situations, such as military zones or locations with major infrastructure significance, call for multimodality. The computational demand of the system makes it almost impossible to scale for environments that require quick deployment and low hardware infrastructure costs. This work, however, provides a crucial foundation for bettering the techniques of detection of UAVs in sensitive areas using Dafrallah, S., and Akhloufi, M. (2024).

Deep reinforcement learning was adopted by Solaimalai, G. et al. (2024) to propose a UAV navigation system that can interact with obstacles in cluttered environments. The proposed DRL-based system attained success rates as high as 94.5% over obstacle-heavy terrains such as urban settings and forests. This model continues to enhance its path planning accuracy with increasing interactions with dynamic obstacles. However, the system also requires a lot of computational power during training, which has been a major disadvantage. This contribution relates to the larger field of UAV autonomous navigation and provides evidence on the effectiveness of DRL in real-time route planning and decision-making.

This work by Zhao et al. (2018) is a review aimed at comparing the state-of-the-art deep learning models in detecting objects from autonomous systems such as drones using the YOLO, SSD, and Faster R-CNN approaches. They also provided very valuable and complete comparative research; the YOLO model is much faster while keeping similar results, the SSD

models represent the trade-off of speed with accuracy, while the best performance, by far, will come with the most computationally expensive: the Faster R-CNN approach. The review also mentions that hybrid models may generate even better solutions for real-time applications by fusing the advantages of these three models. Thus, this review gives a clear guideline on how to choose an object detection model suitable for UAV applications by Zhao et al. (2018).

A deep learning-based framework for the deforestation detection and health assessment of the forest using UAV-based observation is developed by Marques, T. et al.(2024). It has an accuracy in identifying change of vegetation cover of up to 93.2%. Therefore, it serves best for wide and comprehensive environmental monitoring. The system's limitation was that "high-resolution aerial photos may involve various computational problems when being processed real-time on vast regions.". This system could be improved to support faster speeds, but it has good scalability for environmental conservation. Conclusion The employment of autonomous UAVs in enhancing forest monitoring is advantageous to the conservation step.

A multi-task learning framework was developed by Fang, W. et al.(2023) to enhance obstacle detection performance in low visibility situations, such as foggy weather conditions. Therefore, by fusing object detection algorithms with image enhancement, the accuracy of detection increases by 15% in the worst weather conditions. This model is extremely useful in the case of UAVs deployed in hazardous search and rescue situations. However, the system's performance is limited by its ability to function effectively in real time under extremely dynamic environments due to the high computational cost. Notwithstanding these drawbacks, the research from Fang, W. et al.(2023) offers important improvements for the operation of UAVs in inclement weather.

The research developed a collaborative DRL approach: Zhao, Z., Wan, Y., and Chen, Y. (2024) showed that several UAVs can work together to detect and avoid obstacles. In contrast, single-drone systems can enhance the obstacle detection performance by 12%. Particularly helpful will be the latter in broad operations, such as environmental monitoring or disaster response, where multiple drones are required to provide the full scope. The central challenge, however, is whether the efficiency of deployment will be compromised by the complexity of managing multiple UAVs simultaneously. Despite this, the approach has the potential for modifying the coordination of UAVs in hazardous situations.

Katkuri et al. (2024) proposed a deep learning-based computer vision technique for an autonomous UAV navigation system. The model is based on a CNN architecture that processes live video feeds to detect and avoid obstacles in dynamic environments. This system is beneficial in situations involving autonomous UAV surveillance because it achieved 90% detection accuracy. In situations that require real updates all the time, like very complex ones, scaling remains a challenge. While this model has shown promise for enhancing the level of UAV autonomy, it as yet needs further optimization for even more complex and larger environments.

Huang et al. (2024) applied YOLOv8 for research on a lightweight drone detection system. It focused on various ways to improve the efficiency of the detection while performing real-time operations. The model can be installed in drones with very little processing power since it has 95% of the detection accuracy with very small computational overhead. It did exceptionally well when operating in a congested urban environment where accuracy and speed are key. It is yet unknown, though, if it functions well in other environments such as cluttered or rural ones. This work will look into the possible application of YOLOv8 in resource-constrained UAV applications and present an effective implementation method.

A real-time path modification-based dynamic path planning obstacle avoidance system was designed for the UAV using an RL-based approach by Tu, G. -T., and Juang, J. -G (2021). In cluttered conditions, its success rate reached as high as 92.5%, outperforming all conventional methods with RL-based approach. Its key defect lies in the extremely high computation cost caused by real-time path modification. This work does, however, illustrate how RL-based techniques can significantly improve the navigation of UAVs in dynamic and uncertain environments.

In enhancing obstacle detection for low-flying drones in an urban environment, N and U. S. V. (2021) used YOLOv3. This was more focused on dynamic and small obstacles like pedestrians and moving cars, with a small obstacle detection rate up to 88.9%. Because of this, it will be highly appropriate for use in scenarios such as drone delivery services and urban logistics. Nevertheless, the test showed some low-light performance problems that necessitate further optimizations. In general, this model could serve well in low-flying drone operations for such challenging urban scenarios.

Sayed, L. J., Souripalli, P. K., and Chiddarwar, S. S. (2024) proposed an X-DRL model for UAV navigation through foggy conditions. From their model architecture, interpretability has been kept very salient so that operators can easily judge the decision-making of the UAV model. In this scenario, they could achieve very high accuracy-87.6% for very dense fog conditions using this approach. For maintaining a trade-off between interpretability and the autonomy of a model, This faces challenges in real-time applications, though, because the model's complexity impacts the processing speed. The authors say this is a significant advance of the field of explainable AI in autonomous UAV operations.

Hsieh, Y.-Y. et al. (2019) developed a novel approach for UAV navigation that combines obstacle detection with deep learning and depth estimation. Their system, which is based on a stereo vision setup and CNNs, evaluates an aerial image stream in real time to

determine the distance of obstacles and prevent collisions. The model's 91.2% obstacle detection rate in dynamic environments and corresponding depth estimation make it ideal for tasks involving urban surveillance or search and rescue. This will make the system much safer because it combines depth estimation and obstacle detection, especially in cluttered environments. However, some issues of real-time processing must be resolved for the flying of drones to meet industrial standard requirements. Table 2 shows the technology and solution comparison of the above surveys.

Table 2

Summary of Technology Survey

Authors	Dataset	Models	Results	Conclusion
Deng, Z. et al. (2017)	Aerial Image Dataset	Coupled R-CNN	4.8% improvement in vehicle detection accuracy	Suitable for urban monitoring but lacks scalability
Hassan, S. A., Rahim, T., and Shin, S. A. (2019)	UAV Detection Dataset.	YOLOv4	92.3% detection accuracy, 28 FPS	Highly suitable for real-time UAV detection
Kucukayan, G., and	Indoor Drone Navigation Dataset	YOLO-IHD	7% improvement in human detection accuracy	Effective in low-light indoor environments

Karacan, H. (2024)				
Dafrallah, S., and Akhloufi, M. (2024)	Multi-Modal UAV Detection Dataset	Multi-Modal Data (Radar, Visual, Thermal)	89.7% detection accuracy for malicious UAVs	Effective but computationally expensive
Solaimalai, G. et al. (2024)	Cluttered Environments Dataset	Deep Reinforcement Learning	94.5% success rate in navigating cluttered environments	Requires high computational power during training
Zhao, Z. et al. (2018)	Various Datasets	YOLO, SSD, Faster R-CNN	Comprehensive review of object detection models	Emphasizes need for hybrid models
Marques, T. et al. (2024)	UAV Forest Monitoring Dataset	CNN	93.2% accuracy in detecting forest degradation	Scalable for environmental conservation, but not real-time
Fang, W. et al. (2023)	Foggy Weather Dataset	Multi-Task Learning	15% improvement in obstacle detection accuracy	Effective in foggy conditions but computationally expensive

Zhao, Z., Wan, Y., and Chen, Y. (2024)	Deep Reinforcement Learning	Deep Reinforcement Learning	12% improvement in obstacle detection accuracy	Promising for large-scale operations with multi-drones
Katkuri, A. V. R., et al. (2024)	Real-Time Video Feed Dataset	CNN	90% detection accuracy in dynamic environments	Needs optimization for large-scale operations.
Tu, G. -T. and Juang, J. -G (2021)	Path Planning Dataset	Reinforcement Learning	92.5% success rate in navigating cluttered environments	High computational cost for real-time path adjustments
N and U. S. V (2021)	Low-Flying Drone Dataset	YOLOv3	88.9% accuracy in detecting dynamic obstacles	Effective for urban logistics and drone delivery
Sayed, L. J., Souripalli, P. K., and Chiddarwar, S. S. (2024)	Foggy Environment Dataset	Explainable DRL	87.6% accuracy in foggy weather	Balances autonomy and explainability
Hsieh, Y. -Y. et al. (2019)	Stereo Vision UAV Dataset	CNN	91.2% accuracy in detecting obstacles and estimating depth	Combines depth estimation with obstacle detection

Singh, R. et al, (2021)	Helmet Image Dataset	Detectron2, EfficientDet	High accuracy in helmet detection with scalable performance	Detectron2 provided precise segmentation, while EfficientDet ensured lightweight detection for resource-constrained environments
Abdallah, A. B. at al, (2022)	Agricultural Imagery Dataset	Detectron2	Robust segmentation of olive trees and shadows	Demonstrated efficient instance segmentation, making Detectron2 suitable for complex drone imagery tasks
R. C. V et al, (2024)	Custom Text-Image Dataset	Grounding DINO	Enhanced text-to-image correspondence with contextual grounding	Grounding DINO enabled accurate visual grounding, proving effective for obstacle detection through textual prompts

Rahim, A. et al, (2023)	Traffic Imagery Dataset	Grounding DINO, SAM	Automated and precise vehicle annotation for large-scale datasets	The combination of Grounding DINO and SAM offered high-quality annotations, suitable for training drone-based detection models efficiently
E. L. Thin Jun, M. -L. Tham and B. -H. Kwan (2024)	Urban Aerial Dataset	RT-DETR, YOLOv8	YOLOv8 achieved faster inference; RT-DETR delivered higher accuracy	Highlighted the trade-offs between speed and accuracy, offering insights for selecting models for dynamic obstacle detection scenarios
Wang, S., Xia, C., Lv, F., & Shi, Y. (2024)	RT-DETRv3 Benchmark Dataset	RT-DETRv3	Improved detection accuracy with reduced latency	RT-DETRv3 proved ideal for real-time applications due to its hierarchical dense supervision and computational efficiency

Note. The above table illustrates the summary of the technology survey

Literature Survey of Existing Research

Ackermann, Marc, et al. (2022) presented an automated MA island segmentation technique in bainitic steel using sophisticated computer vision methods. While it is based on materials science, several key aspects to active segmentation can be transferred to the processing of drone-based obstacle detection. Its methodology of applying robust segmentation strategies can help in the detection of minute details in drone flight paths, such as thin wires or branches, which are often challenging to identify. The insights into precise segmentation enhance the project's focus on creating a reliable pipeline for object recognition under complex scenarios, ensuring that smaller or less distinguishable obstacles are effectively detected.

Wadhwa, Mehul, et al. (2023) will compare YOLOv8 and Detectron2 with an emphasis on their performances in crowd counting. The evaluation of these models provides insights that can be valuable for the purpose of balancing speed with accuracy within dense environments. In relation to our project, the comparative strengths of Detectron2 in segmentation and classification under challenging conditions resonate with its role in detecting obstacles in cluttered drone flight paths. The paper also highlights key considerations for model selection, such as adaptability to real-time applications and precision, which align with our project's objectives of robust obstacle detection.

Ren, Tianhe, et al. (2023) explored the different developments in the model for grounding and detection tasks called Grounding DINO. It can merge textual and visual inputs to provide a better contextual understanding, thus enhancing its potential use in drone navigation systems. Using such a model, our project will be able to detect obstacles with superior capability: it will be able to understand semantic cues such as distinguishing between different object categories based on their textual description. The detailed

explanation of architecture and improvements in grounding accuracy align well with our focus on high-precision detection in diverse and dynamic environments.

Liu, Shilong, et al. (2024) improved the functionality of DINO by incorporating grounding, thereby enhancing object detection and annotation processes. This novelty in fusing text with visuals results in improved performance of models involving contextual understanding tasks. It would ensure that the annotations developed in the pre-processing are contextually correct and thus enable our model to handle complex flight routes more effectively. The emphasis on flexibility and precision directly supports the project's aim to develop robust detection systems for drones.

Zhao, Yian, et al. (2024) discussed the Real-Time Detection Transformer (RT-DETR) and the advantages it has over the traditional YOLO models in real-time scenarios. Removing NMS significantly improves the speed of RT-DETR without loss of accuracy. For drone-based obstacle detection, this paper's findings emphasize RT-DETR's potential in high-speed real-time processing, which is crucial for the operation of drones in dynamic environments. The model's adaptability and superior performance metrics strengthen its suitability for integration into our data pipeline.

Zhuofan, Yu (2024) focused on optimizing RT-DETR for small object detection by lightweight design and improved loss functions. Improvement in efficiency and precision will directly address the challenges in detecting small obstacles, such as wires or poles, in drone flight paths. Its emphasis on reducing computational complexity while maintaining accuracy aligns perfectly with our project's goal to develop resource-efficient solutions for real-time applications. This paper serves as a guide for optimizing models to meet the specific demands of drone navigation systems.

Guo et al. (2023) designed an algorithm for enhancing obstacle avoidance with target tracking of drones using deep Q-networks. The use of reinforcement learning in the system

will enable the drones to come out with the best navigation strategy through trial and error. The experiments proved that obstacle avoidance maintains high precision over moving targets in dynamic environments like urban landscapes or forests, and is successful 92.4% of the time. It thereby reduced manual interference while presenting major steps ahead in the autonomy of drones. However, the high computational requirements that remain make its applicability limited to smaller drones with low processing power.

Application of deep learning models for drone obstacle avoidance while guaranteeing goal-directed navigation was investigated by Capi, G., Sugiyama, N., and Kaneko, S.-i. (2021). CNNs combined with a reinforcement learning framework let the system navigate through complex environments with an 87% success rate. The purpose was to ensure that drones could autonomously adapt their flight paths in order to approach a target and avoid different stationary and moving obstacles. When applied in a rural area with unstructured obstacles, the model did not perform as well as it did in an urban environment. This will be a great domain for the development of drone navigation systems, especially those applied in delivery and logistics.

Rezaee, M. R. et al. (2024) have given a comprehensive analysis of the current drone collision avoidance systems and also highlighted the key obstacles and areas for future development. The avoidance strategies are divided into algorithmic approaches in this paper, which include sensor-based and vision-based systems. In addition, it highlights the shortcomings of each of these avoidance techniques. The authors' discussion is based on the necessity of multi-sensor fusion and machine learning techniques as promising ways to overcome these issues, especially those regarding poor performance in cluttered environments. Thus, an analysis of the advantages and disadvantages of the different systems discussed in this study provides valuable information for future advancements in UAV safety

and dependability. However, as long as testing and real-world implementation remain pending, most of the research done up to this time falls short (Zhu et al., 2022).

To support the development of more precise models for obstacle detection for UAVs, Kolbeinsson, B., & Mikolajczyk, K. (2023) created a new dataset they called the Drone Depth and Obstacle Segmentation (DDOS). It contains depth maps, RGB pictures, and mappings with segmentation annotations from urban to rural environmental types. The dataset helped better train and evaluate the performance of deep learning models, especially for more complex environments that include thin structures such as wires and branches, the authors have said. It is expected that this will be of great benefit for the improvement of safety and dependability in high-risk autonomous drone scenarios, such as disaster response.

Katkuri, A. V. R. et al. (2024), performed deep reinforcement learning to enable a UAV to be more capable of performing in unseen environments. With no prior mapping, the model autonomously implemented a DRL-based framework that allowed drones to adapt to new environments. In such smooth flights through urban areas and obstacle-heavy fields or forests, the system recorded a success rate of 91.6%. In return, the study extracted a very high computational load for which real-time performance degradation was foreseen for the learning algorithm, encouraging the idea development for even more intelligent and adaptive UAV systems, directed to applications related to ground exploration and surveillance.

Darwish, A. A., and Nakhmani, A. (2023), in their model of deep reinforcement learning, incorporated a cascade reward mechanism into the drone to improve its navigational capabilities and target interceptions. Their system allowed for making complex behaviors-such as avoidance of obstacles and movement toward a moving target-by the drones in a very proficient manner. In comparison to the traditional Q-learning methods, a high model performance in target interception accuracy of 88.9% is produced. The cascade reward structure in the approach contributed much to balancing navigation and interception tasks,

hence an adaptive strategy in UAVs operating in dynamic environments, like surveillance missions.

A. N. and U. S. V. (2021) compared Faster R-CNN, YOLOv3, and SSD as three deep-learning algorithms for drone object detection. According to the findings of this work, YOLOv3 ranked highest among those tested models with an accuracy of 90.2% on real-time object detection. The main objective of the project was to increase the speed and precision of drone detection, especially in places with many moving objects like cities and building sites. It realized its weaknesses regarding the detection of smaller objects and recommended that any future improvements be geared toward more effectively navigating around obscured or partially visible obstacles.

Li, Y. et al. (2024), through modifying YOLOv5 and further optimizing it into a tunnel environment, proposed a lightweight, real-time obstacle detection model. The research focused on how to improve the speed and accuracy of obstacle detections, such as tunnel walls and building materials, with a 93% obstacle detection rate. This can enable a lightweight model architecture to run on devices with limited resources, such as mobile robots or drones, that have small operating areas. The authors mention that Zhou et al. (2023) developed similar approaches because this is crucial in underground construction and inspection tasks, for which the traditional models run too slowly or require too many resources.

Wang, W. et al.(2024) researched obstacle avoidance and target tracking systems on drones automatically using deep learning in 2024. The proposed system presents a modified YOLOv4 algorithm that offers up to 91% tracking accuracy in relation to real-time object detection and tracking. This should enhance autonomy in different types of applications, say search and rescue, in which the tracking of mobile targets is quite critical, especially avoiding dynamic obstacles. Also, this helps to improve the performance over challenging terrain and

tackles weaknesses in earlier systems that tend to primarily fail when an environment possesses high obstacle densities, Li et al, (2023).

Isaac-Medina, B. K. S., et al. (2021) focused on the visual-based advancements in drone tracking and detection using deep neural networks. The authors investigated the use of CNNs to enhance the accuracy and efficiency of real-time drone detection. The authors proposed a powerful multitask learning model that utilizes pre-trained networks for feature extraction, such as ResNet, while simultaneously carrying out tracking and detection. The work contributes mainly by handling dynamic and unpredictable environments. Hence, it finds a perfect fit in various applications in defense and surveillance systems. Therefore, the prime contribution of this work in this study was the outstanding performance in significantly improving the accuracy of detection, outperforming the traditional models, particularly under low light levels with much denser populations.

Gervi, P. R., Harati, A., and Ghiasi-Shirazi, S. K. (2021), have also proposed a DRL method to achieve vision-based obstacle avoidance while piloting drones. They have proposed a DRL framework that could enable the drones to avoid obstacles while learning the best flight path in dynamic environments simultaneously. By merging CNNs with reinforcement learning algorithms such as DQN, the drone can adapt to difficult terrain and make decisions autonomously. It performed better than the state of the art in a simulator by reducing collision rates and increasing the accuracy of obstacle detection. Future applications in rescue operations and urban exploration can be envisioned.

Yaseen, M. (2024) discussed the recent release of YOLOv9, an improved object detection version of the YOLO series with further improvements in the directions of accuracy and computational efficiency. Two of the major novelties of YOLOv9 are GELAN-generalized efficient layer aggregation network and PGI-programmable gradient information. These provide a solution for the bottlenecks of gradient flow and feature extraction. These

make possible significant reductions in computational complexity with high accuracy. Extensive comparisons between YOLOv9 and its predecessors demonstrate state-of-the-art improvements, including a 49% reduction in parameters and a 0.6% improvement in accuracy on MS COCO. It highlights how YOLOv9 can be used for real-time drone detection across a wide range of hardware platforms. The work of Yaseen et al. was completed in 2024.

Yilmaz, B., and Kutbay, U. (2024) The proposed performance analysis for the drone YOLOv8 detector concentrates on using data augmentation and tuning hyperparameters to find the optimized model. The authors utilize the UAV dataset, which is publicly accessible and allows the simulation of different conditions by applying augmentations such as rotation, scaling, and blurring. With a precision of 0.946 and recall of 0.9605, YOLOv8 outperformed even more conventional models like Mask R-CNN and previous versions of YOLO. This work really depicts that proper model tuning and augmentation can help a lot in more accurate detection, especially in security and defense applications where prompt and precise detection is crucial.

Wang et al. (2024) helped solve two major problems of deep neural networks: gradient instability and information loss. In this paper, Programmable Gradient Information is proposed, referred to as PGI. YOLOv9 depends on it to improve model precision and dependability by keeping more information intact during the training process. It proposes GELAN, which stands for a generalized efficient layer aggregation network, besides the first, for further improvement of parameter utilization and reduction of computational overhead. They have used the MS COCO dataset for testing their methodology, and it shows that YOLOv9 can perform better than its predecessors in both accuracy and inference speed.

Data and Project Management Plan

Data Management Plan

A Data Management Plan (DMP) tool is used to guarantee that created or gathered data is categorized, stored, analyzed, and disseminated. It facilitates data management by professional, ethical, and regulatory criteria.

Data Collection Approach

The data for the study will primarily be sourced from the DDOS dataset, a source of rich, high-quality data regarding drone flights: movement paths, weather, captured images, and obstacle depth. The data will be divided into training, validation, and testing, depending on the environmental settings, such as parks, neighborhoods, etc. This will be well effective in ensuring the even distribution of the data across the different scenarios. Additional information is added when the enlargement of the dataset is required to fill the gap in performance or improve the model.

In this light, the complementary data should be aimed to represent all possible environmental situations, such as fog, rain, low light intensity, or highly populated metropolises. It would be useful if these conditions come from crowdsourcing, open-source drone imagery, or computer-generated environments based on the challenge scenario. The resultant effect is that it will cope better with challenges in dynamic and complex real-world situations and improve the variability in training data. Data supplementation with obstacle types of various kinds and in a wide range of environmental scenarios provides improved robustness in obstacle detection and avoidance tasks, enhancing the model's capacity for generalization under diverse circumstances.

Data Management Methods

Data management allows version control to be implemented for each update of the dataset through a Google Cloud storage bucket. Metadata is organized into CSV files through

a Python metadata extraction script. Updates will be performed regularly to keep the data updated given new obstacle types and conditions. Structured data allows any new information or updates to be seamlessly integrated into model development. Metadata annotation, on the other hand, allows for appropriate classification, thus enabling efficient retrieval and analysis.

Data Storage Methods

This should then be uploaded into GCS, which is a highly scalable, secure cloud storage and functions well with Google Cloud's data processing tool, as highlighted in Table 3. The organization of hierarchical data can enable the access and management of views efficiently. Drone images, depth maps, segmentation masks, and metadata-for example all be stored in separate folders to enable easy access in later steps of the project. This provides an effective, well-defined structure for which relevant data can promptly be made available to train and evaluate the models. Besides GCS, BigQuery-an effective, serverless data warehouse provided by Google-manages and stores large datasets and their metadata.

This will include extracting significant metadata from raw data in CSV format by building indexes to the entire dataset. This would, in turn, enable better storage management. For sorting, the metadata will use critical parameters like flight IDs, obstacle types, environmental data, image paths, depth maps, etc. This kind of arrangement in the metadata would be very easy to search and analyze. Additionally, BigQuery will allow for efficient querying of large datasets, which will speed up access to certain data points that may be required for real-time processing and model development. Regular backups have been planned to avoid data loss or corruption during and after storage.

Table 3

Dataset, Data type, and Data Size

Dataset	Data Type	Data Size
DDOS (Drone Depth and Obstacle Segmentation)	Images, Depth Maps, Metadata	~256 GB
Metadata (depth, flow, segmentation, and surface normals)	CSV, Relational Data	~445 KB for each flight

Note. The above table illustrates the Dataset, Data type, and Data Size

Data Usage Mechanisms

Building these machine learning preprocessing pipelines that clean, annotate, and preprocess the data comes after the steps involved in storage and organization. So, the basic actions should include standardization of the data formats, adding noise, rotations, scaling, filtering out either irrelevant data or simply cleaning up noisy ones, along with augmenting such data into synthetic variations. BigQuery will store the metadata that can be used to query subsets of the dataset according to environmental conditions or obstacle types. To these carefully selected datasets, train the YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN models. Use this cleaned data in a feedback loop where the performance of drones gets fed back into the system, increasing the model's accuracy continuously. Visualization tools such as Looker Studio or Python libraries like Matplotlib are used to ensure the performance and accuracy of the various models so that the detection and obstacle avoidance meet the project requirements. Table 4 shows the assignment of accountability for each member in this project.

Table 4

Assignment of Accountability

Role	Assignee	Tools/Steps Taken
Data Collection	Vansh	Hugging Face dataset cloning, drone data
Data Cleaning and Preprocessing	Harsh	Python scripts, and OpenCV for image cleaning
Data Storage	Tanvi	Google Cloud Storage (GCS)
Data Management	Akshay	Jira, Metadata management, version control
Ethical and Legal Compliance	Harsh	CCPA, GDPR compliance

Note. The above table illustrates the Role and Assignee for each task

Project Development Methodology

CRISP-DM Methodology

For this project, the Cross Industry Standard Process for Data Mining, commonly called the CRISP-DM methodology, has been followed. The CRISP-DM model provides an organized and systematic process to develop the obstacle detection system using techniques of machine learning and deep learning. In the entire project life cycle, the six main stages of problem understanding, data collection, data processing, modeling, training, and evaluation

precede the deployment of the system. With this methodology, each phase can be well-planned and executed on time.

Planned Project Development Processes and Activities

Problem Definition. It should provide automatic obstacle detection to flying drones with the help of a deep learning model. The enhancement of its navigating abilities becomes necessary if it is to work among lots of obstructive objects like buildings, trees, poles, and wires among many others. The most significant impact of the work would thus be to develop an AI that identifies such obstacles with swift efficiency for a safe autonomous flight. Hence, the objective was to design an obstacle detection model based on AI that could identify obstacles in variable environmental conditions accurately for a drone, improving dependability in different industries related to agriculture, logistics, and surveillance.

Data Collection. Data collection entails extracting a huge dataset from the DDOS dataset, which includes photos with depth maps and segmentation masks for obstacles in diverse flying settings. The data is split into environmental circumstances, such as weather (fog, rain, sunshine, and so on) and location or backdrop (urban, rural, and forest), ensuring that the model encounters a variety of challenges and environmental situations. The data is safely stored in GCS and can be easily retrieved from there for further processing and model building.

Data Analysis. Then, the collected data undergoes a series of preprocessing steps such as formatting, cleaning, augmentation, and annotation. Data annotation is the process of labeling different objects present in the photos, such as wires, buildings, and trees. Data augmentation increases the variety in the dataset by making it look more realistic through adjustments in brightness, scaling, and rotation. Cleaning will ensure that unnecessary or noisy data has been removed and, consequently, that only high-quality input is delivered to

the model training phase. Hence, the data now is ready to undergo various analyses and transformations.

Model Development. In an improved system, obstacle detection models would include support for model selection: YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. During this phase, GCP infrastructure will be utilized to execute the deep learning models, PyTorch and TensorFlow. In the best possible scenario, each model might be trained on a pre-processed dataset. This model needs several tests and adjustments to have better performance and stabilize the entire system even further with top-notch accuracy. This infrastructure enables rapid iterations and model development and allows model training to be scaled.

Model Evaluation. The performance of these built models will be verified on several precision metrics, including accuracy, precision, recall, F1 score, and Intersection over Union (IoU.) These metrics ensure the model performs the task of detecting obstacles under various conditions and scenarios correctly and that it classifies them as required. A range of test data sets, including an unseen test set, will be used to establish how well the models generalize. These metrics are qualitatively complemented by a manual check of the prediction to make sure that the model works in real-world scenarios.

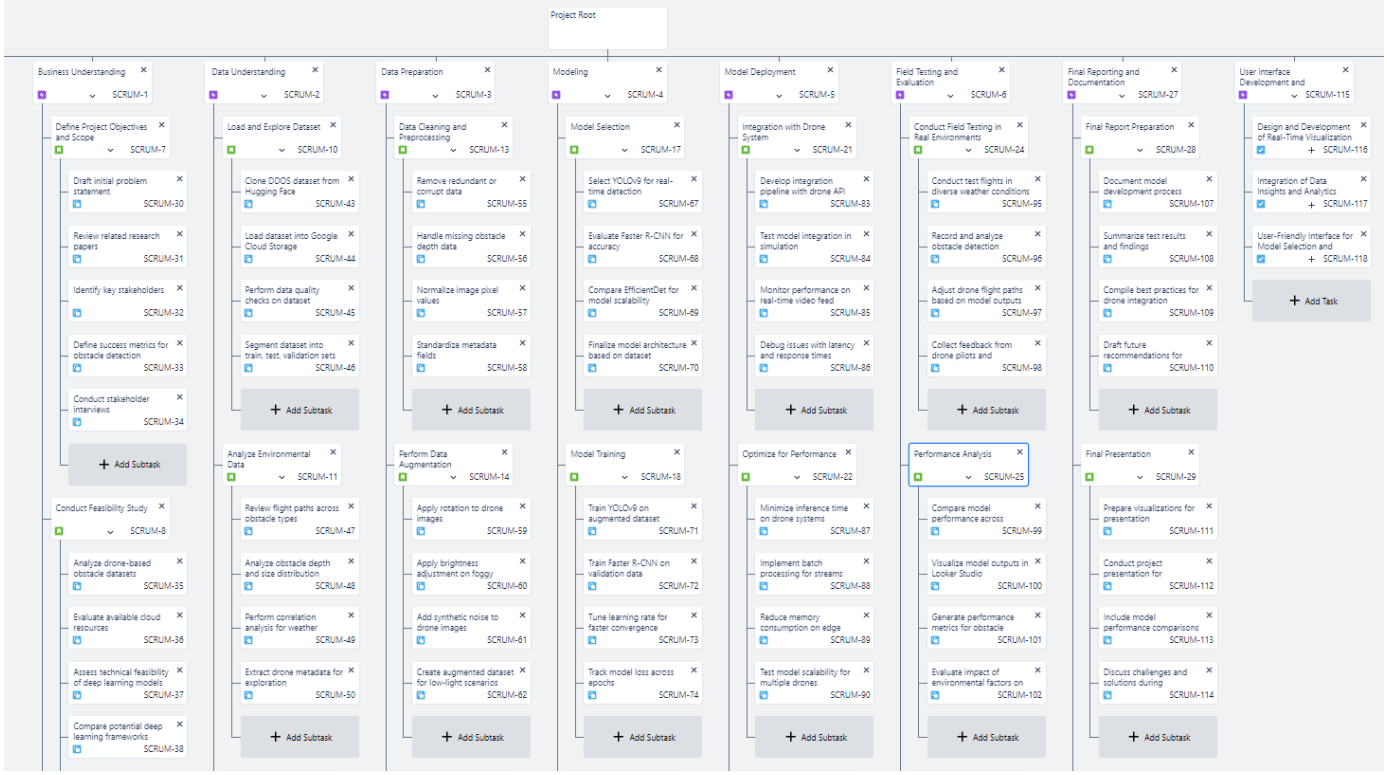
Documentation. All stages in the development process, starting with data handling and pre-processing to model setup, will be well-documented. In addition, complete user manuals will be provided to ensure smooth deployment and operation of the obstacle detection system with drone operations. The technical aspects of this documentation include model architecture, training procedures, and system configuration necessary for the project to be appropriately documented in the event of further updates or improvements.

Project Organization Plan

The Work Breakdown Structure (WBS) is a project management approach for understanding the project's flow by breaking it down into smaller, more manageable portions. Figure 1 displays the project's work breakdown structure (WBS) created using Jira.

Figure 1

Work Breakdown Structure (WBS)



Note. The above image depicts the work breakdown structure of the project

Project Resource Requirements and Plan

Hardware Requirements

Since the backbone of all deep learning approaches lies in the fact that large-scale processing of data, model training, and testing should have a strong hardware setup, this project presents some important hardware configurations. Careful attention has been paid to choosing each component so that the project goes smoothly from data collection down to the deployment of the model. These will range from hosting large datasets, and high-performance

model training capability, all the way to seamless integration of models on either simulated or real-world testing environments for the drones. The hardware requirements for this project are shown in Table 5.

Table 5

Hardware Requirements

Machine	Hardware	Memory	Usage
Lenevo LOQ (15" AMD) with RTX 4050	AMD Ryzen 7 7840HS w/ Radeon 780M Graphics 3.80 GHz, NVIDIA GeForce RTX 4050	RAM: 16 GB SSD: 1 TB	Data collection, storage, and preprocessing
SJSU Computer Lab		RAM: 120 GB SSD: 3 TB	Model training and testing
Macbook Pro M3	11 Core CPU, 14 Core GPU, 16 Core Neural Engine	Ram: 18 GB, SSD: 1 TB	Data preprocessing and light model development

Note. The above table illustrates the Hardware Requirements

Software Requirements

Many libraries, frameworks, and APIs will be used for data collection, model training and testing, and visualization. Only one cloud-coding environment is chosen to facilitate scalability and collaboration; TensorFlow and PyTorch are chosen as particular libraries for deep learning. Confirm that the software stack of the project will work with cloud and hardware infrastructure. Table 6 gives details about the software requirements.

Table 6

Software Requirements

Environment/ APIs/ Libraries/ Packages.	Usage	Version
Python	To develop the code script for the model	3.10
Pandas	To perform data manipulation and pre-processing tasks	1.3.5
NumPy	To perform numeric calculations	1.26.4
TensorFlow	To develop the deep learning models	2.15.0
PyTorch	To develop the deep learning models	2.0
Matplotlib and Seaborn	To visualize the data	3.8.3
BigQuery API	For data querying and analysis	-
Google Cloud SDK	For cloud resource management	-

Note. The above table illustrates the Software Requirements

Tools and Licenses Used

This section covers some of the licenses and tools required to implement the project successfully. The selection of tools has been made based on their scalability, security features, and handling of volumes of data. They ensure that code execution, communication, version control, and project management are effectively carried out. Table 7 presents the different tools and licenses that were put to use to implement the projects.

Table 7

Tools and Licenses Used

Tools	Usage	License
Jupyter Notebook	For executing the model in a web-based environment.	7.1
Google Meet	To conduct virtual project meetings.	2024.1.8
JIRA	To manage workflow and organize the tasks.	8.2.6
Grammarly Premium	To check the grammar score, plagiarism and improve documentation	1.2.63
ChatGPT API	To get knowledge about new technicalities and external resources for knowledge and research	GPT 4
GitHub	To maintain code repository and documentation	3.3.10

Note. The table above tools to be used in the project.

Cost and Justification

Several costs are associated with the project's duration. This includes licensing costs for software tools, subscriptions, hardware, cloud services, and APIs. Overall, cost justification should look to the long-term needs of software licensing, virtual collaboration tools, and infrastructure. The summary of costs and their descriptions is given in Table 8.

Table 8

Cost and Justification

Resource	Justification	Duration	Cost
Hardware	To build and deploy the model	1 year	$2500

Google Cloud Services	For storing data and running compute-intensive models	1 year	$1,200
ChatGPT API	For external knowledge source and technical support	6 months	$200
Jira	For project management and task tracking	1 year	$0
Google Meet and Discord	For virtual meetings	1 year	$0
Grammarly Premium	To get a better grammar score.	6 months	$100

Note. The above table illustrates the Cost and Justification

Project Schedule

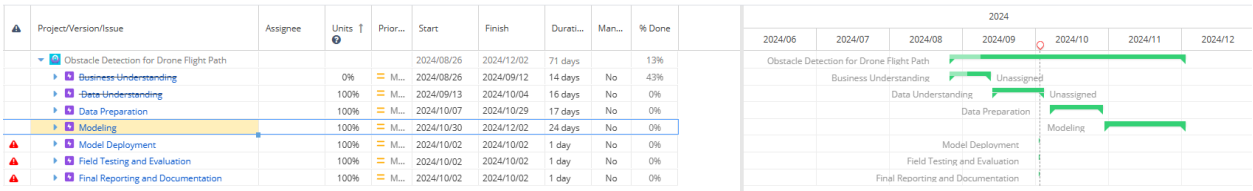
Gantt Chart

In Jira, a Gantt Chart is built, as illustrated in Figure 2, to visually display the project timeline and task list, as well as their dependencies. It meticulously schedules the project and arranges the workflow among all assignees. It displays an Epic hierarchy, followed by User Stories, and then Sub Tasks. In this context, each sub-task was also allocated a start and end date, allowing the project's timetable to be followed and overall growth to be noted.

Link: Gantt-Chart

Figure 2

Gantt Chart with all collapsed hierarchies



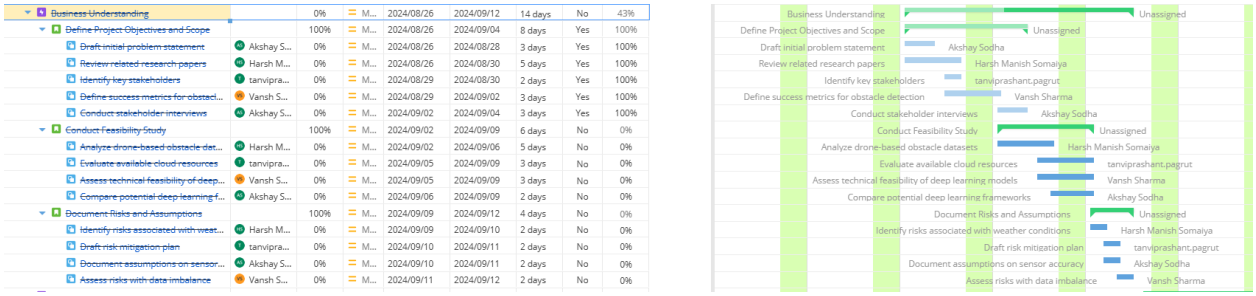
Project/Version/Issue	Assignee	Units	Prior...	Start	Finish	Durati...	Man...	% Done
Obstacle Detection for Drone Flight Path				2024/08/26	2024/12/02	71 days		13%
~~Business Understanding~~		0%	M...	2024/08/26	2024/09/12	14 days	No	43%
~~Data Understanding~~		100%	M...	2024/09/13	2024/10/04	16 days	No	0%
Data Preparation		100%	M...	2024/10/07	2024/10/29	17 days	No	0%
Modeling		100%	M...	2024/10/30	2024/12/02	24 days	No	0%
Model Deployment		100%	M...	2024/10/02	2024/10/02	1 day	No	0%
Field Testing and Evaluation		100%	M...	2024/10/02	2024/10/02	1 day	No	0%
Final Reporting and Documentation		100%	M...	2024/10/02	2024/10/02	1 day	No	0%

Note. The above image shows the epics of the project

First, the team focused on defining the project's goals, objectives, and success metrics during the first epic, Business Understanding, from August 26, 2024, to September 12, 2024. Further, as shown in Figure 3, to ensure that it was timely completed, the various activities of problem statement development, research paper review, stakeholder identification, significant stakeholder interview conduction, and feasibility study conduction were executed parallel to each other.

Figure 3

Gantt Chart for Business Understanding Phase



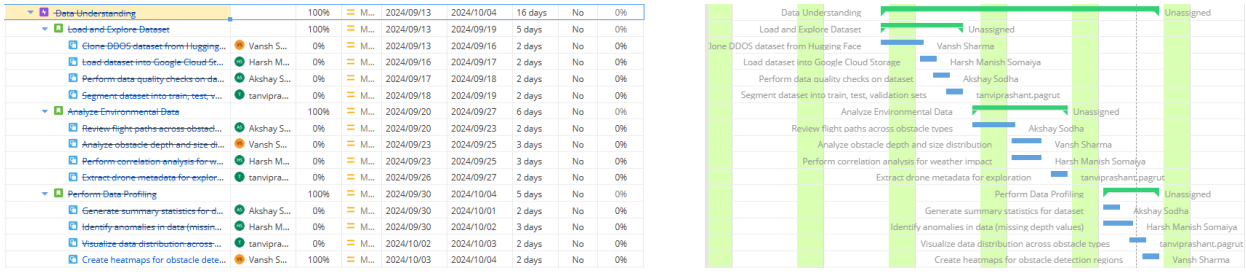
Business Understanding		0%	M...	2024/08/26	2024/09/12	14 days	No	43%
Define Project Objectives and Scope		100%	M...	2024/08/26	2024/09/04	8 days	Yes	100%
Draft initial problem statement	Akshay S...	0%	M...	2024/08/26	2024/08/28	3 days	Yes	100%
Review related research papers	Harsh M...	0%	M...	2024/08/26	2024/08/30	5 days	Yes	100%
Identify key stakeholders	tanviprа...	0%	M...	2024/08/29	2024/08/30	2 days	Yes	100%
Define success metrics for obstacl...	Vansh S...	0%	M...	2024/08/29	2024/09/02	3 days	Yes	100%
Conduct stakeholder interviews	Akshay S...	0%	M...	2024/09/02	2024/09/04	3 days	Yes	100%
Conduct Feasibility Study		100%	M...	2024/09/02	2024/09/09	6 days	No	0%
Analyze drone-based obstacle dat...	Harsh M...	0%	M...	2024/09/02	2024/09/06	5 days	No	0%
Evaluate available cloud resources	tanviprа...	0%	M...	2024/09/05	2024/09/09	3 days	No	0%
Assess technical feasibility of deep...	Vansh S...	0%	M...	2024/09/05	2024/09/09	3 days	No	0%
Compare potential deep learning f...	Akshay S...	0%	M...	2024/09/06	2024/09/09	2 days	No	0%
Document Risks and Assumptions		100%	M...	2024/09/09	2024/09/12	4 days	No	0%
Identify risks associated with weat...	Harsh M...	0%	M...	2024/09/09	2024/09/10	2 days	No	0%
Draft risk mitigation plan	tanviprа...	0%	M...	2024/09/10	2024/09/11	2 days	No	0%
Document assumptions on sensor...	Akshay S...	0%	M...	2024/09/10	2024/09/11	2 days	No	0%
Assess risks with data imbalance	Vansh S...	0%	M...	2024/09/11	2024/09/12	2 days	No	0%

Note. The above image shows the Business Understanding Phase

With this, the team completed this phase with a sound development of understanding the data that would be used in further phases. This project was started on September 13, 2024, and it was completed on October 4, 2024. Following this, the team cloned the Hugging Face DDOS dataset, did quality checks on the data, and divided the data into training, test, and validation sets. Parallel to other data profiling tasks, such as the identification of anomalies in the data and the creation of heatmaps for data visualization, other obstacle depth, and weather analysis were performed, as shown in Figure 4. This phase was of much importance because it gave an insight into what was expected by the team when cleaning and augmenting the data in the upcoming epic.

Figure 4

Gantt Chart for Data Understanding Phase



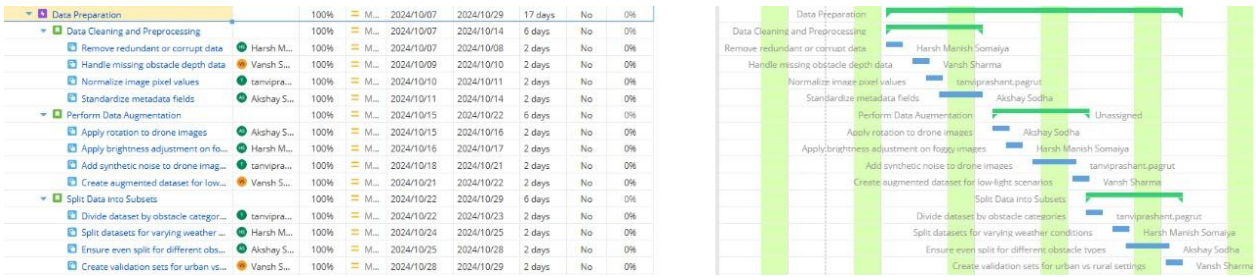
Data Understanding		100%	M...	2024/09/13	2024/10/04	16 days	No	0%
Load and Explore Dataset		100%	M...	2024/09/13	2024/09/19	5 days	No	0%
Clone DDOS dataset from Hugging...	Vansh S...	0%	M...	2024/09/13	2024/09/16	2 days	No	0%
Load dataset into Google Cloud St...	Harsh M...	0%	M...	2024/09/16	2024/09/17	2 days	No	0%
Perform data quality checks on da...	Akshay S...	0%	M...	2024/09/17	2024/09/18	2 days	No	0%
Segment dataset into train, test, v...	tanvipra...	0%	M...	2024/09/18	2024/09/19	2 days	No	0%
Analyze Environmental Data		100%	M...	2024/09/20	2024/09/27	6 days	No	0%
Review flight paths across obstacl...	Akshay S...	0%	M...	2024/09/20	2024/09/23	2 days	No	0%
Analyze obstacle depth and size di...	Vansh S...	0%	M...	2024/09/23	2024/09/25	3 days	No	0%
Perform correlation analysis for w...	Harsh M...	0%	M...	2024/09/23	2024/09/25	3 days	No	0%
Extract drone metadata for explor...	tanvipra...	0%	M...	2024/09/26	2024/09/27	2 days	No	0%
Perform Data Profiling		100%	M...	2024/09/30	2024/10/04	5 days	No	0%
Generate summary statistics for d...	Akshay S...	0%	M...	2024/09/30	2024/10/01	2 days	No	0%
Identify anomalies in data (missin...	Harsh M...	0%	M...	2024/09/30	2024/10/02	3 days	No	0%
Visualize data distribution across ...	tanvipra...	0%	M...	2024/10/02	2024/10/03	2 days	No	0%
Create heatmaps for obstacle dete...	Vansh S...	100%	M...	2024/10/03	2024/10/04	2 days	No	0%

Note. The above image shows the Data Understanding Phase

The data cleaning (Dasari, D., and Varma, P. S, 2022) and augmentation for the dataset began on October 7, 2024, and lasted until October 29, 2024. Figure 5 depicts the handling of missing data, normalizing image pixel values, and rotation, brightness adjustment, and noise addition done to enhance the quality of the data. Then the dataset was split into subsets based on obstacle categories, weather conditions, and environments, preparing it for model training in the successive modeling phase.

Figure 5

Gantt Chart for Data Preparation Phase



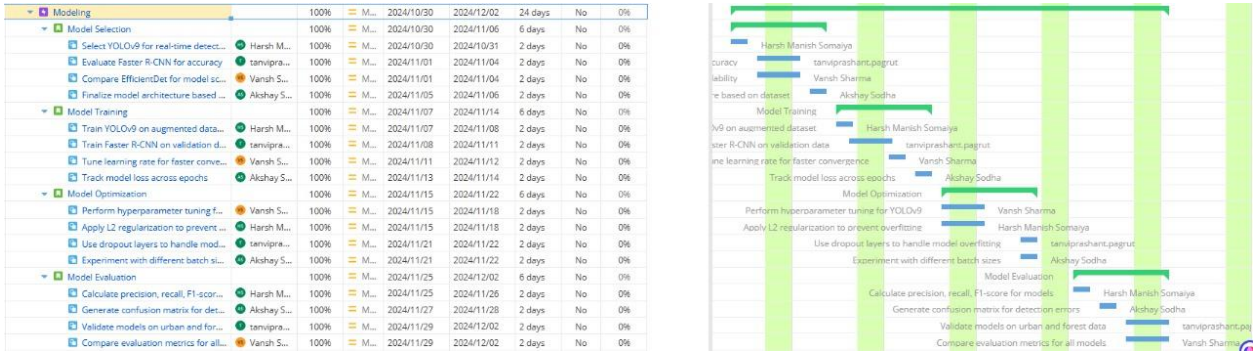
Data Preparation		100%	M...	2024/10/07	2024/10/29	17 days	No	0%
Data Cleaning and Preprocessing		100%	M...	2024/10/07	2024/10/14	6 days	No	0%
Remove redundant or corrupt data	Harsh M...	100%	M...	2024/10/07	2024/10/08	2 days	No	0%
Handle missing obstacle depth data	Vansh S...	100%	M...	2024/10/09	2024/10/10	2 days	No	0%
Normalize image pixel values	tanvipra...	100%	M...	2024/10/10	2024/10/11	2 days	No	0%
Standardize metadata fields	Akshay S...	100%	M...	2024/10/11	2024/10/14	2 days	No	0%
Perform Data Augmentation		100%	M...	2024/10/15	2024/10/22	6 days	No	0%
Apply rotation to drone images	Akshay S...	100%	M...	2024/10/15	2024/10/16	2 days	No	0%
Apply brightness adjustment on fo...	Harsh M...	100%	M...	2024/10/16	2024/10/17	2 days	No	0%
Add synthetic noise to drone imag...	tanvipra...	100%	M...	2024/10/18	2024/10/21	2 days	No	0%
Create augmented dataset for low...	Vansh S...	100%	M...	2024/10/21	2024/10/22	2 days	No	0%
Split Data into Subsets		100%	M...	2024/10/22	2024/10/29	6 days	No	0%
Divide dataset by obstacle categor...	tanvipra...	100%	M...	2024/10/22	2024/10/23	2 days	No	0%
Split datasets for varying weather ...	Harsh M...	100%	M...	2024/10/24	2024/10/25	2 days	No	0%
Ensure even split for different obs...	Akshay S...	100%	M...	2024/10/25	2024/10/28	2 days	No	0%
Create validation sets for urban vs...	Vansh S...	100%	M...	2024/10/28	2024/10/29	2 days	No	0%

Note. The above image shows the Data Preparation Phase

The modeling phase took place from October 30th, 2024, to December 2nd, 2024. In this phase, the research team selected, trained, optimized, and evaluated models, including YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. Following model selection, some of the tasks include applying dropout, adjusting the hyperparameters, and assessing the model using datasets from forests and cities as shown in Figure 6. Precision, recall, and F1 scores were computed as key performance metrics that would ensure the model was accurate and robust enough for integration into the drone system.

Figure 6

Gantt Chart for Modeling Phase



Modeling		100%	M...	2024/10/30	2024/12/02	24 days	No	0%
Model Selection		100%	M...	2024/10/30	2024/11/06	6 days	No	0%
Select YOLOv9 for real-time detect...	Harsh M...	100%	M...	2024/10/30	2024/10/31	2 days	No	0%
Evaluate Faster R-CNN for accuracy	tanviprа...	100%	M...	2024/11/01	2024/11/04	2 days	No	0%
Compare EfficientDet for model sc...	Vansh S...	100%	M...	2024/11/01	2024/11/04	2 days	No	0%
Finalize model architecture based ...	Akshay S...	100%	M...	2024/11/05	2024/11/06	2 days	No	0%
Model Training		100%	M...	2024/11/07	2024/11/14	6 days	No	0%
Train YOLOv9 on augmented data...	Harsh M...	100%	M...	2024/11/07	2024/11/08	2 days	No	0%
Train Faster R-CNN on validation d...	tanviprа...	100%	M...	2024/11/08	2024/11/11	2 days	No	0%
Tune learning rate for faster conve...	Vansh S...	100%	M...	2024/11/11	2024/11/12	2 days	No	0%
Track model loss across epochs	Akshay S...	100%	M...	2024/11/13	2024/11/14	2 days	No	0%
Model Optimization		100%	M...	2024/11/15	2024/11/22	6 days	No	0%
Perform hyperparameter tuning f...	Vansh S...	100%	M...	2024/11/15	2024/11/18	2 days	No	0%
Apply L2 regularization to prevent ...	Harsh M...	100%	M...	2024/11/15	2024/11/18	2 days	No	0%
Use dropout layers to handle mod...	tanviprа...	100%	M...	2024/11/21	2024/11/22	2 days	No	0%
Experiment with different batch si...	Akshay S...	100%	M...	2024/11/21	2024/11/22	2 days	No	0%
Model Evaluation		100%	M...	2024/11/25	2024/12/02	6 days	No	0%
Calculate precision, recall, F1-scor...	Harsh M...	100%	M...	2024/11/25	2024/11/26	2 days	No	0%
Generate confusion matrix for det...	Akshay S...	100%	M...	2024/11/27	2024/11/28	2 days	No	0%
Validate models on urban and for...	tanviprа...	100%	M...	2024/11/29	2024/12/02	2 days	No	0%
Compare evaluation metrics for all...	Vansh S...	100%	M...	2024/11/29	2024/12/02	2 days	No	0%

Note. The above image shows the Modeling Phase

PERT Chart

The PERT chart is one of the most powerful tools in project management, which helps in the visualization of tasks based on their flow, duration, and start and finish dates. It will be easy to provide a mapping of activities regarding the order and interdependencies since tasks are positioned in nodes and arrows. As such, it will be easy for the chart to ensure that the flow of the project is not ambiguous. Therefore, a PERT chart is a master guide that provides a summary of the project's timeline and enables highly effective planning and execution. This visualization makes development easier in the context of our obstacle detection in drone flight paths by enabling meticulous monitoring of every stage of development. Project discussions, final deployment, methodical progress monitoring, and sequential resource allocation are all included in the activities.

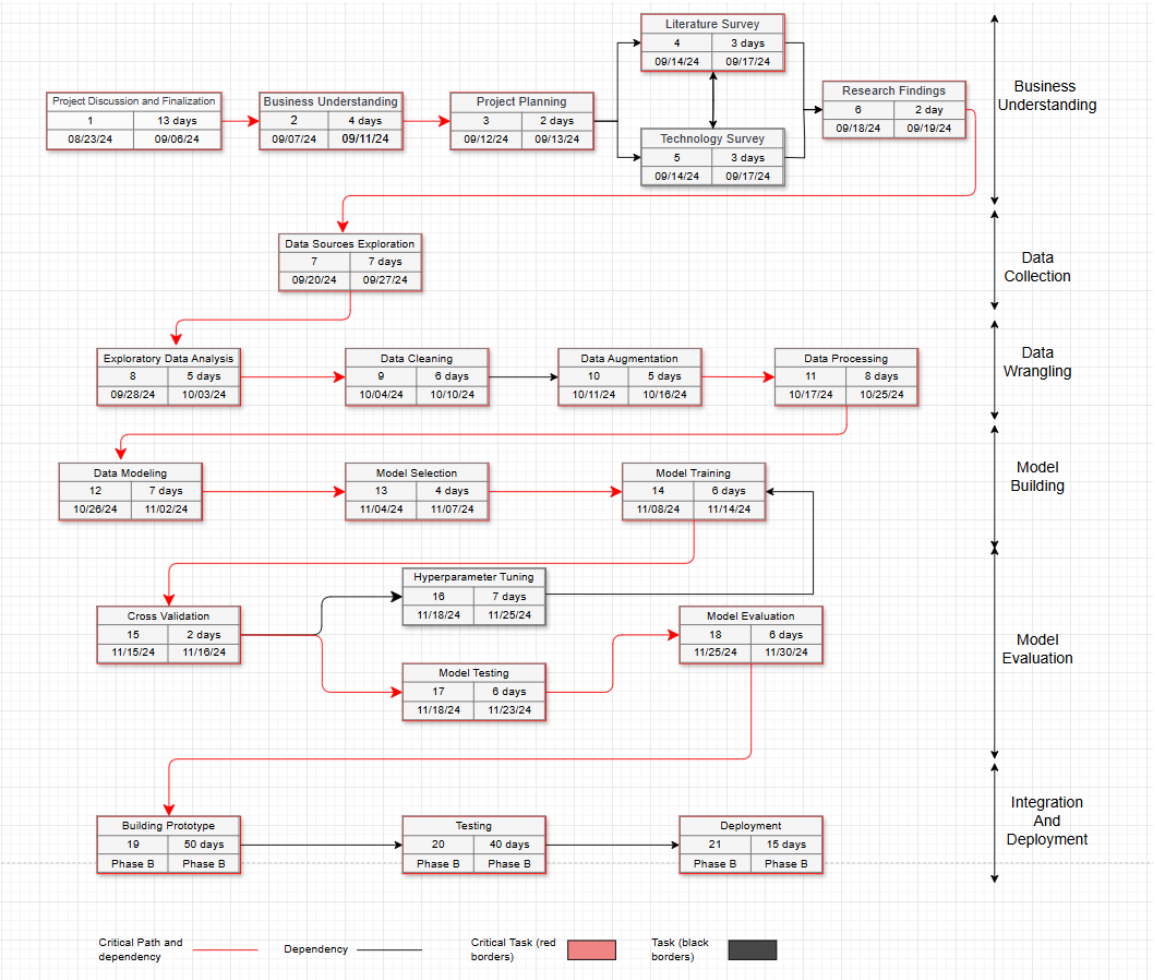
Critical path analysis (CPA) highlights the jobs that are essential to finishing the project as quickly as possible; this is called the critical path. If there is to be no slippage on the project's completion date the red-outlined tasks in Figure 7 are the most important ones that must be completed on time. By highlighting the connections between tasks, the ACTIVITY-based PERT chart facilitates better dependency management and prioritization. Any project can be carried out smoothly when time estimates are determined in such a

systematic manner, ensuring that all tasks are finished by the deadline. This proves how such a strategy ensures that the important tasks are finished in time, allowing flexibility in non-essential areas and reducing delays to ensure the project proceeds smoothly to successful completion

The Critical Path of the project includes Project Discussion & Finalization → Business Understanding → Project Planning → Literature Survey → Research Findings → Data Sources and Exploration → Exploratory Data Analysis → Data Cleaning → Data Augmentation → Data Processing → Data Modeling →Model Selection → Model Training → Cross Validation → Model Testing → Model evaluation → Prototype Building → Testing → Deployment. The Critical Path takes 86 days to complete.

Figure 7

PERT Chart of Obstacle Detection for Drone Flight Path



Data Engineering

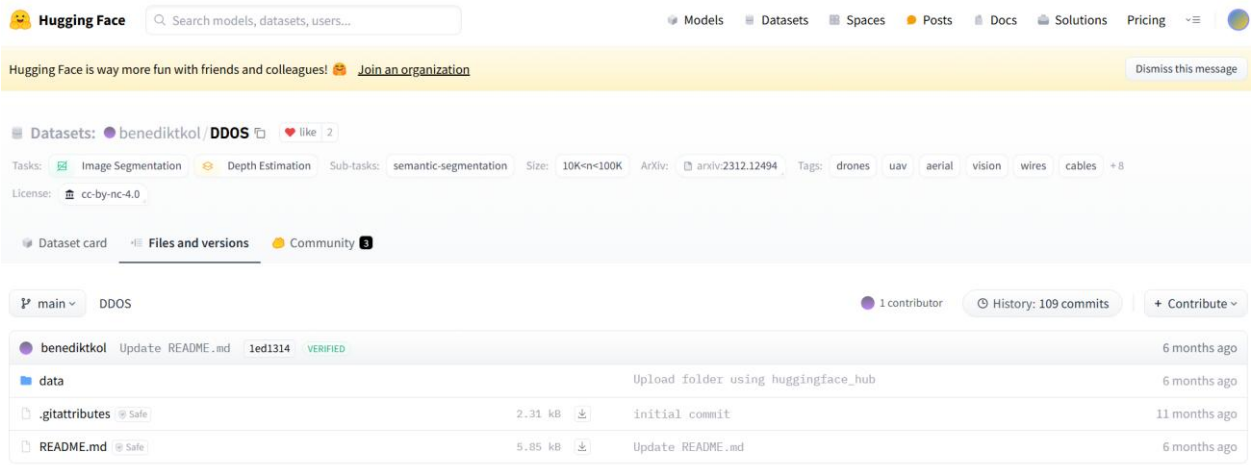
Data Process

Data processing involves the systemic and orderly collection of raw datasets, which need to be preprocessed before being used to train, validate, and test in machine learning. The major data was employed from the DDOS dataset from Hugging Face. This dataset has been designed especially for obstacle detection in drone flight path planning and thus is an ideal fit for promoting drone autonomy. The DDOS dataset provides full coverage of the environment, weather conditions, and obstacle types that drones would experience in the real world. Unlike other datasets, DDOS focuses on identifying thin and ultra-thin objects like wires and cables, which are very important to safe drone navigation but have normally been excluded in benchmark datasets.

Apart from classes of trees, buildings, and cars, DDOS is especially useful for power lines, fences, and thin poles, and difficult to identify obstacles that are critical for precise obstacle segmentation. The DDOS dataset supplied detailed annotations and information on those structures, hence a great site to train algorithms for accurate obstacle detection and flight route planning. Moreover, it considers multiple sequences of variable weather conditions. This makes the test more realistic since a real drone might experience foggy or bad weather conditions (Kolbeinsson & Mikolajczyk, 2023). Figure 8 describes the source of the DDOS data that we downloaded from the Hugging Face website.

Figure 8

DDOS Dataset



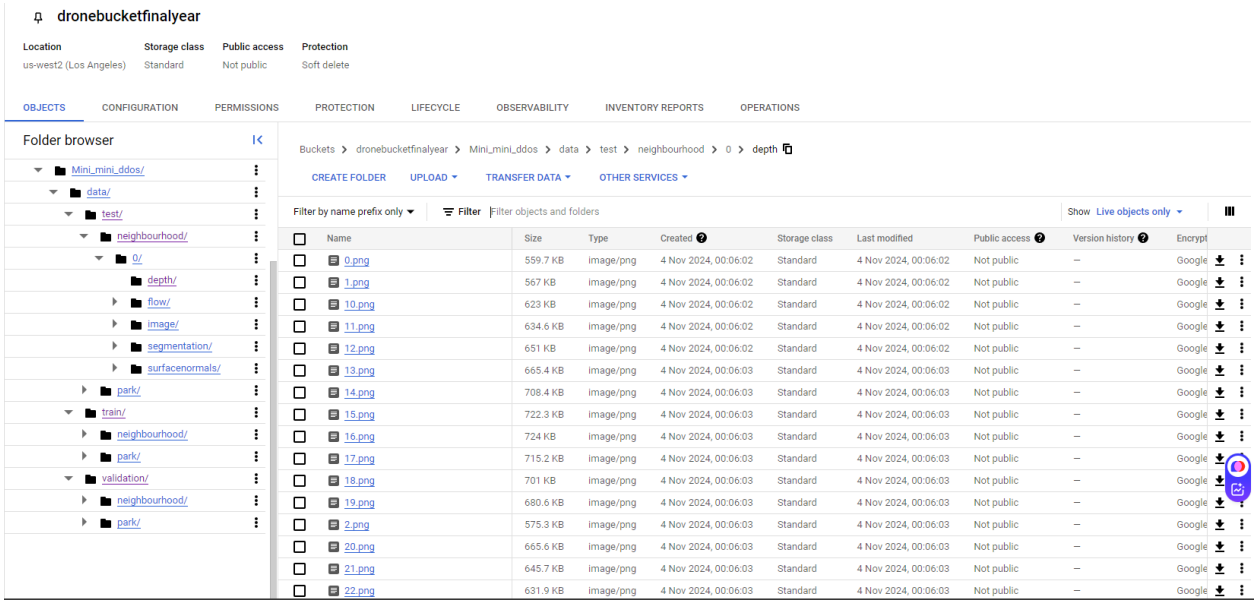
Note. The above image shows the DDOS dataset that is used in the project.

The project needed a method that would guarantee a complete download without interruptions because the dataset is large and it had to be locally accessible for the jobs at all times. The reliance on the rate limit of the API was eliminated because the dataset was cloned straight from the Hugging Face source.

Instead of repeatedly calling the Hugging Face API, the repository was cloned so that the dataset files could be obtained all at once. As a result, the dataset may also be stored on a dedicated 2 TB external hard drive. As a result, this configuration guaranteed a robust offline environment without the possibility of network-related disruptions or API limitations when processing the dataset. The data was loaded to Google Cloud storage with the help of a Python script as seen in Figure 9.

Figure 9

DDOS Dataset downloaded and stored in the Google Cloud Storage



Note. The above image shows that the DDOS dataset was downloaded and stored in the Google Cloud Storage system.

Collection and Structuring of Raw Data

Drones are used to gather unprocessed data in two distinct locations: natural parks and residential districts. There are hurdles in any situation, but the most typical ones are probably the thick trees with big obstructions in parks and the narrow buildings seen in residential neighborhoods.

The open-source drone simulator AirSim was used to synthesize this DDOS dataset in a way that was realistic of real-world circumstances. Simulating intricate drone routes in two distinct environments was done using AirSim. The two environments include are as shown below.

Suburban Town Environment. Power lines, residential structures, and large trees fall within this category. The difficulty of flying in residential areas with potential obstacles everywhere is recreated.

Park Environment. This environment allows for a variety of obstacle kinds and navigation obstacles since it comprises office buildings, park environs, sporting facilities including volleyball courts and football fields, and dense tree cover.

As shown in Figure 10, the data is then meticulously separated into three primary splits: training, validation, and testing. This split is intended to be utilized for efficient model training and tweaking to ensure optimal performance and equitable model assessment.

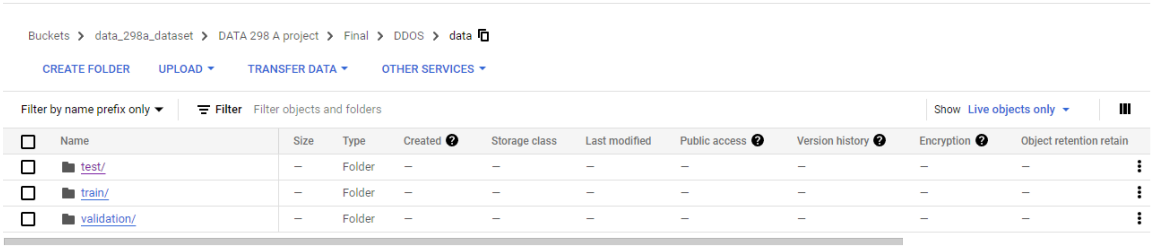
Training Set. The training set, which includes 30,000 photos from 300 drone flights, teaches models how to identify obstructions and depth relationships in a variety of situations. To ensure that the model performs effectively when applied to unknown data, the set includes the majority of possible outcomes.

Validation Set. This set is usually used to fine-tune the models during training, and it has 2,000 pictures and 20 flights. By providing a benchmark for the performance of intermediate models, this validation set aids in hyperparameter adjustment and prevents overfitting.

Testing Set. This collection, which includes 2,000 photos and 20 flights, is used for the model's last training test. This will guarantee an objective test of the model's performance on fresh, untested data in a real-world setting.

Figure 10

DDOS Dataset split into training, testing, and validation sets in Google Cloud Storage system



Note. The above image shows the DDOS dataset split into training, testing, and validation sets.

Data Collection

Git was used to clone the DDOS dataset straight from the Hugging Face repository because their rate limit policy made downloading via API hard. As a result, the dataset is in the appropriate directory to ensure its availability. As seen in Figure 11, it takes up around 253 GB of storage space and 254 GB of disk space. It has 3,40,712 files spread over 62,980 folders, making it an extremely large dataset appropriate for testing and training machine learning models on drone navigation and obstacle recognition (Hinton, J., & Brown, S., 2019). It had approximately thirty-four thousand images in the dataset.

Figure 11

Properties and size of DDOS Dataset stored in the GCS

```
Size: 253 GB (2,72,52,05,00,588 bytes)
Size on disk: 254 GB (2,73,21,74,78,656 bytes)
Contains: 3,40,712 Files, 62,980 Folders
```

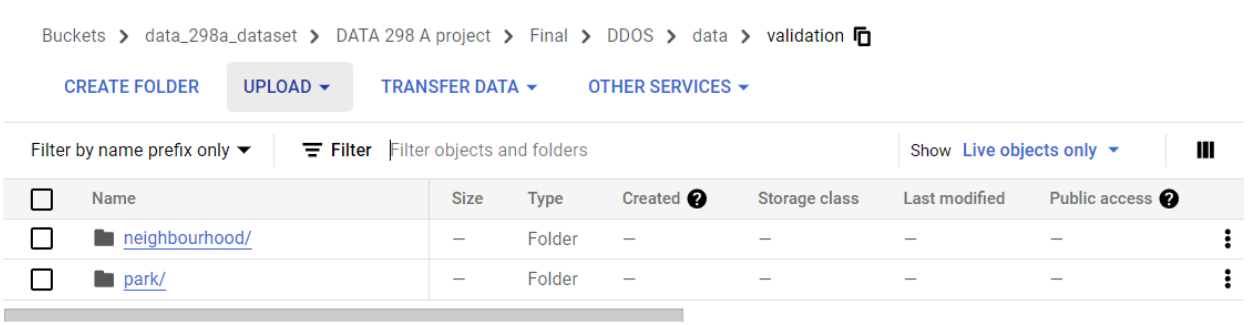
Note. The above image shows the size of the DDOS dataset.

The DDOS dataset is put up to offer a complete and well-structured collection of data for machine learning models tailored to autonomous drone navigation and obstacle detection to be trained, validated, and tested. Since it is well-organized, structured data can be easily accessed and managed. The training set, which includes 30,000 photos from 300 flights, enhances the model via thorough training. Twenty flights totaling 2,000 photos are included in the testing set for evaluating the model's performance. 2,000 images from 20 flights make up the validation set, which is used to fine-tune and alter settings (Jiang, W., & Lu, X., 2019). In total, 34000 images were collected from this dataset.

Figure 12 presents mainly two types of environments: parks and neighborhoods which are presented below in further detail. The park environments contain greenery, such as trees, animals, and openness, while the neighborhood ones comprise urban features such as building wires, cars, etc.

Figure 12

Neighbourhood and Park environments



Note. The above image shows the folder in which neighborhood and park environments are stored in GCS.

Each flight in the dataset is uniquely identifiable by its flight ID and is arranged in directories according to the kind of setting, such as a park or neighborhood. Effective access to data for certain situations or scenarios is made possible by this hierarchical structure.

A wide range of data modalities, including the following, are provided for every trip via DDOS data. By doing so, the dataset becomes more appropriate for various tasks, such as motion analysis, segmentation, and depth estimation. As seen in Figure 13, the raw datasets contain several elements that are essential for training the model.

Images

Figure 14 illustrates the sample image from the image folder in the dataset which describes the visual information required for object detection and segmentation provided by the high-resolution RGB pictures captured by the drone cameras. When training a convolutional neural network, images are essential inputs. The folder where each flight's photographs are kept is shown in Figure 15.

Depth Maps

As seen in Figure 16, these maps use uint16 PNG images to measure the distance between objects and the drone. The pixel values fall between 0 and 65,535, which is equivalent to a distance in meters between 0 and 100. One of the most crucial pieces of information for comprehending any picture is depth information, which aids in determining the relative positions of various objects within the surrounding area. It is used for applications like depth estimation, among others (Pande, B. et al., 2022). The depth photos for every flight are shown in Figure 17.

Segmentation Masks

Figure 18 shows a sample image in which pixel-wise segmentation masks representing distinct obstacle types are performed. Masks employ a class mapping approach to distinguish between a variety of barriers, including thin structures, buildings, and automobiles. Figure 19 depicts the various segmentation-masked pictures for a single flight

path in GCS. This information will be important in segmenting or differentiating different types of barriers for the model to train on.

Optical Flow

It is the apparent mobility of objects between consecutive frames and is utilized to monitor moving barriers and comprehend dynamic situations, as seen in Figure 20. Figure 21 depicts the optical flow images for a trip.

Surface Normal Maps

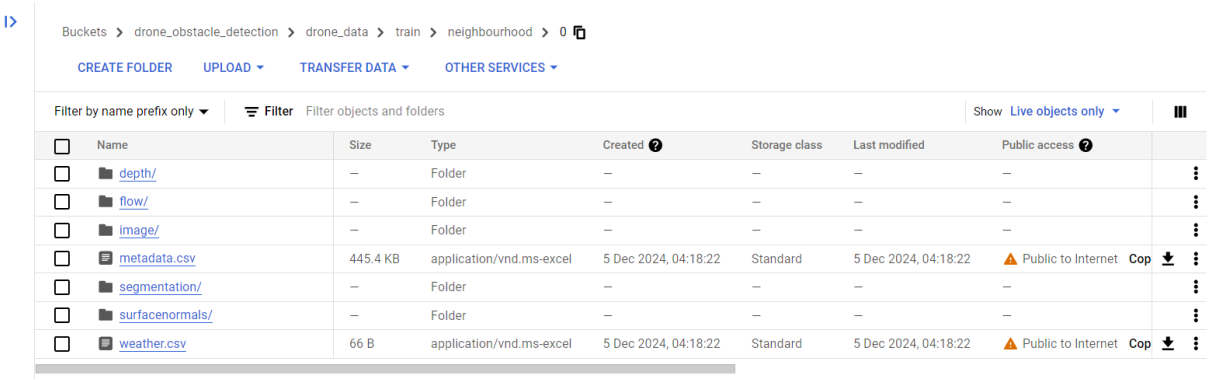
These maps define the orientation of the object surfaces, assisting the model in understanding the geometry and form of barriers in the environment, as seen in Figure 22. Figure 23 illustrates the example surface normal images saved in GCS for a flight.

Metadata

Each flight has an associated metadata file that stores the coordinates and drone pose, acceleration, weather conditions, and camera parameters. All information becomes imperative as contextual data to understand the conditions under which these images were captured.

Figure 13

Data of each flight divided into different categories



Note. The above image shows the folder in which data for each flight is stored in GCS.

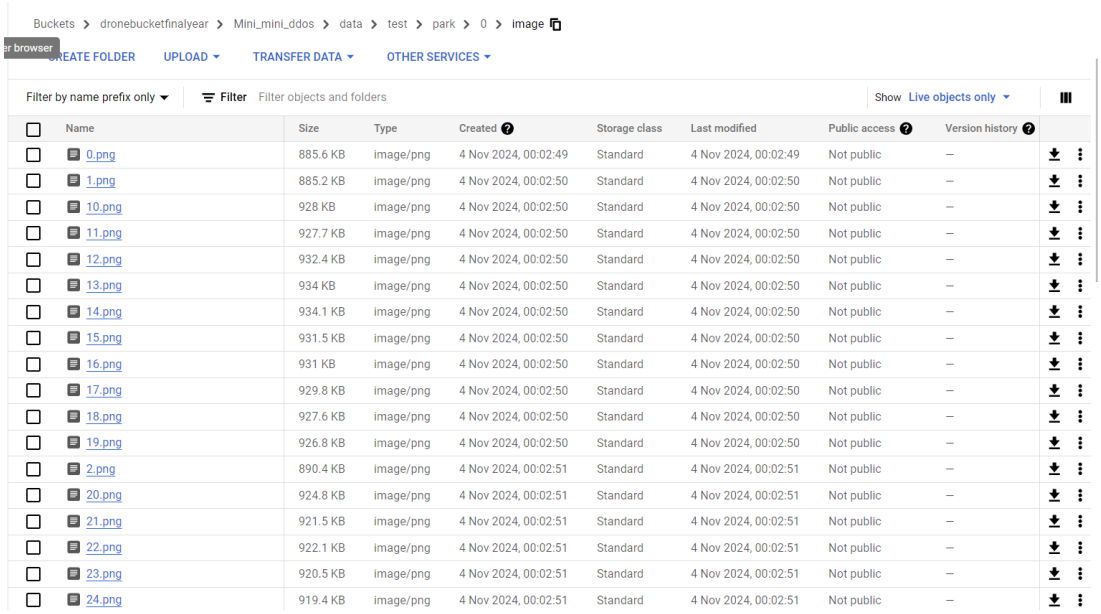
Figure 14

Sample Image data

Note. The above image shows the sample image data of a flight.

Figure 15

Image folder for each flight



Buckets > dronebucketfinalyear > Mini_mini_ddos > data > test > park > 0 > image

er browser

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
0.png	885.6 KB	image/png	4 Nov 2024, 00:02:49	Standard	4 Nov 2024, 00:02:49	Not public	–
1.png	885.2 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
10.png	928 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
11.png	927.7 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
12.png	932.4 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
13.png	934 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
14.png	934.1 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
15.png	931.5 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
16.png	931 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
17.png	929.8 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
18.png	927.6 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
19.png	926.8 KB	image/png	4 Nov 2024, 00:02:50	Standard	4 Nov 2024, 00:02:50	Not public	–
2.png	890.4 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–
20.png	924.8 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–
21.png	921.5 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–
22.png	922.1 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–
23.png	920.5 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–
24.png	919.4 KB	image/png	4 Nov 2024, 00:02:51	Standard	4 Nov 2024, 00:02:51	Not public	–

Note. The above image shows the folder in which image data for each flight is stored in GCS.

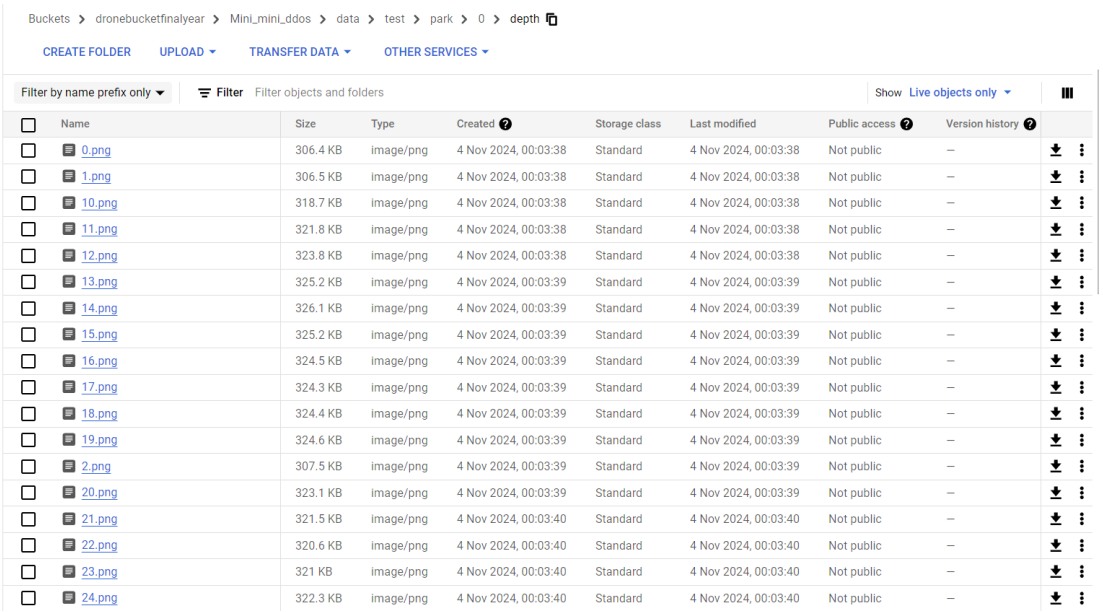
Figure 16

Sample Depth Image data

Note. The above image shows the sample depth data of a flight.

Figure 17

Depth folder for each flight



Buckets > dronebucketfinalyear > Mini_mini_ddos > data > test > park > 0 > depth

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

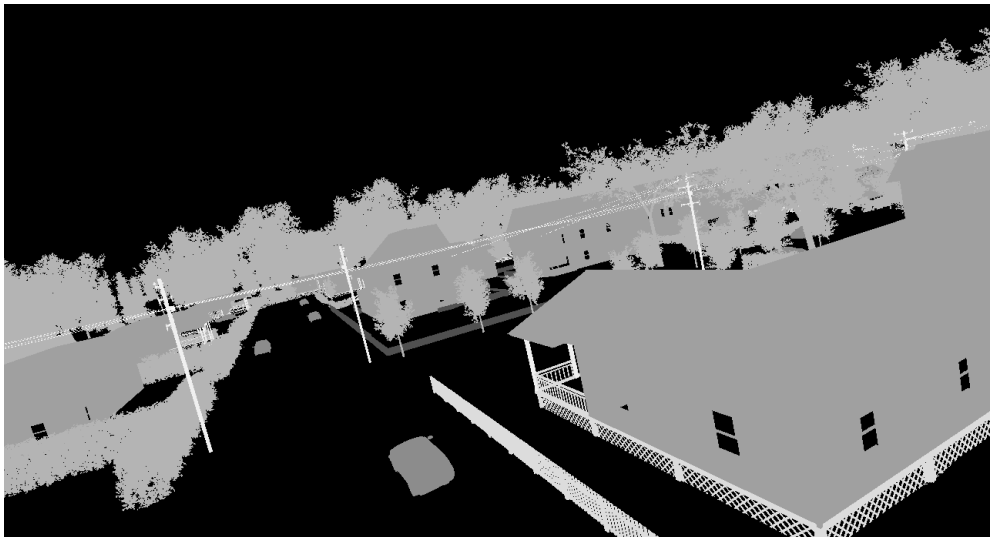
Filter by name prefix only | Filter Filter objects and folders | Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
0.png	306.4 KB	image/png	4 Nov 2024, 00:03:38	Standard	4 Nov 2024, 00:03:38	Not public	–
1.png	306.5 KB	image/png	4 Nov 2024, 00:03:38	Standard	4 Nov 2024, 00:03:38	Not public	–
10.png	318.7 KB	image/png	4 Nov 2024, 00:03:38	Standard	4 Nov 2024, 00:03:38	Not public	–
11.png	321.8 KB	image/png	4 Nov 2024, 00:03:38	Standard	4 Nov 2024, 00:03:38	Not public	–
12.png	323.8 KB	image/png	4 Nov 2024, 00:03:38	Standard	4 Nov 2024, 00:03:38	Not public	–
13.png	325.2 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
14.png	326.1 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
15.png	325.2 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
16.png	324.5 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
17.png	324.3 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
18.png	324.4 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
19.png	324.6 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
2.png	307.5 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
20.png	323.1 KB	image/png	4 Nov 2024, 00:03:39	Standard	4 Nov 2024, 00:03:39	Not public	–
21.png	321.5 KB	image/png	4 Nov 2024, 00:03:40	Standard	4 Nov 2024, 00:03:40	Not public	–
22.png	320.6 KB	image/png	4 Nov 2024, 00:03:40	Standard	4 Nov 2024, 00:03:40	Not public	–
23.png	321 KB	image/png	4 Nov 2024, 00:03:40	Standard	4 Nov 2024, 00:03:40	Not public	–
24.png	322.3 KB	image/png	4 Nov 2024, 00:03:40	Standard	4 Nov 2024, 00:03:40	Not public	–

Note. The above image shows the folder in which depth data for a flight is stored in GCS.

Figure 18

Segmentation Masked Image data



Note. The above image shows the sample segmentation masked data of a flight.

Figure 19

Segmentation folder for each flight

Buckets > dronebucketfinalyear > Mini_mini_ddos > data > test > park > 0 > segmentation

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
0.png	21.4 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
1.png	21.5 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
10.png	22.1 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
11.png	21.9 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
12.png	21.9 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
13.png	21.7 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
14.png	21.4 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
15.png	21.5 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
16.png	21.7 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
17.png	21.8 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
18.png	21.8 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
19.png	22 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
2.png	21.6 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
20.png	22.2 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
21.png	22.2 KB	image/png	4 Nov 2024, 00:02:43	Standard	4 Nov 2024, 00:02:43	Not public	–
22.png	22.1 KB	image/png	4 Nov 2024, 00:02:44	Standard	4 Nov 2024, 00:02:44	Not public	–

Note. The above image shows the folder in which segmentation masked data for each flight.

Figure 20

Flow Image data

Note. The above image shows the sample flow data of a flight.

Figure 21

Flow folder for each flight

Buckets > dronebucketfinalyear > Mini_mini_ddos > data > test > park > 0 > flow

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

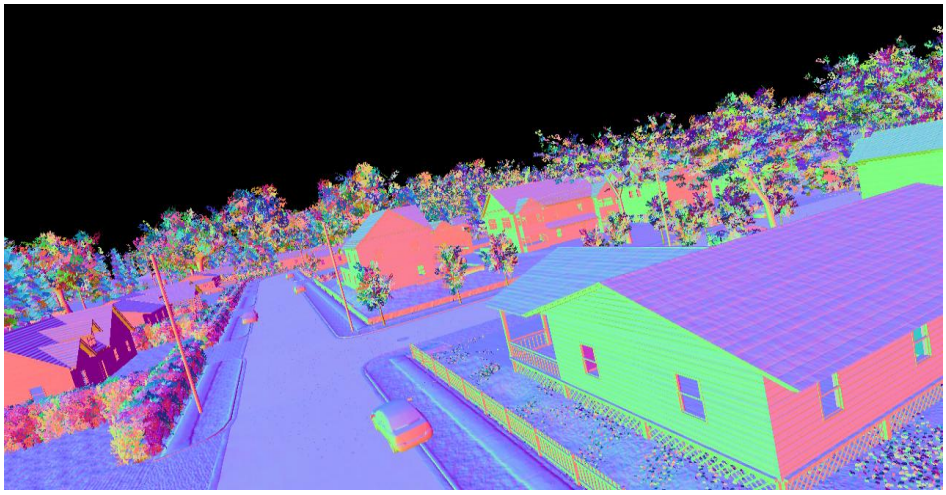
Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
0.png	94.3 KB	image/png	4 Nov 2024, 00:02:59	Standard	4 Nov 2024, 00:02:59	Not public	–
1.png	101.2 KB	image/png	4 Nov 2024, 00:02:59	Standard	4 Nov 2024, 00:02:59	Not public	–
10.png	161.1 KB	image/png	4 Nov 2024, 00:02:59	Standard	4 Nov 2024, 00:02:59	Not public	–
11.png	148.5 KB	image/png	4 Nov 2024, 00:02:59	Standard	4 Nov 2024, 00:02:59	Not public	–
12.png	133.5 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
13.png	129.4 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
14.png	130.2 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
15.png	147.2 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
16.png	148.6 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
17.png	147.4 KB	image/png	4 Nov 2024, 00:03:00	Standard	4 Nov 2024, 00:03:00	Not public	–
18.png	143.6 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–
19.png	142.4 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–
2.png	161.8 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–
20.png	135.3 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–
21.png	142.2 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–
22.png	144.4 KB	image/png	4 Nov 2024, 00:03:01	Standard	4 Nov 2024, 00:03:01	Not public	–

Note. The above image shows the folder in which flow data for each flight.

Figure 22

Surface normal Image data



Note. The above image shows the sample surface normal data of a flight.

Figure 23

Surface normal folder for each flight

Buckets > dronebucketfinalyear > Mini_mini_ddos > data > test > park > 0 > surfacenormals

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
0.png	669.1 KB	image/png	4 Nov 2024, 00:02:32	Standard	4 Nov 2024, 00:02:32	Not public	–
1.png	667.1 KB	image/png	4 Nov 2024, 00:02:32	Standard	4 Nov 2024, 00:02:32	Not public	–
10.png	689 KB	image/png	4 Nov 2024, 00:02:32	Standard	4 Nov 2024, 00:02:32	Not public	–
11.png	692 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
12.png	694.8 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
13.png	695 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
14.png	693.9 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
15.png	692.9 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
16.png	693.1 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
17.png	694.1 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
18.png	695 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
19.png	697.3 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
2.png	666.2 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
20.png	697.6 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
21.png	697.7 KB	image/png	4 Nov 2024, 00:02:33	Standard	4 Nov 2024, 00:02:33	Not public	–
22.png	700.4 KB	image/png	4 Nov 2024, 00:02:34	Standard	4 Nov 2024, 00:02:34	Not public	–

Note. The above image shows the folder containing surface normal image data for each flight.

In addition to photos, depth maps, segmentation, flow, and surface normal data, each flight in the DDOS collection includes metadata.csv and weather.csv files that include precise information about drone parameters and environmental conditions. Here's an overview of each file's contents based on the 9th flight data supplied.

Metadata.csv

The file in Figure 24 provides critical information about the drone's location, orientation, and movement during each recorded frame. This file enables researchers to examine flight dynamics and recreate real-world drone behaviors, resulting in rich contextual data for algorithm training. Important columns include:

Timestamp. This captures the exact time of each frame, allowing for accurate synchronization across data modalities.

Drone Position (x, y, z). The drone's 3D coordinates display its exact location while in flight.

Drone Orientation (w, x, y, z). Quaternion values establish the drone's 3D orientation, which is critical for determining its directional face.

Camera Pose (x, y, z). The camera's location with respect to the drone is useful for perspective computations.

Linear Velocity (x, y, z). The drone's velocity along each axis indicates its movement speed and direction.

Linear Acceleration (x, y, z). The acceleration numbers show how rapidly the drone's speed changes.

Angular Velocity (x, y, z). The rotational speed around each axis indicates how quickly the drone is spinning.

Angular Acceleration (x, y, z). The rate of change of angular velocity is essential for comprehending rotational motions.

Figure 24

Metadata file

PREVIEW

	123 drone_orie	123 drone_orie	123 drone_orie	123 camera_p	123 camera_p	123 camera_p	123 linear_vel	123 linear_vel	123 linear_vel	123 linear_acc	123 linear_acc	123 linear_acc	123 angular_v	123 angular_v	123 angular_
8	-0.2061250061	-0.00808046013	-0.8838824034	-133.4462585	153.9097748	-11.41559219	-1.140495777	-0.3200290501	-0.9031839371	-2.785474777	-1.547499895	2.651776791	-2.542375088	2.399998665	1.549784
7	-0.1759496629	0.06826238334	-0.8791401386	-133.5008087	153.8679504	-11.51187801	-1.295861959	-0.3661121428	-0.7659516931	-2.913666725	-0.1982389987	2.448573589	-1.455849648	2.438046455	1.208880
6	-0.1712698638	0.07612224668	-0.8784626722	-133.5084839	153.8635254	-11.52370834	-1.322035193	-0.3669912219	-0.7440600991	-2.900928259	0.00192009471	2.416027546	-1.257298827	2.448592424	1.151219
3	-0.02713691629	-0.1001393273	-0.9648286104	-43.30193329	138.2822266	-17.46762085	-8.796298027	-2.963681221	0.8268330693	0.4613839388	-1.554380894	1.286223412	-0.3980216682	0.0748809427	-0.3158192
8	0.02402984537	0.000602486543	0.9975328445	69.88542175	88.61496735	-10.12945843	-7.532250404	1.036805034	-0.0505541563	-0.0966770649	-0.03020442836	0.04881191254	-0.00804047100	0.01453807112	0.014999
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
8	0.02326312661	0.000225383555	0.9975891113	69.04869843	88.72938538	-10.13547897	-7.541815758	1.033891559	-0.04563389719	-0.0809134841	-0.02395156398	0.04246520996	-0.00684405490	0.01219328027	0.0059249
1	0.02313286811	0.000161823001	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
1	0.02313286811	0.000161823001	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
4	0.02307836711	0.000135224079	0.9975979328	68.8223877	88.76030731	-10.13699245	-7.544184208	1.033196092	-0.04438394308	-0.07700735331	-0.02242858522	0.04087543488	-0.00656224275	0.0116086863	0.0035405
8	0.[illegible]75165915	-0.00002436281	0.997607708	68.39221191	88.81912994	-10.13974762	-7.548792362	1.031877518	-0.04190891236	-0.06940442324	-0.01949266903	0.03774547577	-0.00600289739	0.01047458872	-0.0009670
4	0.[illegible]4474163	-0.00012548558	0.9976100326	68.09777832	88.85942078	-10.1415472	-7.551022053	1.031257629	-0.04068887979	-0.06572329998	-0.01808215864	0.03620529175	-0.00573356496	0.00992615334	-0.0029884
8	0.02249869332	-0.00014771253	0.9976100922	68.0298233	88.86872101	-10.14195251	-7.551608562	1.031096578	-0.04036485776	-0.06475353241	-0.01771507785	0.03580093384	-0.00562459603	0.00978063885	-0.0034925

Note. The above image shows the metadata file for flight paths.

Weather.csv

This file documents the environmental factors that impact vision and navigation. This weather data is used to replicate real-world environmental problems, ensuring that the dataset is adaptable to a wide range of weather situations while developing obstacle recognition and navigation models. It comprises of the following properties, as seen in Figure 25.

Rain. It indicates the intensity of rain during the flight.

Road Wetness. This describes how wet the road surfaces are, impacting reflectivity and surface texture.

Snow. This specifies snow levels, which can affect visibility and terrain features.

Road Snow. It shows the amount of snow on roads, impacting traction and visual conditions.

Maple Leaf, Dust, Fog. Additional elements that add realism by affecting visibility or scene texture.

Figure 25

Weather data file

123 rain	123 roadwetne	123 snow	123 roadsnow	123 mapleleaf	123 dust	123 fog
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Note. The above image shows the weather file.

Flight Trajectory

Each of these flight routes in the surroundings was designed to add increasing levels of difficulty, such as abrupt turns, low-altitude flights, and near-collision encounters. These trajectories guarantee that the dataset will contain all the relevant training data on complicated flying patterns, which is important to create strong drone navigation algorithms.

Each flight route consists of changes in altitude, speed, pitch, and roll angles to simulate the many situations drones experience in real-world surroundings.

Start and destination points. Every run begins with a distinct starting location inside the area. The route is then defined by generating multiple intermediate target sites.

Collision Avoidance and Near-Misses. A dynamic obstacle recognition system was used to adjust the trajectory in real time based on probable collision hazards. This function allows for safe and realistic near contacts with objects, which is essential in training drones to navigate around them.

Data Collection Rate. Data was acquired from observations made at 10Hz during a 10-second flight, which equaled 100 frames each trip. Overall, the collection contains 340 flights and 34,000 photos.

Annotations

The DDOS dataset offers a wide range of data types, collecting several dimensions of information during each trip. Indeed, such a broad range of annotations is sure to be extremely valuable for training sophisticated models that require a multimodal knowledge of the environment.

RGB Images. The resolution of 1280×720 pixels ensure high-quality graphics with precise details of the surroundings. The perspective would be a monocular camera positioned in front of the drone, similar to the common camera layout in real-world drones.

Depth Maps. The range extends from 0 to 100 meters, providing critical distance data for depth estimate jobs. The format contains depth data recorded in high-resolution format to help in precise obstacle distance measurement, which is critical for obstacle identification and avoidance algorithms.

Pixel-wise Semantic Segmentation Masks. The dataset comprises segmentation masks for 10 separate classes, allowing for accurate item classification and identification. Table 9 shows the following classes which are mentioned below.

Table 9

Data Classes

Class	Description
Ultra Thin	Wires and cables, which are hard to detect but crucial for safety
Thin Structures	Poles, street signs, and other thin objects
Small Mesh	Fences, nets, or other small structures that drones might navigate around
Large Mesh	Transmission towers, pylons, and large open structures
Trees	Trees and foliage, which are common in both suburban and park environments
Buildings	Residential and office buildings
Vehicles	Cars and other vehicles typically found in suburban areas
Animals	Small animals, adding realism and variability
Other	Miscellaneous objects, like bus stops, post boxes, chairs, etc.

Background	Ground, sky, and other non-obstacle elements

Note. The above table shows the type of classes that will be used for data annotations

These segmentation masks are crucial for semantic segmentation tasks, allowing models to distinguish between distinct objects and focus on obstacle-specific classes such as ultra-thin cables (Chen, L. et al., 2018).

Optical Flow and Surface Normals. Optical flow is the apparent mobility of objects between consecutive frames. It is extremely beneficial for following moving objects and forecasting the future location of obstacles. Surface normals show the orientation of surfaces, which helps with 3D comprehension and reconstruction.

Metadata (Flight Parameters). Each frame in this dataset comprises flight characteristics (position, orientation, velocities, and accelerations) that offer context to flight dynamics. This metadata will help us understand how the drone interacts with its environment.

Environmental and Weather Conditions. The DDOS dataset includes demanding environmental circumstances that are required for creating strong algorithms that are unaffected by weather conditions. Therefore, the recording circumstances include the following.

Clear Skies. Standard conditions when visibility is good.

Dusk and Night Scenes. Low-light conditions increase the difficulty in obstacle detection.

Rain and Fog. Weather conditions reduce visibility and introduce noise into the visuals.

Snow and Wet Surfaces. This covers cases with snowy or wet surfaces that become reflective or alter from their normal appearance, making depth estimation and segmentation difficult.

These changes present real-world difficulties that allow the model to adapt to changing situations, which is critical for applications such as emergency response, package delivery, and other autonomous services supplied by drones.

Flight Trajectory Dynamics

DDOS provides a variety of flight trajectories, including pitch and roll degrees, speeds, and altitude. In this approach, the dataset can capture the complexities of real flights while also supporting models that must be stable across a wide range of flight patterns.

Pitch and Roll Angles. The dataset includes a large range of orientations in both roll and pitch variations, which correspond to a wide range of turns, forward accelerations, and brakes.

Altitude Range. Mission flying altitudes range from low-altitude flights near obstacles to higher-altitude flights that provide data for various navigation reasons.

Speed Variation. Variations in speed will occur between flights; hence, it ensures a realistic dataset when considering the flight speed variable, which influences obstacle recognition accuracy.

This variety is critical in developing algorithms capable of managing the diverse and dynamic surroundings encountered by drones.

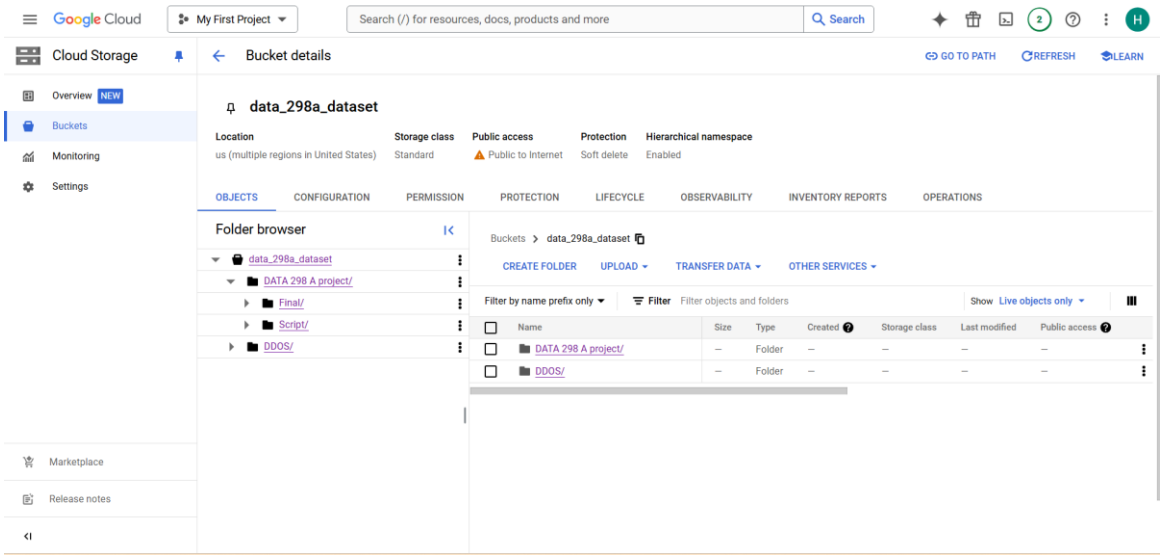
Data Pre-processing

This dataset is huge, 253 GB with more than 340,000 files in almost 63,000 folders. Since the volume of data was huge and very complicated, Google Cloud Storage was used as

the primary storage; it provides scalable storage, as shown in Figure 26, which is globally accessible and affordable while providing high reliability and integration for data processing.

Figure 26

Data stored in Google Cloud storage



Note. The above image shows the data storage in GCS.

To acquire a comprehensive view of the image data in this dataset, three different scripts were written to generate three distinct master metadata files corresponding to the train, test, and validation sets. The metadata and weather data will now be collected and compiled into integrated databases corresponding to the various data splits, which are the final products, in a very organized manner. Each script will crawl over its appropriate data split directory, extracting the necessary metadata and weather files for each flight, and compiling them into a single master metadata file. These are consolidated files that contain all key flight and environmental characteristics, allowing for more efficient dataset analysis and pre-processing.

Each data split (train, test, and validation) required its own master metadata file. This is a key step in structuring and organizing the dataset's information. We ensured that we

captured the full picture of every single flight's detail including geographical, temporal, and environmental aspects by storing the information of each set in a single spot.

First, a local script was written to generate a master metadata file for the training dataset. It iterates through each flight folder, dynamically extracting information and weather columns before aggregating all data into a single CSV file. The script collects each flight's information, meteorological data, and a number of picture file locations for depth, segmentation, flow, and surface normals. It locates and writes items to a centralized CSV, as seen in Figure 27. The master file is well-organized, allowing for easy data access, cleaning, and preparation

Figure 27

Metadata file containing details for every flight

PREVIEW

123 flight_id	Tr dataset	Tr location	123 timestamp	Tr image_file	Tr image_pat	Tr depth_patl	Tr segmentat	Tr flow_path	Tr surface_n	Tr weather_c	123 timestamp	123 img_index	123 drone_pos	123 drone_pos
0	train	neighbourhood	1.70828E+18	12.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	13.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	14.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	15.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	16.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	17.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	18.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	19.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	2.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	20.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	21.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	22.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	23.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	24.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	25.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	dataset/DATA 29	1.70828E+18	0	42.30175018	126.73895

Note. The above image shows a metadata file containing details for every flight.

Data augmentation and preprocessing are among the most important procedures to ensure quality and consistency for any dataset used in machine learning pipelines, including drone-based obstacle detection projects. These steps help in standardizing the input data and improving model performance while guaranteeing robustness against different real-world conditions. The following steps describe the augmentation and preprocessing techniques

employed in this project, detailing their importance and giving insight into the transformations applied to the dataset. The data augmentation and pre-processing of image data were also performed which included the following steps.

Augmentation Techniques

Augmentation is such a critical step toward building diversity in the datasets- mimicking variations encountered naturally. It ensures that the model will generalize better to unseen conditions, considering the changes in environmental and operating factors. Figure 28 shows the sample image before performing augmentations.

Figure 28

Orginal Image before performing augmentation techniques

Note. The above image shows a sample image before performing augmentation.

Horizontal flipping was then done on these images with a 50% chance, hence introducing mirrored perspectives; random rotation within ±45° varied the angular diversity of the dataset. To simulate various lighting conditions, brightness and contrast changes were also applied, where ±30% randomness in brightness and ±0.5 random changes, in contrast, ensured real-world illumination variability within this dataset. Gaussian blur was then applied

to simulate motion blur with a kernel size of (3, 3), with variance from 0.1 to 0.5 to simulate realistic scenarios of drone movement. Finally, sensor inaccuracies and environmental disruptions were simulated by adding Gaussian noise with a standard deviation in the range of 0.01 to 0.05. Taken together, all these augmentation techniques not only diversified the dataset but ensured standard resolution at 224x224 pixels to enable good generalization of the model over varying conditions,as seen in Figure 29, which shows the image after performing all the augmentations on the image.

Figure 29

Original Image after performing augmentation techniques

Note. The above image shows a sample image after performing augmentation techniques.

Preprocessing Techniques

The preprocessed data are those which have been transformed from their raw state into standardized forms; thus, they are in ideal consumption by the model. Preprocessing ensures that the data are consistent in input, feature-rich, and meet the requirements of the model, enhancing overall quality and informativeness.

Resize and normalize. Images were then resized to a fixed resolution of 224x224 pixels, so all images would be uniformly resolved within the dataset. Then, pixel values were normalized by dividing them by 255, scaling them between 0 and 1. Originally, images in the dataset ranged in resolutions from 1024x768 to 1920x1080 as seen in Figure 30, which created some problems with the density of pixels and the aspect ratio.

Figure 30

Original Image before performing resize and normalizing techniques

Note. The above image shows a sample image before performing resizing and normalizing.

These images were all resized to 224x224 pixels after preprocessing and further normalized in the range of [0, 1] to make them apt for deep learning models as in Figure 31.

Figure 31

Original Image after performing resize and normalizing techniques

Note. The above image shows a sample image after performing resizing and normalizing.

Color Augmentation. To make the photos more bright, color enhancement was performed at a ratio of 1.2. Originally, color intensity levels ranged from 0.7 to 1.0 as seen in Figure 32, depending on illumination and drone camera quality.

Figure 32

Image before performing color augmentation

Note. The above image shows a sample image before performing color augmentation.

Following augmentation, color intensity levels were enhanced to a constant range of 0.84 to 1.2, adding bright and recognizable elements to the datasets illustrated in Figure 33.

Figure 33

Image after performing color augmentation

Note. The above image shows a sample image after performing color augmentation.

Normalization of Depth Maps. Depth maps, which reflect the relative distances between objects, were normalized to a consistent range of 0 and 1. As seen in Figure 34, before normalization, depth numbers fluctuated greatly based on the drone's altitude and camera settings, ranging from 0 to 255.

Figure 34

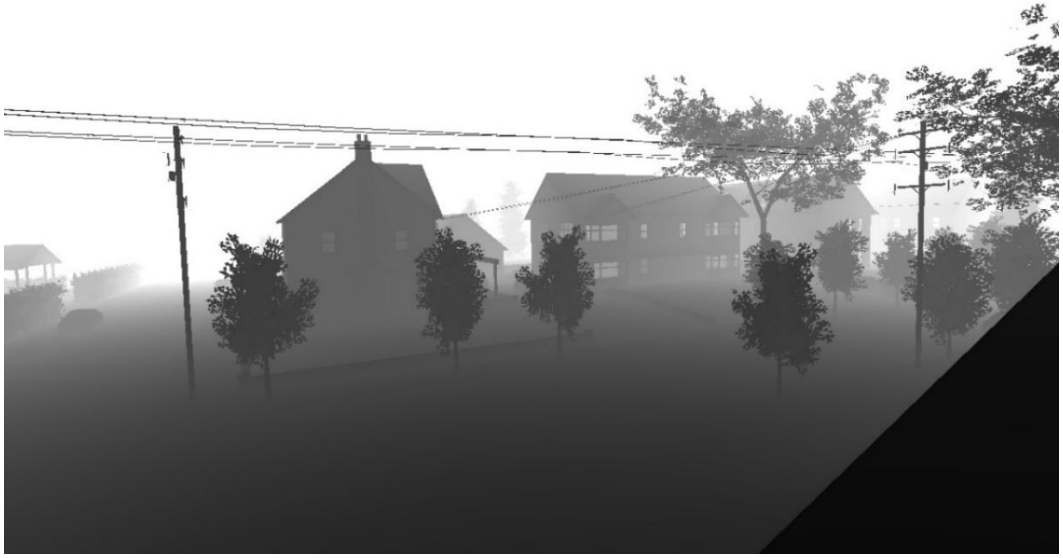
Image before performing normalization of depth maps

Note. The above image shows a sample image before performing normalization of depth maps.

Following normalization, Figure 35 illustrates depth data were uniformly scaled to the range [0, 1], ensuring a consistent representation of relative distances for model training.

Figure 35

Depth Map Normalized Image



Note. The above image shows a sample image which is a depth map normalized.

Processing of Segmentation Masks. Segmentation masks were transformed to grayscale, with each class label assigned a preset pixel value. Before processing, masks were saved in RGB format, with class labels rendered in an inconsistent color scheme as seen in Figure 36. This created extra complications during training.

Figure 36

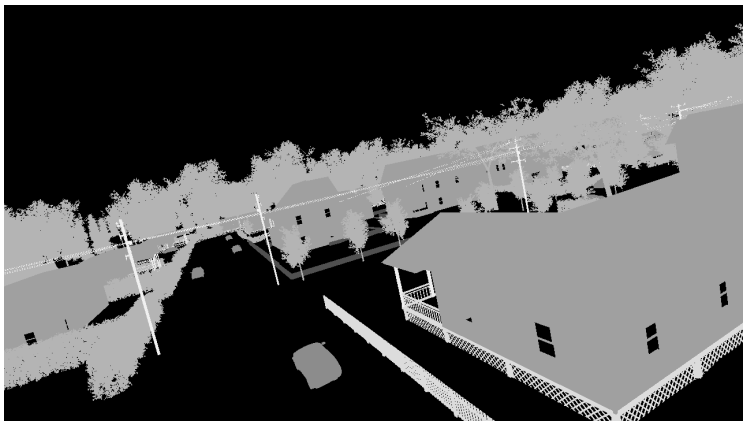
Image before performing segmentation masks on images

Note. The above image shows a sample image before performing segmentation masks on images.

Masks were simplified after preprocessing, with pixel intensity values assigned to their respective class labels (255 for ultra-thin structures, 200 for big mesh) in Figure 37. The grayscale scale spanned from 80 to 255, therefore the segmentation information was computationally efficient for the model.

Figure 37

Segmentation Mask Processed Image



Note. The above image shows a sample image that is segmented and masked.

Histogram Equalization. Histogram equalization was applied to redistribute pixel intensity values across the full dynamic range. Initially, many images had pixel intensity values concentrated in narrow ranges, particularly in low-light conditions, with intensity distributions skewed between 50 and 150 as seen in Figure 38.

Figure 38

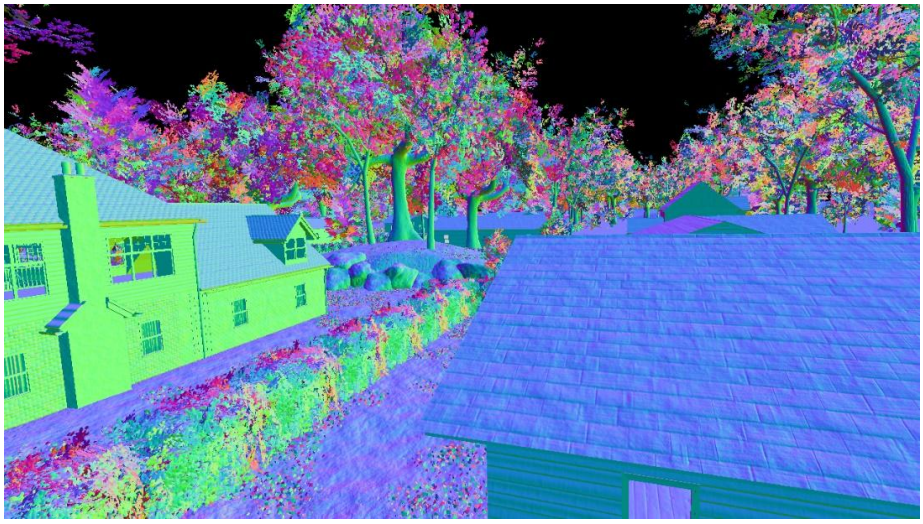
Image before performing histogram equalization on images

Note. The above image shows a sample image before performing histogram equalization.

After histogram equalization, pixel values spanned the full range of [0, 255] as illustrated in Figure 39, enhancing contrast and making finer features more visible.

Figure 39

Image after performing Histogram Equalization



Note. The above image shows a sample image after *performing histogram equalization*.

Standardization. Normalization was achieved by subtracting the mean pixel intensity and dividing by the standard deviation. Originally, distributions of pixel intensities ranged across large spans, with mean values from 120 to 150 and standard deviations from 40 to 60 as seen in Figure 40.

Figure 40

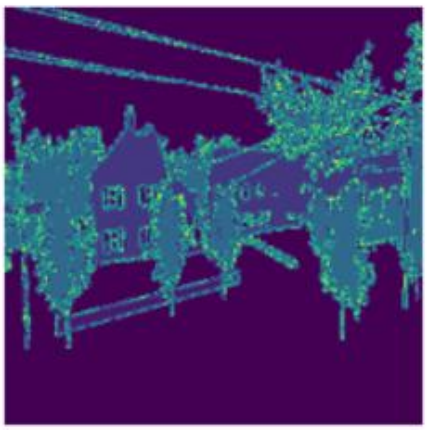
Image before performing standardization on images

Note. The above image shows a sample image before performing standardization.

The normalization resulted in the images taking the same distribution for the mean value of 0 and standard deviation of 1 as illustrated in Figure 41.

Figure 41

Image after performing standardization



Note. The above image shows a sample image after standardization

Data Transformations

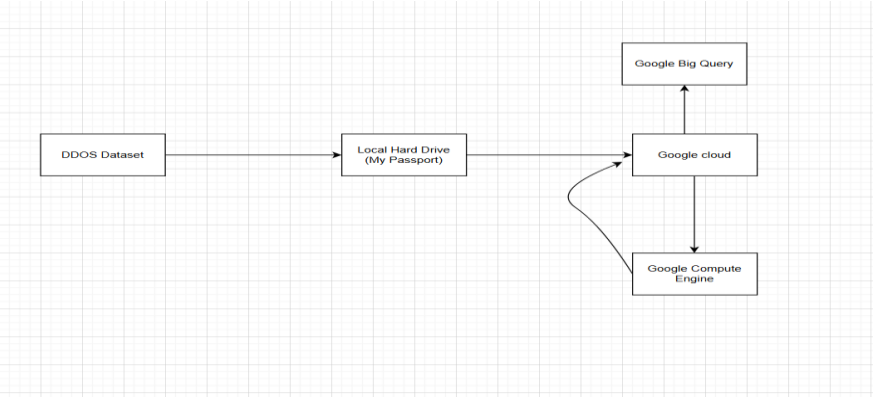
Before transitioning to a cloud-based solution using Google Compute Engine, an environment-specific local script was successfully built. This cloud-based code was designed to quickly analyze the massive DDOS dataset, which was first stored on a local hard disk for offline access and initial setup. When the dataset was ready to be moved to Google Compute Engine, its size and complexity were managed using an N2 instance with 1000 GB of boot disc space. The program was able to incorporate critical details by iterating over the weather files and metadata for each aircraft in this environment (Thangaraj, P., & Srivatsan, K., 2014).

The cloud-based script starts by importing the required libraries: logging to record activity, tqdm to track progress, CSV to manage CSV files, and os for file path manipulations. To connect with Google Cloud Storage (GCS), we additionally imported the required Google

Cloud libraries. To facilitate troubleshooting, a logging setup was put up to record script progress. We set up a storage client and specified important variables like base_path (the path to the validation data split inside the bucket) and bucket_name (the GCS bucket name) to communicate with GCS as seen in Figure 42. Before uploading the combined metadata CSV file to GCS, we also set up a temporary output route on the Compute Engine virtual machine.

Figure 42

Google cloud flow



Note. The above image shows the Google cloud flow.

By downloading model documents from GCS and examining their headers, the content may effectively establish the sections for climate and data records. Diverse envionments across multiple metadata and weather conditions records were handled by gradually separating these segment names using metadata_columns and weather_columns records.

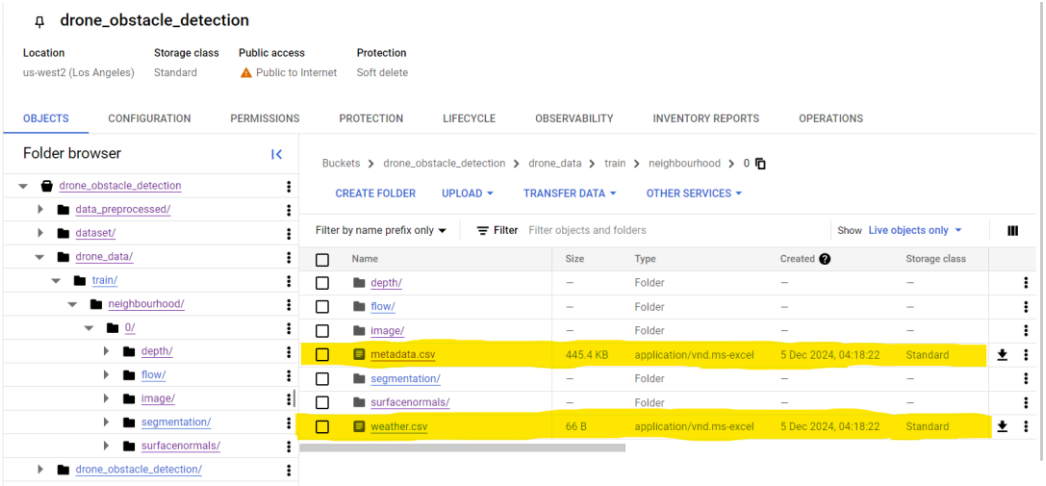
Following the layout of the segment structure, the content generated a strong rundown of parts including metadata, image paths, flight data, and weather information. The headers for the latest metadata CSV document were created using these segments. The content then captures and collects masses (documents) by aircraft ID, repeating across every location and park in the dataset. As a result of this, each flight folder was handled perfectly.

The content downloads and reads the weather.csv and metadata.csv files for each flight, provided they are available. To ensure consistency, weather condition values, and metadata are set to "N/A" for lacking information fields. The content repeats over the comparative picture documents after reading each line in the metadata record to describe routes for each image in addition to going with profundity, division, stream, and surface normals records. From that point forward, this data is exported from the CSV document as a single column including established picture relationships, climate, and metadata.

Figure 43 depicts the folder in which metadata and weather information were saved in GCS for each flight. Figures 44 and 45 show the metadata for each flight path and weather metadata for each flight before performing preprocessing. Figure 46 depicts the completed master metadata CSV document that is provided on GCS after each flight has been handled. This record is now being stored in GCS and is ready to be loaded into Google BigQuery for advanced analysis and exploratory data analysis. With exceptional metadata records for each data split, this intentional approach improves information investigation and model preparation while utilizing Google Cloud's adaptable framework for solid and viable information management. Figures 47 depicts the contents of the master metadata files.

Figure 43

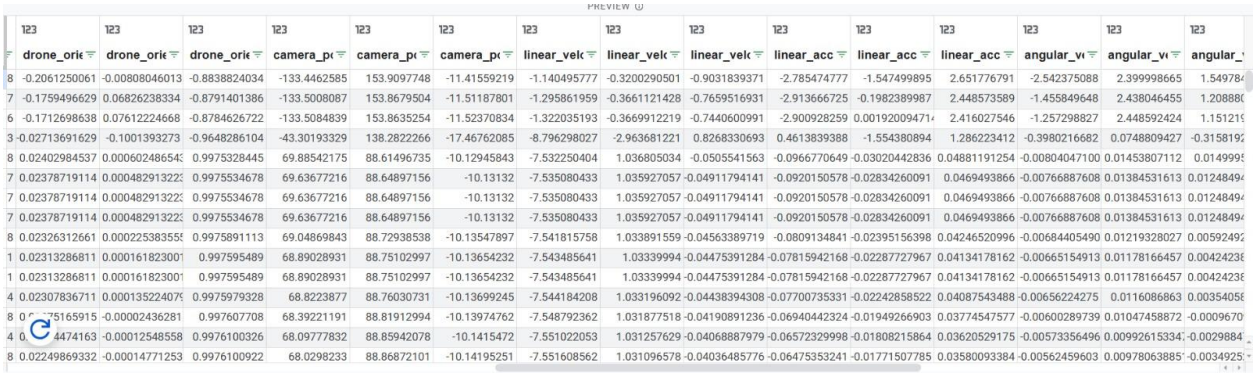
Metadata and weather files are stored for each flight



Note. The above image shows the GCS folder in which flight and weather metadata is stored.

Figure 44

Flight metadata file before preprocessing



	drone_orie	drone_orie	drone_orie	camera_p	camera_p	camera_p	linear_vel	linear_vel	linear_vel	linear_acc	linear_acc	linear_acc	angular_v	angular_v	angular_
8	-0.2061250061	-0.00808046013	-0.8838824034	-133.4462585	153.9097748	-11.41559219	-1.140495777	-0.3200290501	-0.9031839371	-2.785474777	-1.547499895	2.651776791	-2.542375088	2.399998665	1.54978
7	-0.1759496629	0.06826238334	-0.8791401386	-133.5008087	153.8679504	-11.51187801	-1.295861959	-0.3661121428	-0.7659516931	-2.913666725	-0.1982389987	2.448573589	-1.455849648	2.438046455	1.20888
6	-0.1712698638	0.07612224668	-0.8784626722	-133.5084839	153.8635254	-11.52370834	-1.322035193	-0.3669912219	-0.7440600991	-2.900928259	0.00192009471	2.416027546	-1.257298827	2.448592424	1.15121
3	-0.02713691629	-0.1001393273	-0.9648286104	-43.30193329	138.2822266	-17.46762085	-8.796298027	-2.963681221	0.8268330693	0.4613839388	-1.554380894	1.286223412	-0.3980216682	0.0748809427	-0.3158192
8	0.02402984537	0.00060248654	0.9975328445	69.88542175	88.61496735	-10.12945843	-7.532250404	1.036805034	-0.0505541563	-0.0966770649	-0.03020442836	0.04881191254	-0.00804047100	0.01453807112	0.014999
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
8	0.02326312661	0.00022538355	0.9975891113	69.04869843	88.72938538	-10.13547897	-7.541815758	1.033891559	-0.04563389719	-0.0809134841	-0.02395156398	0.04246520996	-0.00684405490	0.01219328027	0.0059249
1	0.02313286811	0.00016182300	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
1	0.02313286811	0.00016182300	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
4	0.02307836711	0.00013522407	0.9975979328	68.8223877	88.76030731	-10.13699245	-7.544184208	1.033196092	-0.04438394308	-0.07700735331	-0.02242858522	0.04087543488	-0.00656224275	0.0116086863	0.0035405
8	0.[illegible]75165915	-0.00002436281	0.997607708	68.39221191	88.81912994	-10.13974762	-7.548792362	1.031877518	-0.04190891236	-0.06940442324	-0.01949266903	0.03774547577	-0.00600289739	0.01047458872	-0.0009670
4	0.[illegible]4474163	-0.00012548558	0.9976100326	68.09777832	88.85942078	-10.1415472	-7.551022053	1.031257629	-0.04068887979	-0.06572329998	-0.01808215864	0.03620529175	-0.00573356496	0.00992615334	-0.0029884
8	0.02249869332	-0.00014771253	0.9976100922	68.0298233	88.86872101	-10.14195251	-7.551608562	1.031096578	-0.04036485776	-0.06475353241	-0.01771507785	0.03580093384	-0.00562459603	0.00978063885	-0.0034925

Note. The above image shows the flight metadata before preprocessing.

Figure 45

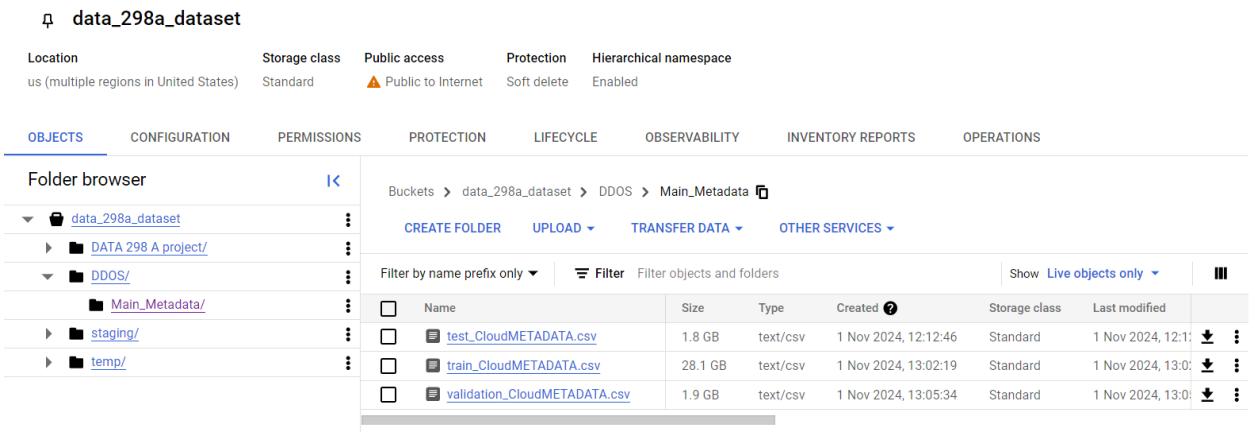
Weather metadata file before preprocessing

rain	roadwetne	snow	roadsnow	mapleleaf	dust	fog
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Note. The above image shows the weather metadata before preprocessing.

Figure 46

Master metadata file created and stored in GCS



Note. The above image shows the master metadata file created and stored in GCS.

Figure 47

Contents of the master metadata file

flight_id	dataset	location	timestamp	image_file	image_pat	depth_pat	segmentat	flow_path	surface_n	weather_c	timestamp	img_index	drone_pos	drone_pos
0	train	neighbourhood	1.70828E+18	12.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	13.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	14.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	15.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	16.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	17.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	18.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	19.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	2.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	20.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	21.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	22.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	23.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	24.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	25.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895

Note. The above image shows the content of the master metadata file.

Several new fields were added to the unified master metadata file to improve data structure and allow for more detailed analysis. The flight_id column uniquely identifies each flight session, allowing for the monitoring and differentiation of data points from several

flights—an important feature for data organization and analysis. The dataset column simplifies operations by indicating the data split (train, test, validation, and so forth). The location column gives valuable information for evaluating model performance across different terrains by describing the kind of setting (e.g., park, neighborhood).

To record the file paths for the various data modalities connected to every image, additional columns were created. Each picture file's name is included in the image_file column, and the image_path column gives the whole path to the original RGB image that the drone took.

The depth_path, segmentation_path, flow_path, and surface_normals_path columns contain the paths to the respective files containing depth maps, segmentation masks, optical flow data, and surface normal maps. These pathways allow for flexible and efficient data retrieval by simplifying the process of fetching multiple data representations for each image. When combined, these columns provide a well-organized dataset that facilitates in-depth analysis and model training by giving quick access to all required images and auxiliary data files (Fernandes, R. et al., 2024).

Data Annotations

The annotation starts by describing class mappings for each type of obstacle in the division covers, for example, ultra_thin, thin_structures, small_mesh, and so on, with each class addressed by extraordinary pixel esteem in grayscale. We also assign a random color to each class using COLOR_MAPPING which will be used for drawing jumping boxes and names on the clarified pictures.

For each flight folder, the content downloads images and their comparing division covers from GCS into a brief neighborhood catalog on the Google Process Motor occasion. Using OpenCV, the content cycles over each partition veil to identify forms for each obstacle

class. For each distinct form across a particular area limit (e.g., 50 pixels to ignore noise), we determine that the bouncing box facilitates and attracts a square shape on the initial image of the doled-out variety. At this stage, each bouncing box, along with its class name, focuses on the visual interpretation of the groups.

Figure 48 shows the sample image before performing annotations on the image. Once the annotated picture has been created as shown in Figure 49, it is briefly saved and then uploaded back to GCS, using a specified route for annotated photos. This systematic technique aids in keeping the original and annotated photos separate. The script is also built to handle large amounts of data effectively by utilizing multiprocessing (Pool from Python's multiprocessing package), which allows for parallel flight processing. This technique shortens processing time by using several CPU cores, making it perfect for cloud-based execution.

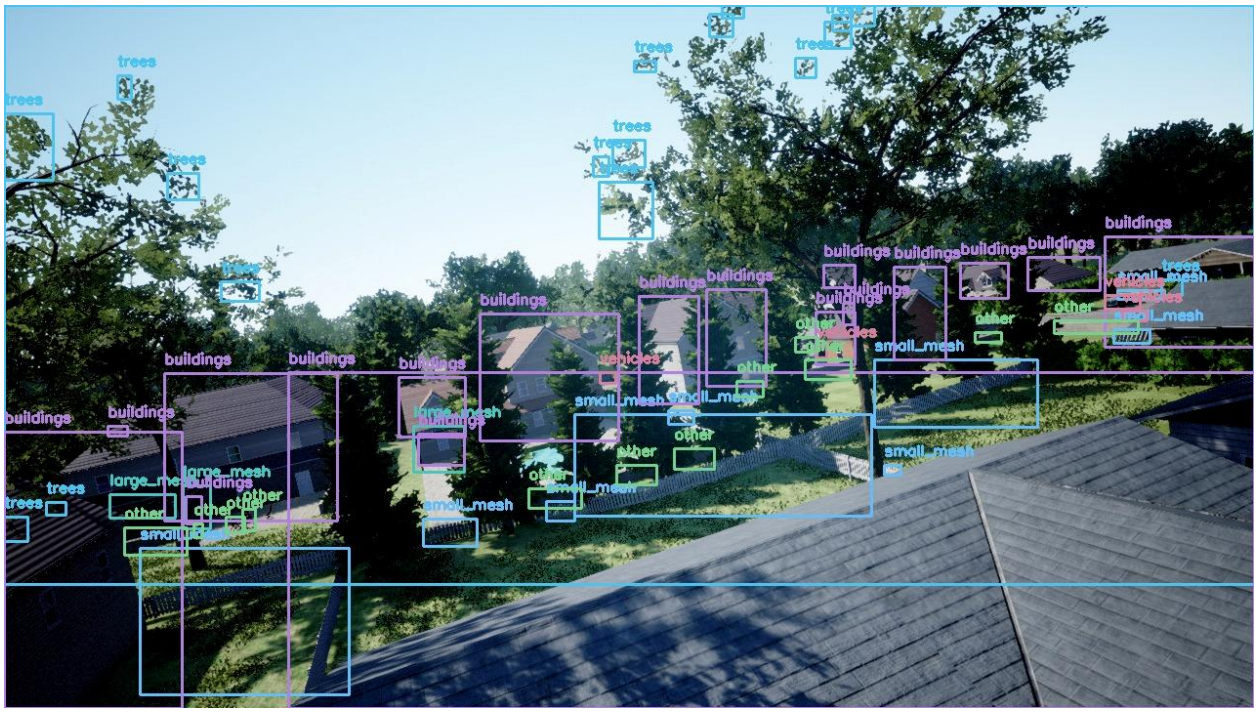
Figure 48

Image before performing annotation

Note. The above image shows a sample image before performing the annotations using OpenCV.

Figure 49

Image after performing annotation



Note. The above image shows a sample image after performing the annotations using OpenCV.

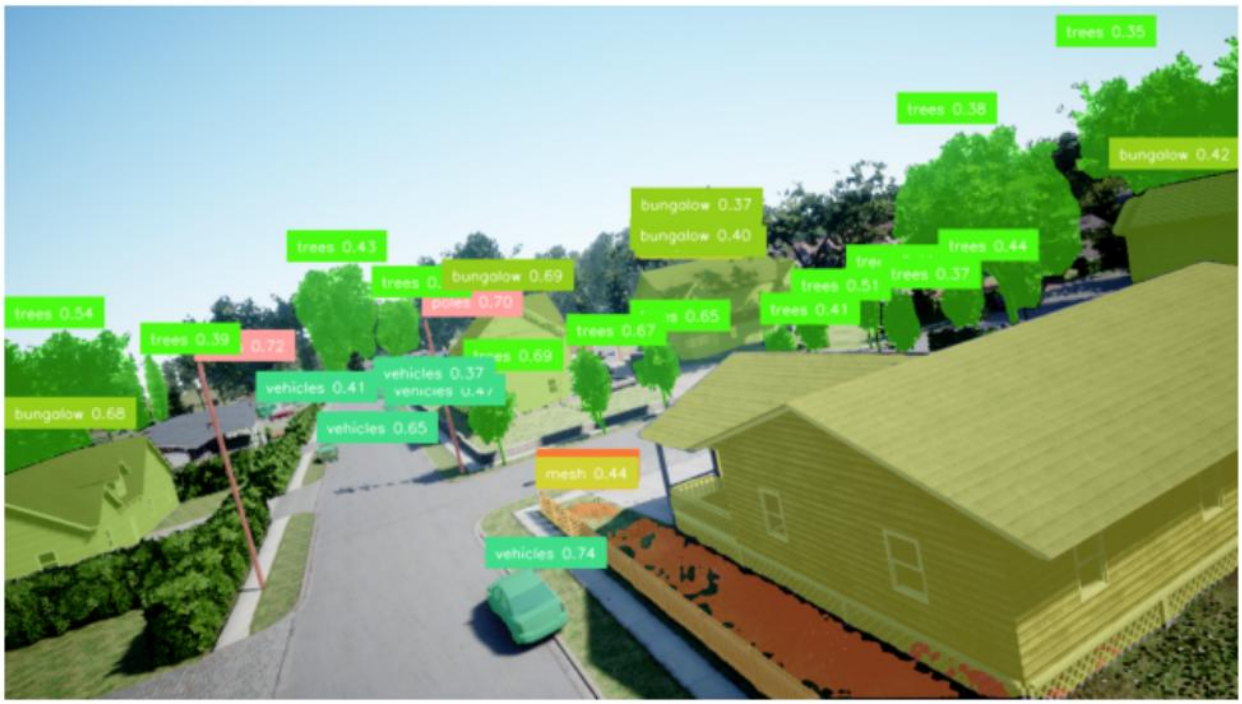
Data annotation is a prerequisite for creating a sound dataset that will be used for object detection and classification tasks. Therefore, the annotation for this project was performed using a state-of-the-art tool for automatic object detection and segmentation called the Grounded Segment Anything Model. It uses ontology-based classification, enabling efficient identification and labeling of objects in images, producing quality and consistent annotations. This process greatly reduces the manual effort in annotation, yet it is as precise and accurate on a diverse dataset.

These provided the necessary semantic representation mapping from classes of objects including ultra-thin, fences, vehicles, and trees, so annotation could be fast and more straightforward. For detecting such objects from images, Grounded SAM would identify

them through a lookout for their semantic counterparts while segregating them based on their set categories as seen in Figures 50 and 51. The model then detects objects in the images, estimating their bounding box coordinates and class labels. This automation not only accelerates the annotation of data but also scales well to large datasets, hence being suitable for obstacle detection in drone navigation.

Figure 50

Image after performing annotation using GroundedSam



Note. The above image shows a sample image after performing the annotations using GroundedSam.

Figure 51

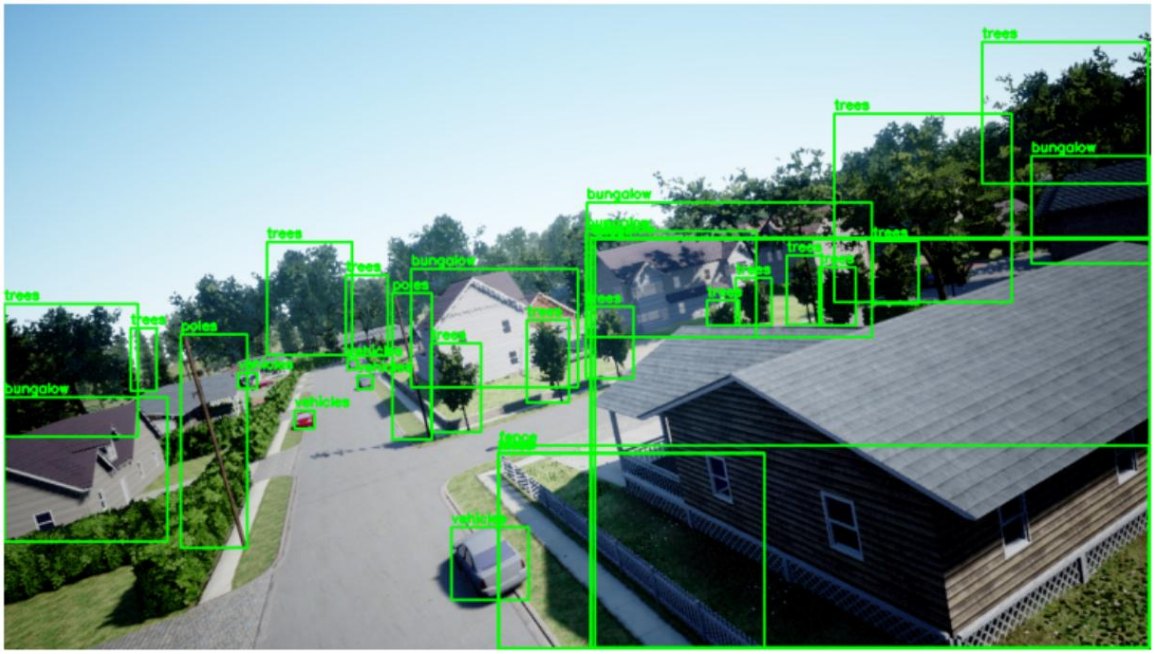
Image after performing annotation using GroundedSam

Note. The above image shows a sample image after performing the annotations using GroundedSam.

The annotations were kept in a YOLO-friendly format, which is popular for training deep-learning object detection models. Normalizing the bounding box coordinates regarding image dimensions was done for each of the detected objects as seen in Figures 52 and 53, while the annotations were written down in text files. Each of the annotation files contained the class ID, the center coordinates of the bounding box, and the width and height for compatibility with object detection frameworks. These annotation files were then stored along with annotated images in a structured format and were ready for the next stages of training and model development.

Figure 52

Image after performing annotation and generating a bounding box for classes

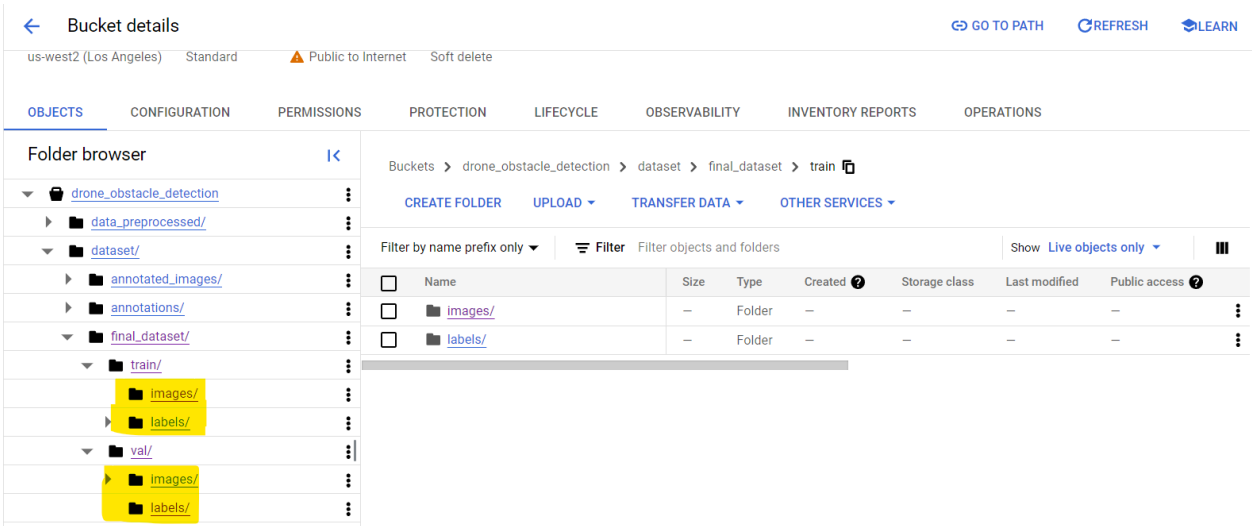

trees
trees
bungalow
bungalow
trees
trees
trees

Data preparation

Along with the annotation, a dataset was prepared and arranged in a format that could be used by the model for training. The dataset would be divided into three splits: training, validation, and testing. Each split would consist of images and corresponding annotation files, which are stored in separate folders on GCS as seen in Figure 54. This makes the data organized and hence directly accessible by the model without any pre-processing or preparation at the time of training. The YAML configuration file then had to be created defining the structure of the dataset: the path to training, validation, and test data; class ID mapping to labels as seen in Figure 55. It was also uploaded on GCS for easy integration into the model training pipeline. Figure 56 shows all annotations stored in a single folder after annotations whereas Figure 57 shows all the images stored in the folder after performing all the annotations.

Figure 54

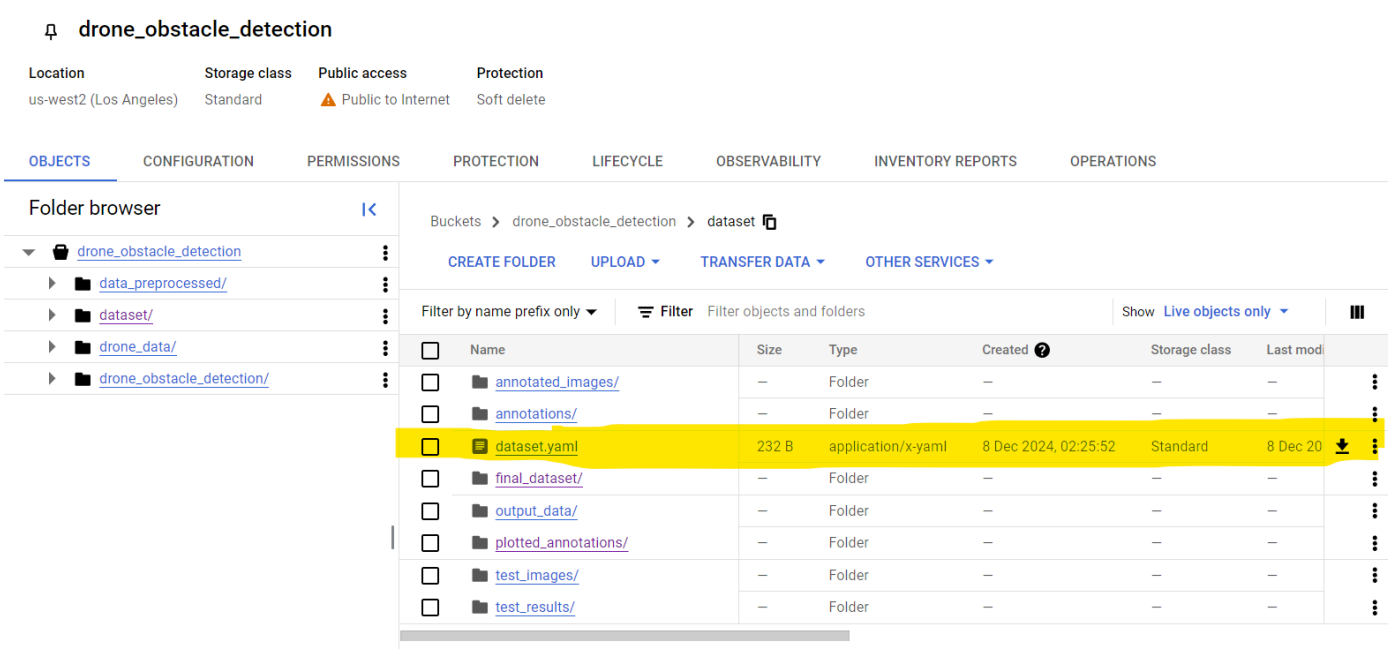
Images and annotations stored in a structure suitable for modeling in GCS



Note. The above image shows the GCS folder in which images and annotations are stored in GCS.

Figure 55

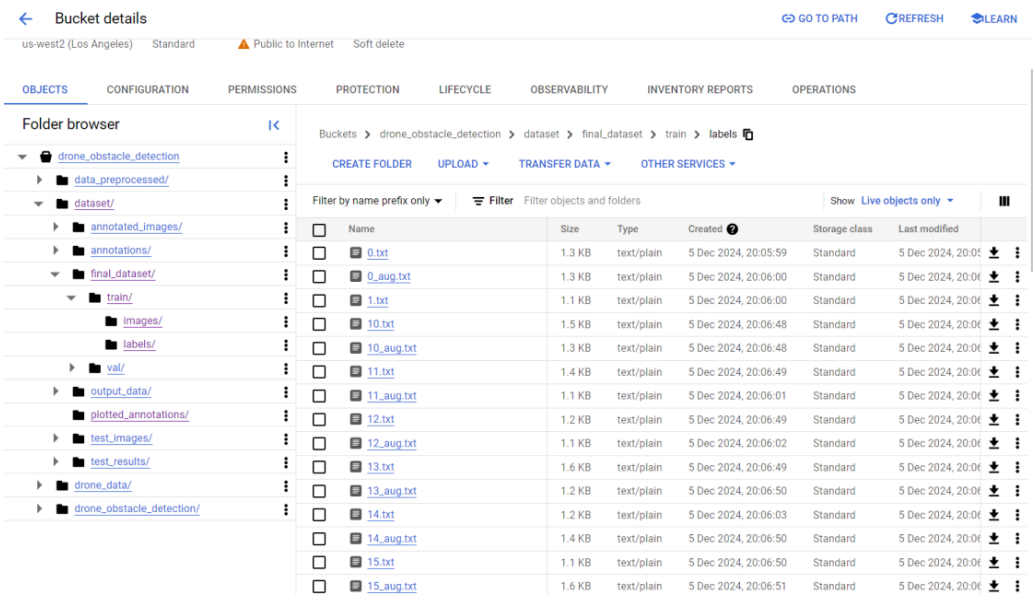
Yaml file created and stored on GCS



Note. The above image shows the Yaml file created and stored in GCS.

Figure 56

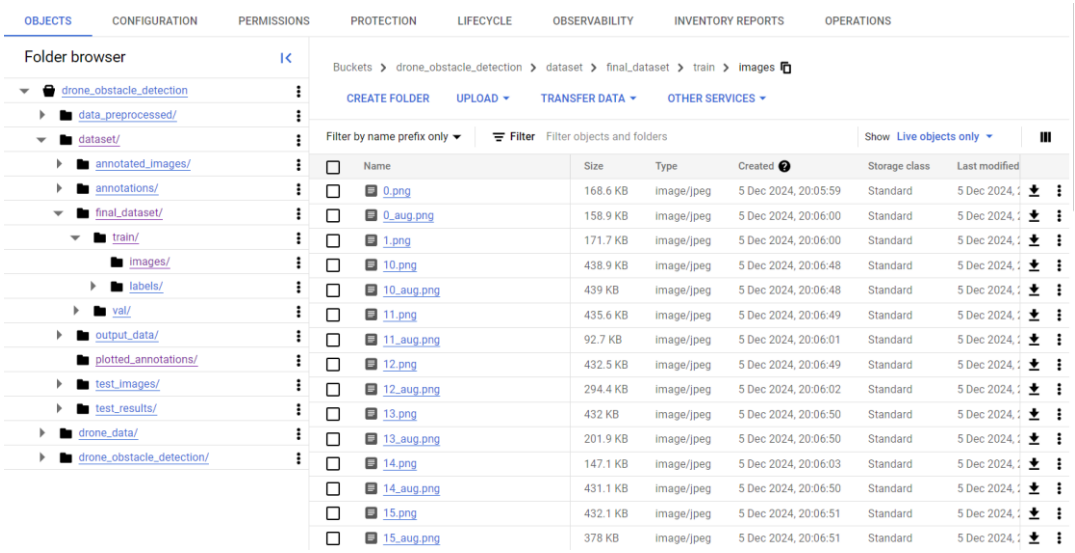
Annotations are created and stored on GCS in compatible folders



Note. The above image shows the annotations created and stored in GCS.

Figure 57

Images created and stored on GCS after annotations in compatible folders



Note. The above image shows the images created and stored in GCS after annotations.

This pipeline for annotation and preparation of the dataset ensured that the data was comprehensively labeled, well-organized, and fully compatible with the YOLO training framework. Using the Grounded SAM has ensured efficiency and accuracy in addressing such challenges as finding small, complex, or overlapped objects in the images. These high-quality annotations and structured datasets were extremely important in developing an object detection model that would work in real-world scenarios, like detecting and classifying obstacles for drone navigation in a diverse set of environments. This step not only laid the foundation for model training but also ensured that the dataset met the required standards for performance and scalability. Figure 58 shows the contents of the Yaml file created in GCS. Figure 58 shows the sample annotations file for an image showing the coordinates and classes for objects in a file.

Figure 58

Contents of Yaml file created

```yaml
names:
- ultra-thin
- thin
- poles
- fence
- wires
- mesh
- trees
- forest
- bungalow
- vehicles
- miscellaneous
nc: 11
path: gs://drone_obstacle_detection/dataset/final_dataset
test: test/images
train: train/images
val: val/images
```

Note. The above image shows the content of yaml file created.

Figure 59

Sample annotations file

```
2 0.18205034732818604 0.6775521172417535 0.0586888313293457 0.3293611314561632
2 0.35408620834350585 0.5597344716389974 0.03992838859558105 0.22976286146375868
3 0.13670705556869506 0.7771304660373264 0.27192614078521726 0.4421639336480035
4 0.18223272562026976 0.677783711751302 0.058362364768981934 0.3307512495252821
5 0.49991512298583984 0.5201109144422743 0.996614646911621 0.9503467135959202
5 0.7140974521636962 0.6806719462076823 0.5701842308044434 0.632631598578559
6 0.49987058639526366 0.4949000040690104 0.9984533309936523 0.9052585177951389
6 0.39382143020629884 0.5942600250244141 0.04415895938873291 0.1366967095269097
6 0.47624912261962893 0.5555509355333116 0.03676929473876953 0.1250732421875
6 0.5264222621917725 0.5212158626980252 0.044704437255859375 0.1119524637858073
7 0.49994487762451173 0.497383541531033 0.9972160339355469 0.9113605075412327
8 0.0887477695941925 0.7170099894205729 0.17653871774673463 0.2271900177001953
8 0.7552656650543212 0.6794887966579861 0.48825788497924805 0.6349016401502822
8 0.42795820236206056 0.5013636271158854 0.14358701705932617 0.1884569803873698
9 0.4234774112701416 0.8665782504611546 0.06721293926239014 0.11560821533203125
```

Note. The above image shows the sample annotations file with labels and coordinates

Information planning is an essential step that ensures the transformed datasets are well-organized and set for model preparation, validation, and testing. This is a process that

incorporates the data into three clear subgroups or divisions, including training, validation, and testing data, each with its particular role within the workflow of AI.

The planning system begins by parting the changed and preprocessed datasets into preparing, approval, and test sets to keep a reasonable portrayal across all subsets. This splitting is based on the information gathered from various flight conditions, such as metropolitan areas and parks, to ensure that the model learns from various scenarios and generalizes well to new data. The data is split in a ratio of 80:10:10, meaning 80% goes to training and 10% each for testing and validation. Figure 60 shows the metadata split in the test, train, and validation set in GCS.

Training Dataset

The preparation set contains 30,000 images over 300 independent flights. It is used to train the article detection models, helping them to learn examples, highlights, and item features. Such a broad preparation set size is able to ensure that these models are outstanding in complex scene handling and variable types of obstacles.

Validation Dataset

The approved set includes 2,000 images from 20 distinct aircraft. During the model turn of events, this collection is used to fine-tune model hyperparameters and screen execution, taking into account modifications that improve accuracy and reduce overfitting.

Test Dataset

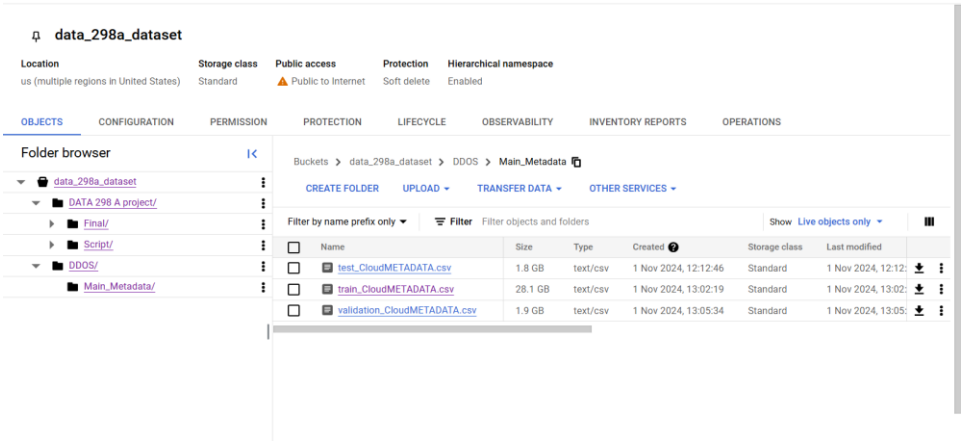
The test set, including 2,000 pictures from 20 flights, remains for the last assessment of this model's exhibition. It then provides a fair appraisal as to how well the trained model generalizes to subtle information.

Sample Data from Prepared Datasets

The presented samples from the preparation, approval, and test sets demonstrated the adequacy of the information readiness process. Each example represents different flight conditions about the comprehensiveness of the data set. Figure 61-63: Sample of training image data, test, and validation, respectively.

Figure 60

Splitting the metadata in test, train, and validation splits and uploading in GCS



Note. The above image shows the metadata split in test, train, and validation sets.

Figure 61

Sample of Training data

Note. The above image shows the sample of training data.

Figure 62

Sample of Testing data

Note. The above image shows the sample of testing data.

Figure 63

Sample of Validation data

Note. The above image shows the sample of validation data.

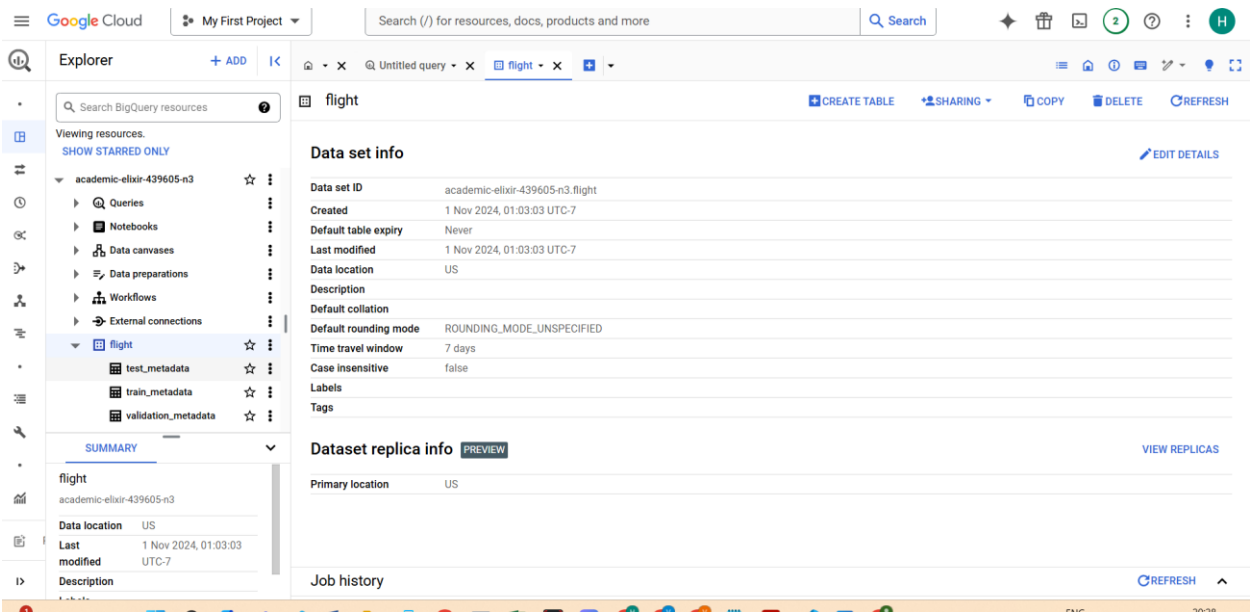
Data Statistics

Following the successful execution of the script on Google Cloud, we uploaded the master metadata files for the train, test, and validation datasets to three separate tables in Google BigQuery—train_metadata, test_metadata, and validation_metadata—all of which are contained under the flight dataset. This systematic strategy allows for excellent querying and analysis inside a centralized platform while ensuring metadata separation for each dataset split.

With different tables, we can execute unique queries for each dataset split, which is extremely useful for performing exploratory data analysis (EDA). Figure 64 depicts the data travel in GCS.

Figure 64

Information of a single flight of data



Note. The above image shows the information of a single flight of data.

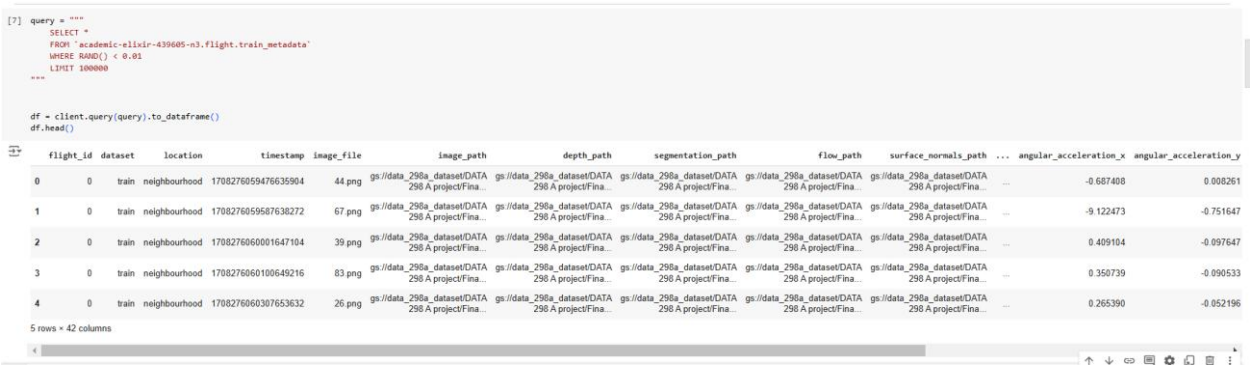
After developing and uploading the metadata documents to learn about BigQuery, we continued to see the data using Google Colab. By accessing the Colab scratch pad, we were

able to directly connect to BigQuery, allowing us to conduct SQL queries and get data for analysis. This collaboration between Colab and BigQuery creates an adaptive and intuitive environment for information inquiry and perception, allowing us to gain expertise in training, testing, and approving datasets (Adhikari, B., & Huttunen, H. 2020).

Figure 65 shows that in the query, an irregular 1% example of up to 100,000 records is chosen from the flight train_metadata table inside the scholarly mixture 439605 dataset. This testing is finished utilizing WHERE RAND() < 0.01 to guarantee that the main a little, irregular subset is recovered from a bigger dataset. This query is structured to draw on various metadata fields, such as flight_id, dataset, area, timestamp, image_file, image_path, and a couple of sensor-related credits, including exact speed increase values. This is a reasonable chunk of information for an introductory investigation.

Figure 65

Sample of data shown using the query



```
[7] query = """
        SELECT *
        FROM `academic-elixir-439605-n3.flight.train_metadata`
        WHERE RAND() < 0.01
        LIMIT 100000
    """

    df = client.query(query).to_dataframe()
    df.head()
```

	flight_id	dataset	location	timestamp	image_file	image_path	depth_path	segmentation_path	flow_path	surface_normals_path	...	angular_acceleration_x	angular_acceleration_y
0	0	train	neighbourhood	1708276059476635904	44.png	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	...	-0.687408	0.008261
1	0	train	neighbourhood	1708276059587638272	67.png	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	...	-9.122473	-0.751647
2	0	train	neighbourhood	1708276060001647104	39.png	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	...	0.409104	-0.097647
3	0	train	neighbourhood	1708276060100649216	83.png	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	...	0.350739	-0.090533
4	0	train	neighbourhood	1708276060307653632	26.png	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	gs://data_298a_dataset/DATA 298 A project/Fina...	...	0.265390	-0.052196

5 rows × 42 columns

Note. The above image shows the sample of data shown using the query.

Figure 66 depicts a descriptive overview of the sampled data, revealing a wide range of drone flight and sensor data, recording a variety of flight occurrences and maneuvers. The flight_id range implies numerous distinct flights, although high-resolution timestamps indicate exact time monitoring. Positional data (drone_pose_x, drone_pose_y, drone_pose_z)

exhibit substantial spatial variance, indicating that the dataset includes flights at various locations and elevations. Orientation values across many axes represent consistent but varying rotational data, whereas angular acceleration values show a wide range, suggesting fast changes in movement and orientation. This heterogeneity across locations, orientations, and accelerations indicates a dataset rich in flight dynamics, which might be useful in training robust models capable of managing a wide range of aerial conditions.

Figure 66

Description of the whole data using query on the metadata file

```
df.describe()
```

	flight_id	timestamp	timestamp_1	img_index	drone_pose_x	drone_pose_y	drone_pose_z	drone_orientation_w	drone_orientation_x	drone_orientation_y	...	angular_acceleration_x	angular_acceleration_y	angular_accelerat
count	100000.0	100000.0	100000.0	100000.0	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	...	100000.000000	100000.000000	100000.0
mean	71.23958	1708352131487220480.0	1708352131487220480.0	4.09174	1.606876	13.729930	-11.295329	0.604523	-0.001587	-0.039287	...	0.007457	0.020438	0.0
std	71.532975	779551805375 15.34375	779551805375 15.34375	17.72511	77.666237	81.690412	3.413860	0.313291	0.128613	0.128144	...	10.897318	9.076429	2.3
min	0.0	1708276059395634176.0	1708276059395634176.0	-1.0	-172.079056	-174.164337	-21.304558	-0.070593	-0.376206	-0.379458	...	-141.418259	-154.479446	-7.6
25%	22.0	1708279384099559680.0	1708279384099559680.0	-1.0	-53.411667	-43.314053	-13.760966	0.326195	-0.061103	-0.104969	...	-0.520869	-0.245636	-0.6
50%	42.0	1708286008336873728.0	1708286008336873728.0	-1.0	11.014219	23.067546	-11.492371	0.685390	-0.001361	-0.028725	...	0.001571	-0.017875	0.0
75%	99.0	1708436705030606592.0	1708436705030606592.0	-1.0	57.201659	79.166908	-8.969328	0.901031	0.059034	0.009694	...	0.580370	0.144849	0.7
max	249.0	1708437303723378432.0	1708437303723378432.0	99.0	184.926224	183.128372	-2.448147	0.999741	0.361040	0.368041	...	159.814056	142.623108	7.6

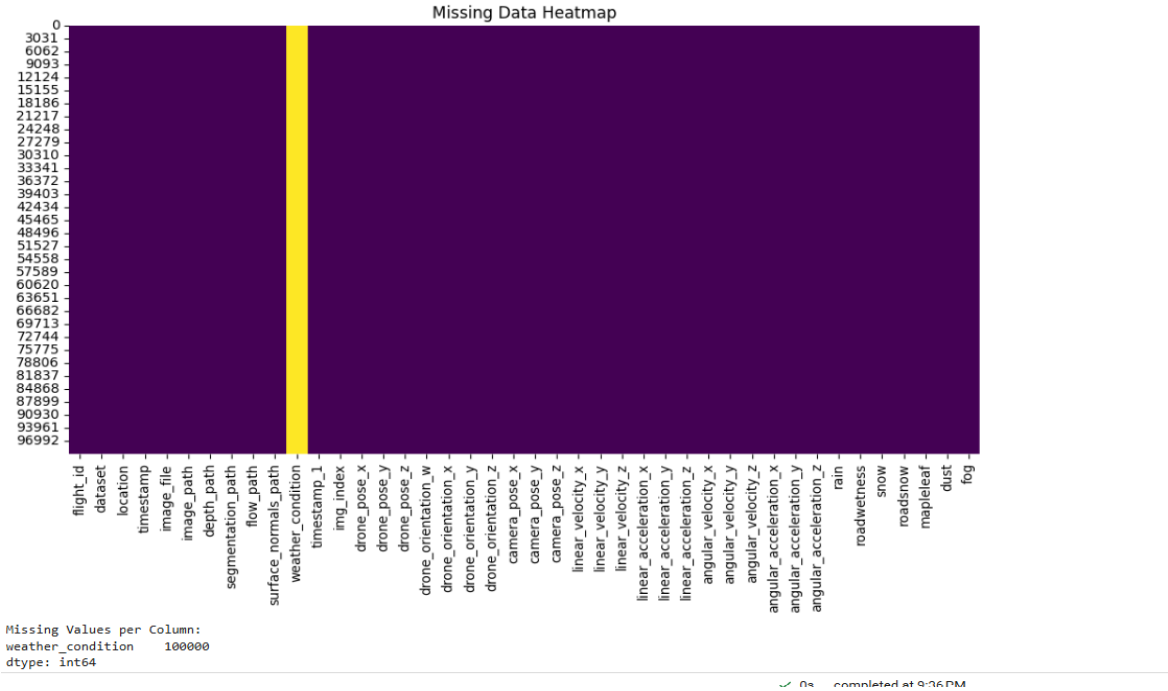
8 rows × 33 columns

Note. The above image shows the description of the whole data using query on the metadata file.

The heatmap represented in Figure 67 clearly indicates that the "weather_condition" sector is invalid, as revealed by the yellow stripe. This is usual, as it was recently created by the script and expects direct data from the creator of the dataset. Once there is directive input on how to infuse information from columns such as "rain" and "fog", this field will be ready for population in showing detailed climatic status for any flying event. This expansion will improve the dataset's capability to capture ecological elements, which could be huge in model preparation and assessment (Fernandes, R. et al. 2024).

Figure 67

Heatmap to show that the weather condition segment is less valid

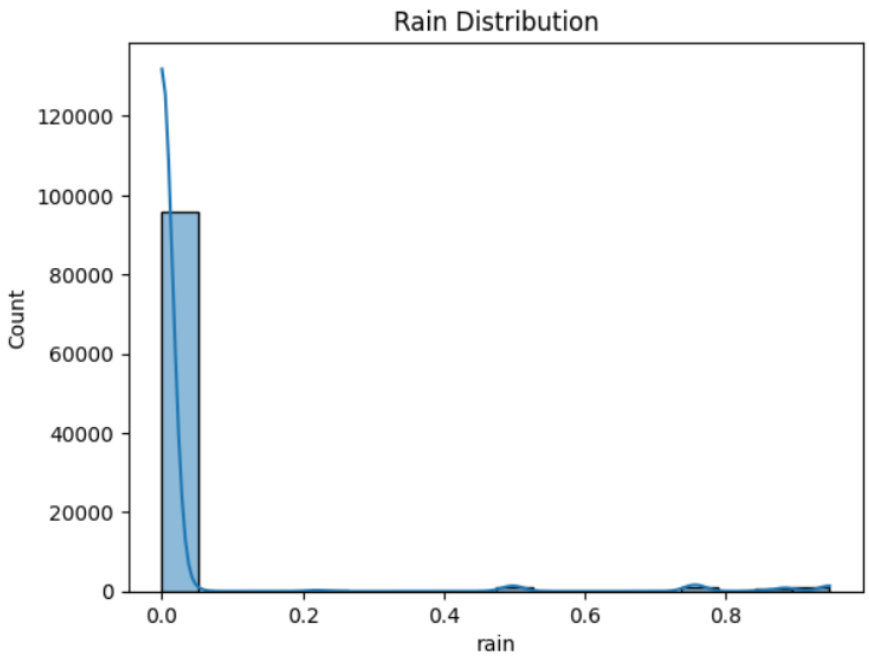


Note. The above image shows a heatmap to show that the weather condition segment is less valid

Figure 68 depicts a plot in which the majority of the values in the "rain" section are at or near zero, indicating largely dry weather with occasional light rain. There are a few cases of higher grades, with diverse but restricted rain events.

Figure 68

Rain Distribution

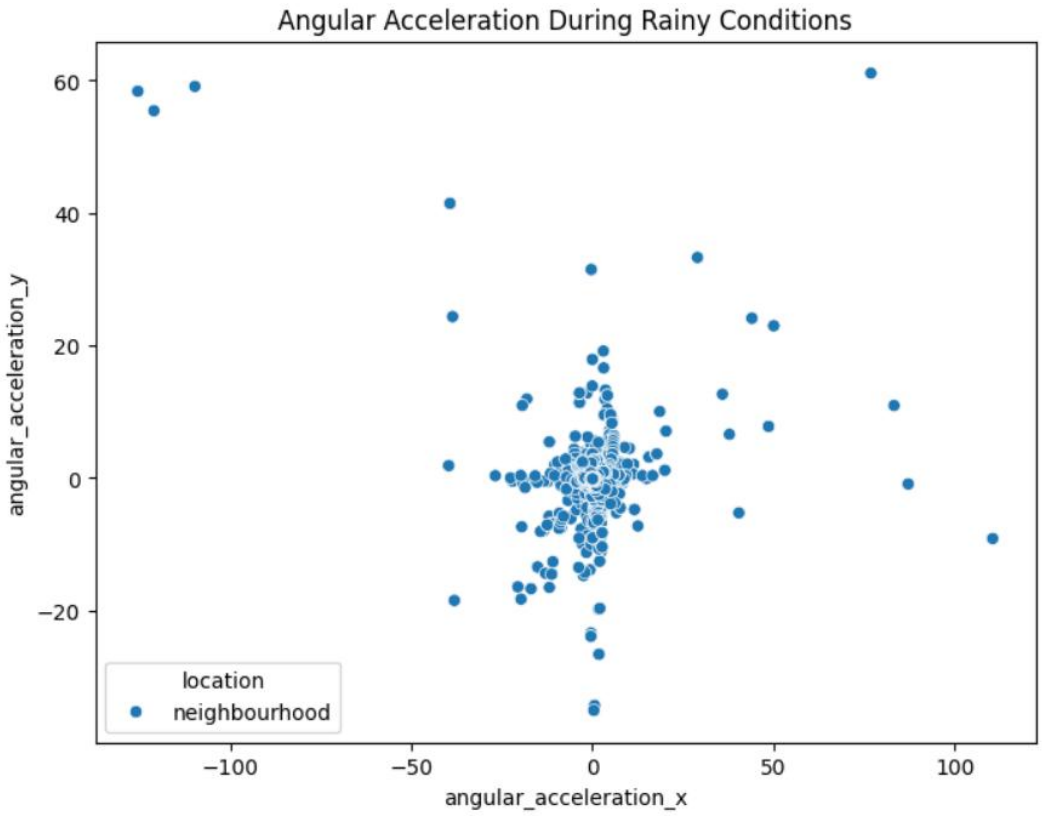


Note. The above image shows the rain distribution in the weather mertadata.

Figure 69 depicts the scatter plot of angular_acceleration_x and angular_acceleration_y during rainy circumstances, with data points grouped around the origin. The bulk of points on both axes are close to zero, showing that most drone movements in the "neighborhood" area have relatively modest angular accelerations in both directions during rain. Outliers with greater values along both the x and y axes do exist, indicating that there are occasional fast angular shifts. This spread might indicate specific flying tactics or environmental effects that affect the drone's stability during wet weather, making it useful for assessing how rain impacts drone movement and control.

Figure 69

Angular Acceleration during Rainy Conditions

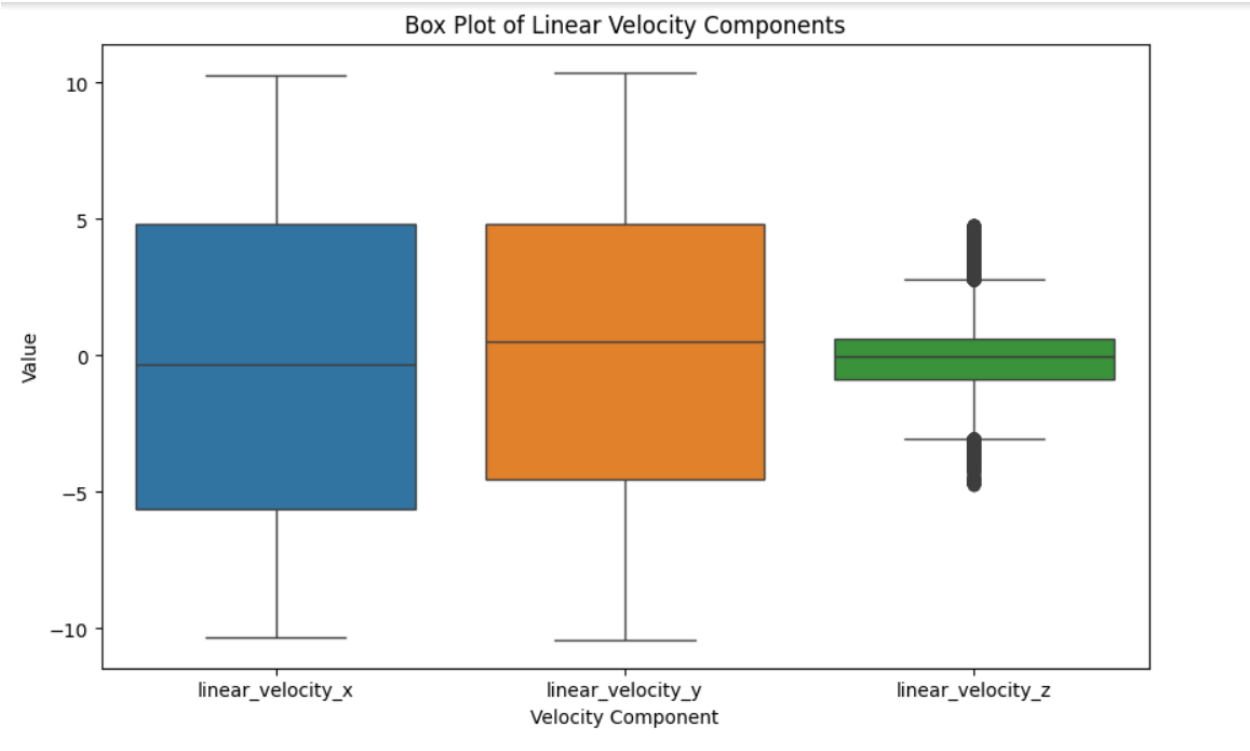


Note. The above image shows the angular acceleration during rainy conditions.

Figure 70 shows the box plot of linear velocity components (linear_velocity_x, linear_velocity_y, and linear_velocity_z) The box plot above shows that the dispersion in linear_velocity_x and linear_velocity_y is wider, ranging from approximately -10 to 10, while there is more variability in the horizontal movement directions. For both, the median value is close to zero, which means that around the center, the distribution is quite balanced. Conversely, linear_velocity_z is rather narrow with few outliers, and an interquartile range that centers closely around zero, which implies a much more stable vertical velocity without much fluctuation. It would therefore indicate from this plot that the majority of the movements occur along the horizontal axes and with relatively stable vertical movement.

Figure 70

Box plot of linear velocity components

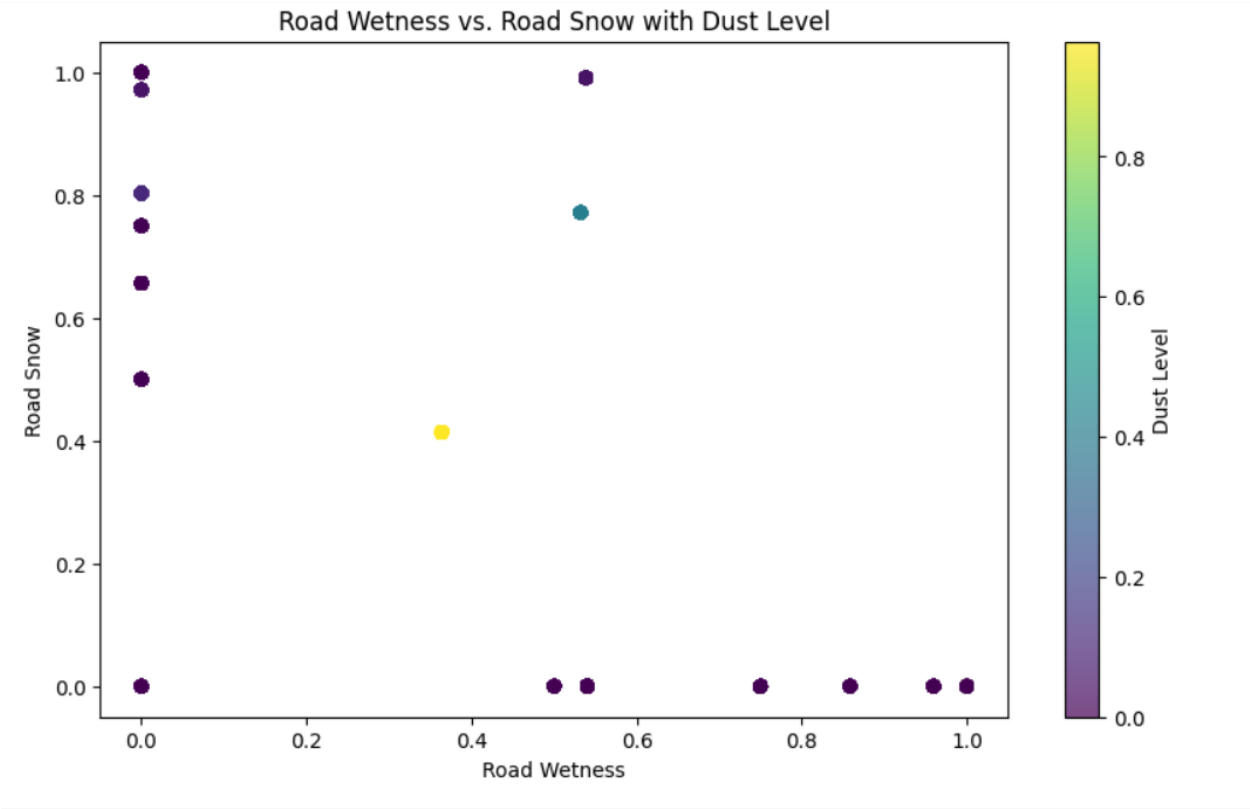


Note. The above image shows the box plot of linear velocity components.

This is the scatter plot in Figure 71 of Road Wetness versus Road Snow colored by Dust Level. Most points fall at the extremes of either road wetness or road snow, indicating that conditions are usually dominated by either wet or snowy roads and not a mix of both. The dust levels vary but are generally low across these conditions, with a few higher dust levels observed at moderate road snow and wetness levels. This plot suggests that the road conditions in this dataset rarely experience both high wetness and snow simultaneously, with isolated occurrences of dust present primarily when one condition (wetness or snow) is dominant. This can help in understanding environmental impacts on road surfaces and potential factors of visibility due to dust.

Figure 71

Road wetness vs Road Snow with Dust Level

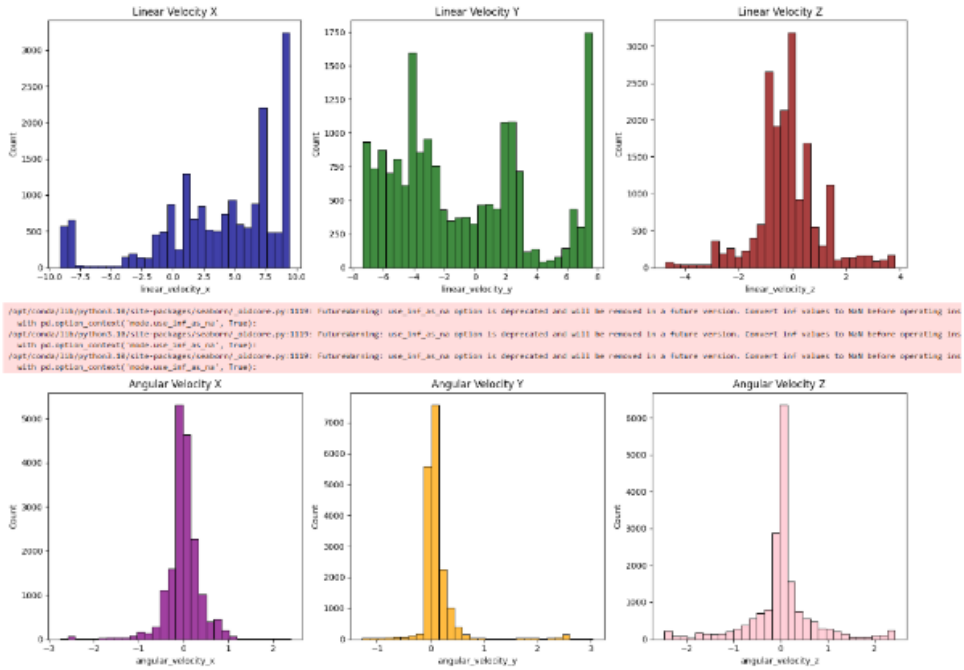


Note. The above image shows the road wetness vs road snow with dust level.

The histograms in Figure 72 show the distribution patterns of both linear and angular velocities of the drone along the X, Y, and Z axes. The linear velocity histograms show variations in drone speed across different directions, pointing to some velocities that are more common and may reflect preferred movement dynamics or operational constraints of the drone. Similarly, the angular velocity histograms show the frequency of rotational movements; large peaks indicate frequent rotations around a particular axis.

Figure 72

Distribution of linear and angular velocity components

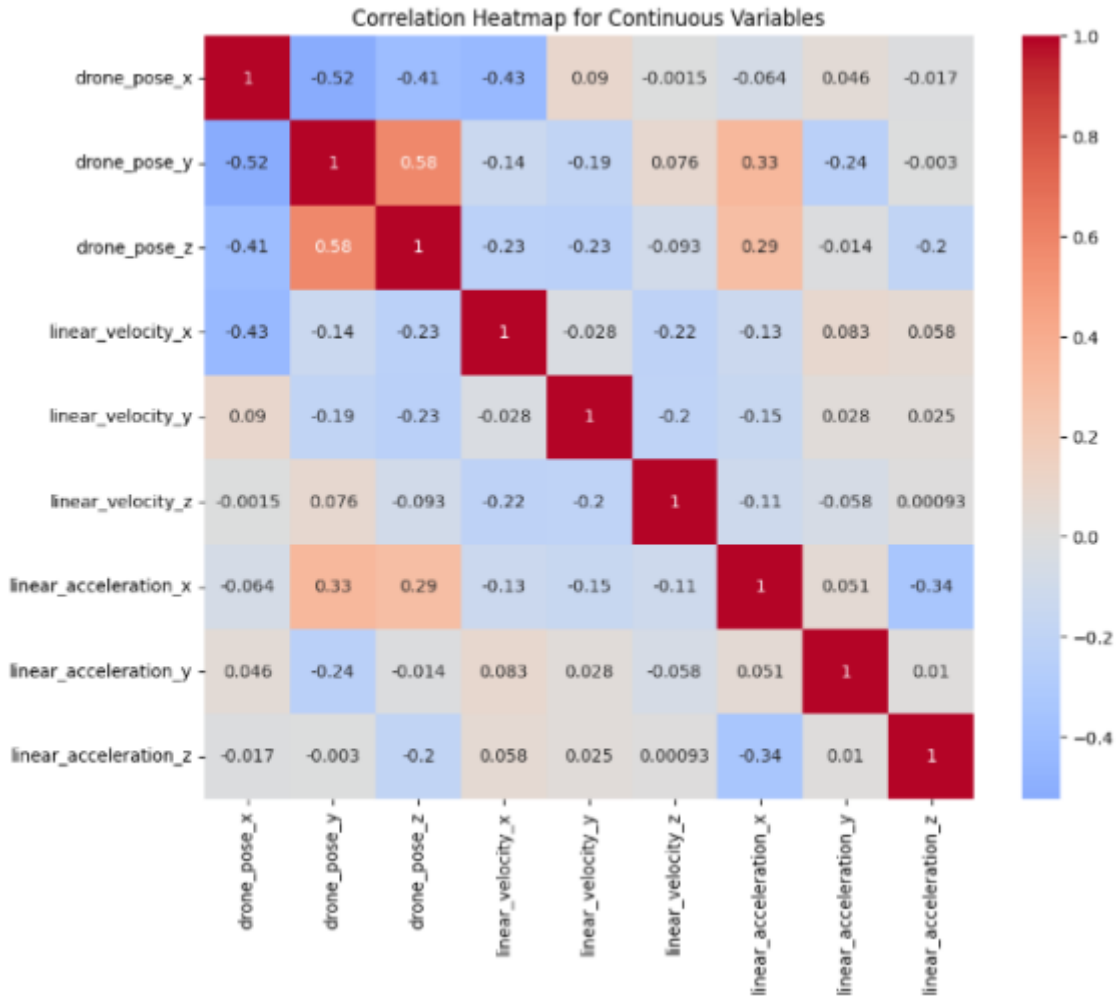


Note. The above image shows the distribution of linear and angular velocity components.

Figure 73 shows the correlation heatmap showing the linear relationships among different continuous variables. The color shade varies from dark red (strong positive) to dark blue (strong negative). For example, the strong positive value of drone_pose_y concerning drone_pose_z may suggest that these positional components change proportionally, whereas a strong negative value between drone_pose_x and drone_pose_y suggests an inverse relationship. Lighter-colored weak correlations indicate a small amount of linear dependence among the variables, which is crucial for identifying independent features. The analysis will help gain insight into variable interactions and facilitate feature selection relevant to the model, improving the predictive power of the models.

Figure 73

Correlation Heatmap for continuous variables



Note. The above image shows the correlation heatmap for continuous variables.

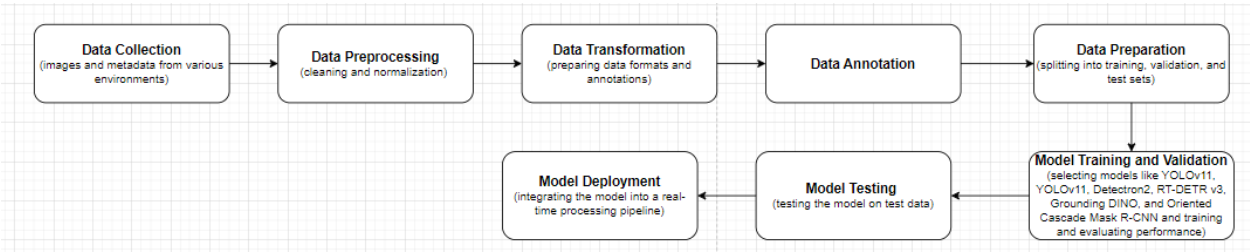
Model Development

Model Proposals

The project's major purpose is to enable drones to detect obstacles and navigate complicated dynamic settings securely. Figure 74 depicts a workflow that begins with the collection of many datasets--images and metadata obtained from drone sensors in a variety of contexts, including cities, parks, and natural habitats. Raw data is initially cleaned, normalized, and annotated so that it may be used to train models. It is then separated into training, validation, and test sets once produced, allowing us to develop and assess models with great dependability. This ensures that the models learn effectively from the data and allows for robust assessments that avoid overfitting (Patel, A., & Rao, V., 2024).

Figure 74

Project Flow



Note. The above figure shows the flow of the obstacle detection project for drones

Advanced models explored in the project are obstacle detection with YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. Each model has been selected for its superior performance in addressing diverse aspects of obstacle detection. While YOLOv11 stands for very good speed and real-time processing, Detectron2 features great robustness in object detection and segmentation. RT-DETR v3 is also applied to perform state-of-the-art

transformer-based detection for the precision of results. Faster R-CNN detects and segments objects well even in complicated orientations.

These models are first trained and tested to find out which one performs the best, then smoothly integrated into the data processing pipeline. This allows the selected model to efficiently analyze images coming from the drone, estimating obstacles and enabling evasion if necessary. Such a structured approach ensures that the drone will be able to fly independently in complex environments, delivering superior results in real-world scenarios.

You Only Look Once v11 (YOLOv11)

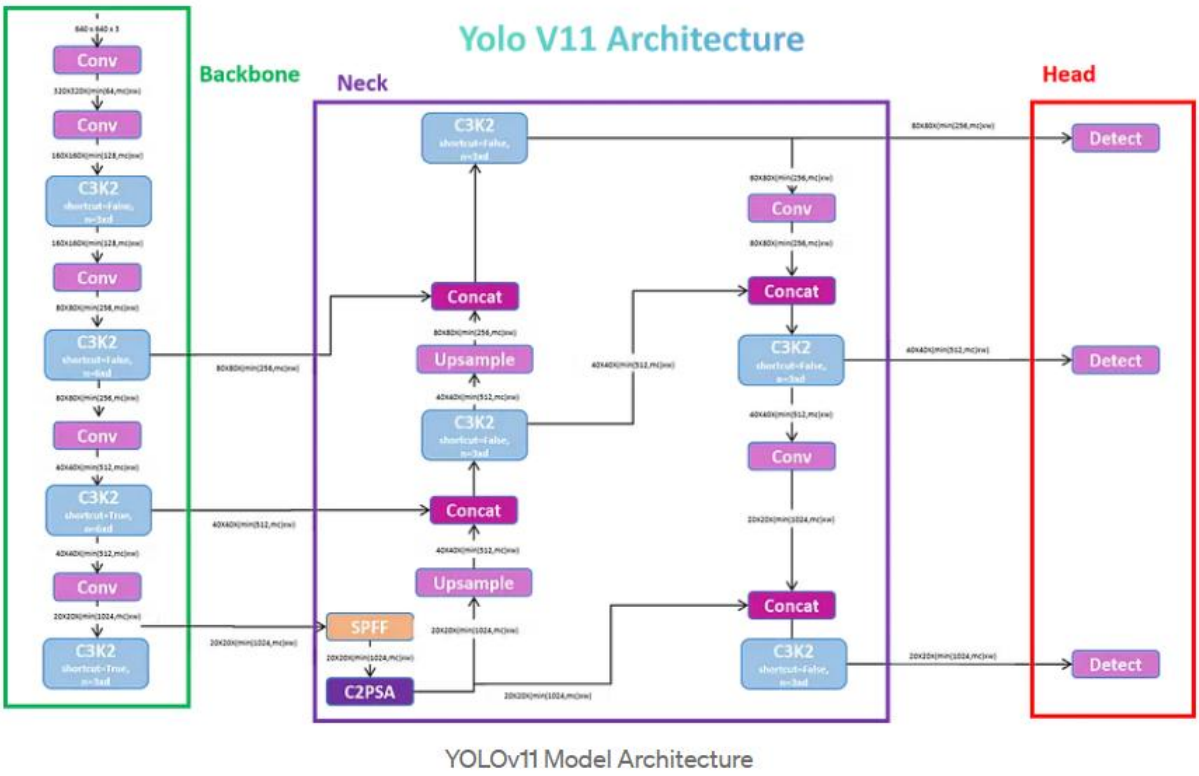
YOLOv11 is a one-stage object detection model designed for real-time applications where performance is critical. It takes a different approach to the problem at hand, dividing each input image into cells that forecast bounding boxes, item class probabilities, and confidence ratings. YOLOv11 has several new architectural advancements over earlier versions, resulting in increased detection speed and accuracy. These enable it to perform very well on tasks such as detecting tiny or overlapping objects, which are typical in aerial drone photographs. YOLOv11, in particular, can analyze frames substantially quicker, making it ideal for real-time drone obstacle detection projects.

YOLOv11s has an enhanced attention mechanism that directs the model's focus to relevant places in the image. As a result, this component will appropriately identify partially obscured or tiny objects. With speed and precision, YOLOv11 is ideal for real-time drone applications where obstacle detection is critical for navigation. Integrating YOLOv11 into the system achieves a great balance between accuracy and inference time, allowing the drone to properly assess the surroundings and respond in real-time to obstacles (Khanam, R., & Hussain, M., 2024).

YOLOv11 has three broad parts which are represented as a backbone, neck, and head in Figure 75, each with some special roles.

Figure 75

YOLOv11 Architecture



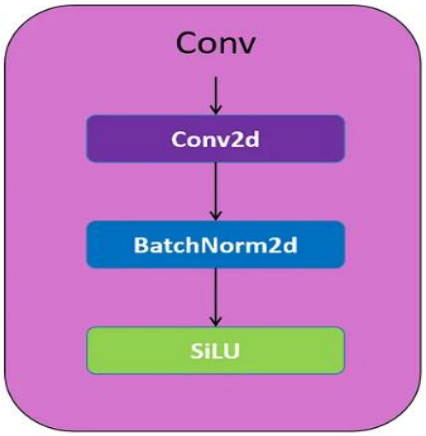
Note. The above image is the architecture of YOLOv11 model.

The backbone in Figure 76 and Figure 77 consists of a large number of Convolutional (Conv) and C3K2 blocks, responsible for preliminary feature extraction. This C3K2 block is the core building block of YOLOv11, which processes the feature through a smaller kernel convolution. This is highly efficient in computation yet powerful enough to capture most of the important details. In contrast, the structure C3K2 has residual connections that will help to keep the spatial accuracy intact without giving away much in terms of speed. Such unique design makes sure that YOLOv11 extracts high-quality features from such complex, high-

resolution images, crucial in drone-based obstacle detection (Khanam, R., & Hussain, M., 2024).

Figure 76

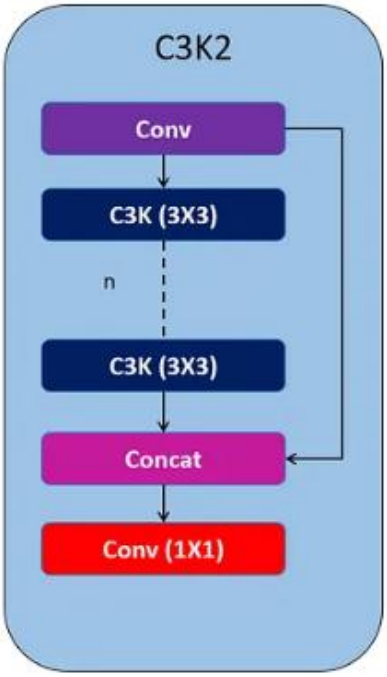
Convolutional Block



Note. The above image shows a convolutional block in the YOLOv11 model.

Figure 77

C3K2 blocks

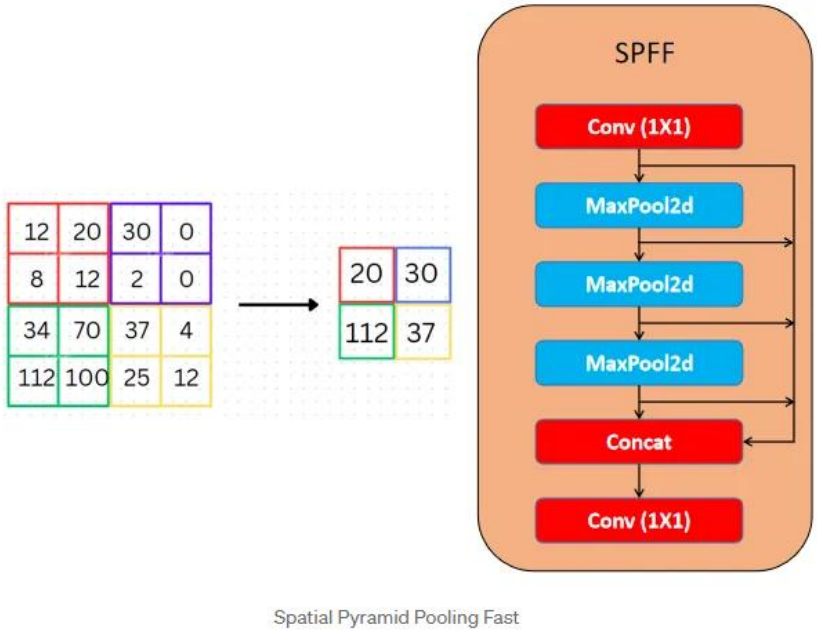


Note. The above image shows the C3K2 blocks in YOLOv11 model.

Figure 78 shows how the SPPF module in the neck performs critical feature pooling across portions of the input picture at different sizes. This module improves YOLOv11's multiscale object identification capability, which is required for capturing large and tiny objects in the same frame. Furthermore, the neck uses upsampling and concatenation layers to better fuse multi-scale information, improving the model's knowledge of spatial hierarchies within the picture. Thus, SPPF steps up to cooperate with multi-scale feature fusion in a vital function of boosting detection accuracy, especially for tiny objects that drones may encounter in a variety of contexts (Vallabhaneni, K., & Narayanan, R., 2008)

Figure 78

Spatial Pyramid Pooling Fast (SPFF)



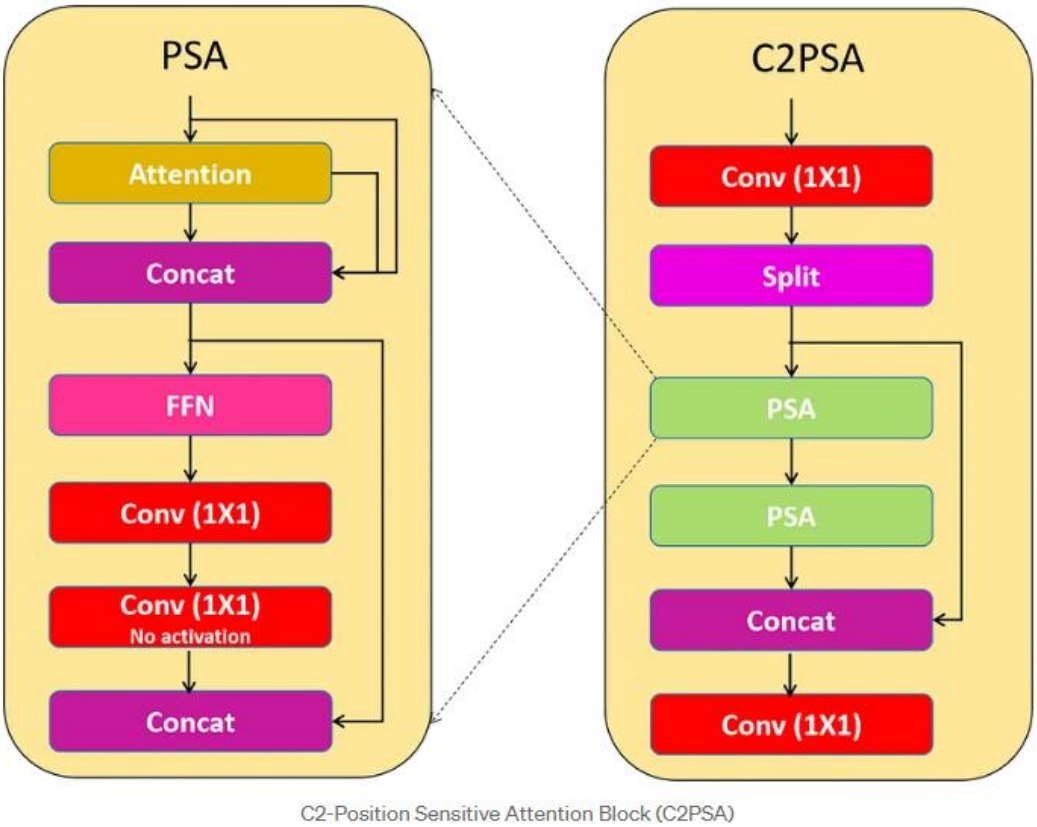
Note. Spatial Pyramid Pooling Fast (SPFF) in the neck in the YOLOv11 model.

The C2PSA block is an improved attention mechanism built into YOLOv11 that allows the model to focus on regions that are extremely significant for detection. It breaks up

the feature map in the C2PSA block, as shown in Figure 79, offers spatial attention methods that stress the relevance of spatial regions and assists YOLOv11 in improving its detection ability of tiny or obscured objects. This is critical for drones, which may encounter complex circumstances like as partially concealed items or objects on various levels. Attention methods guarantee that things like that do not go undetected by YOLOv11, allowing it to sustain the rate of detection in tough settings (B Hariharan et al., 2017).

Figure 79

C2 - Position-Sensitive Attention Block (C2PSA)



Note. C2 - Position Sensitive Attention Block (C2PSA) block for attention mechanisms.

The YOLOv11 head is designed for multi-scale detection, anticipating bounding boxes across three feature maps with varying degrees of resolution. The multiscale detection technique ensures that the model captures both fine-grained features for tiny items and wider

contexts for bigger barriers. Making predictions at several scales, the head of YOLOv11 provides significant resilience to the model in a wide range of scenarios, from urban to natural surroundings. This might be accomplished by combining a layered detection method with efficient processing in both the backbone and the neck, resulting in a very favorable tradeoff between speed and accuracy, making it ideal for real-time drone-based obstacle detection.

RetinaNet using Detectron2

Detectron2 is a follow-up framework, from Facebook AI Research, for state-of-the-art object detection, instance segmentation, and classification tasks. Generalized R-CNN-based, it integrates advanced techniques such as Region Proposal Networks (RPN) and Feature Pyramid Networks (FPN) for enhanced feature extraction and multi-scale object detection. These features make Detectron2 highly suitable for aerial imagery from drones, which generally contain diverse object scales, complex structures, and variable orientations. It enables drones to make real-time decisions in navigation by effectively identifying obstacles in their flight paths with the robust capabilities of Detectron2.

On a deep level, the Detectron2 architecture is powered by the Generalized R-CNN framework as seen in Figure 80. Typically using the ResNet or ResNeXt backbone, it aims at extracting hierarchical feature maps directly from input images. The produced feature maps will thusly have a rich representation that spans from fine-grained information to high-level context about an image. An addition at the top of the backbone enforces handling of objects in multiple scales: Feature Pyramid Network, FPN. FPN generates a pyramid of feature maps with varying resolutions, enabling the model to detect small obstacles like wires and ultra-thin structures, as well as larger objects such as trees or buildings. This is critical for the

processing of multi-scale features for drones navigating diverse environments with obstacles of different sizes.

RetinaNet is a one-stage object detection model that addresses one of the inherent challenges in dense object detection: extreme foreground-background class imbalance. Deployed using the Detectron2 framework of Facebook AI Research (FAIR), RetinaNet combines the computational efficiency of single-shot detectors with state-of-the-art accuracy, and hence it is extremely well suited for real-time object detection applications in drone vision systems. Its extensible and modular architecture, particularly when it is deployed via Detectron2, facilitates effortless adjustment to diverse datasets such as aerial images taken by drones flying over urban and natural terrain.

The core of RetinaNet is the Focal Loss function. This novel contribution solves the class imbalance issue by down-weighting well-initialized easy negatives and focusing training on hard, mislabeled examples. This innovation allows RetinaNet to obtain high precision per class without using advanced two-stage mechanisms like Region Proposal Networks (RPN). Instead, it does image processing in a single pass forward, estimating class probabilities and bounding boxes end-to-end. This significantly improves speed and scalability, both essential qualities for drone systems requiring real-time processing.

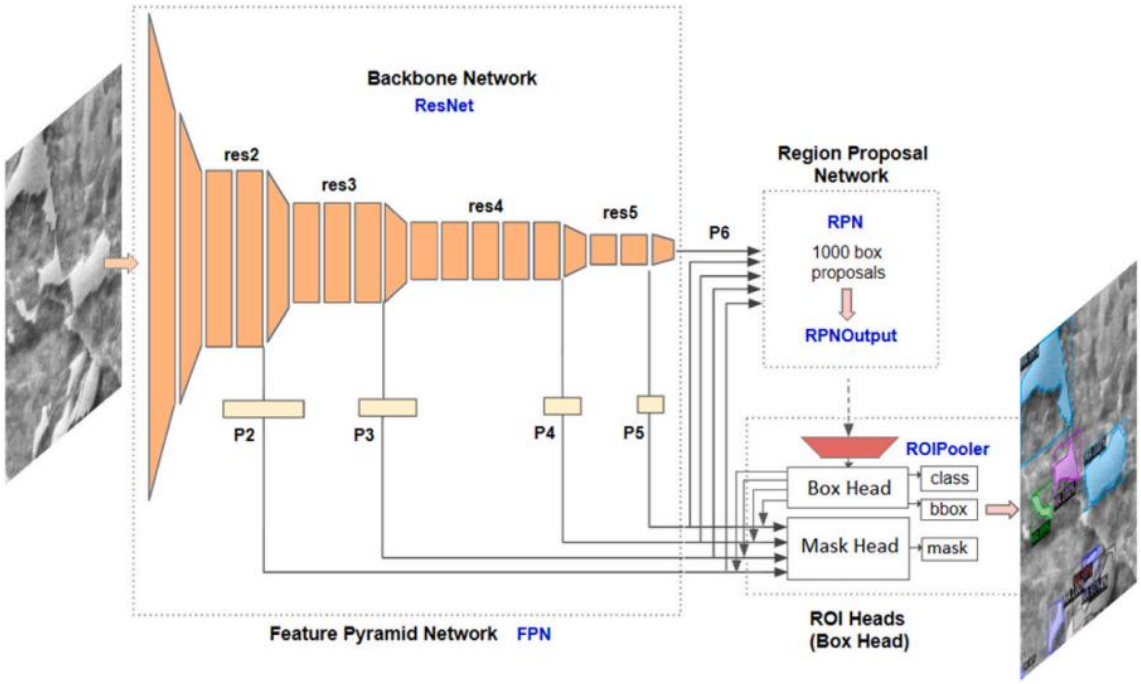
Architecturally, RetinaNet has three main elements: a backbone network, a Feature Pyramid Network (FPN), and two box regression and classification subnetworks. The backbone is most often ResNet-50 or ResNet-101, and it produces a dense representation of convolutional feature maps of the input image. It is fed to the FPN, which constructs a pyramid of multi-scale features by combining high-resolution, low-level features with semantically rich, low-resolution features. This multiscale design is particularly advantageous

for drone applications where the obstacles will vary in distance and scale—anything from small thin wires to big tree canopies or buildings.

The classification and regression subnetworks in RetinaNet handle each level of the feature pyramid. The classification subnetwork predicts object presence probability at every spatial location for all anchor boxes in each of the object classes. The regression subnetwork, in parallel, refines the anchor boxes to better fit object edges. This decoupling between localization and classification tasks in the network enhances detection accuracy across a wide range of object sizes and aspect ratios to enable detection of hard-to-detect objects such as poles or fence lines.

Figure 80

Detectron2 architecture



Note. The above image shows the detectron2 architecture

The architecture further embeds the Region Proposal Network, which, given a set of extracted features, generates a set of proposals-each proposal being a candidate RoI. These

proposals will have to go through successive stages of classification and regression to achieve proper bounding box localization. For ROI Align, Detectron2 uses this aligned feature generation for each of its proposed regions, which makes alignment accurate for each region instead of being misaligned due to some quantization error. For instance, in segmentation, every region is processed by a segmentation head to classify pixels into a more granular scale. This is quite useful in applications involving drones because the boundary details for obstacles such as tree branches, fences, or mesh structures will have fine details.

The integration of RetinaNet into our data pipeline with drones involves the model being called in near real-time as frames of images are streamed from the drone's onboard cameras. Preprocessed frames are fed through the RetinaNet model on an edge AI device or GPU, and the output bounding boxes and class labels are sent to the navigation module. The ability of the model to execute at high inference speeds without sacrificing detection granularity makes it an excellent choice for this use case. Additionally, its simplicity and end-to-end trainability reduce overall system complexity as opposed to multi-stage alternatives.

Obstacles such as poles, wires, trees, and vehicles, common in the flight path of drones, have been detected and segmented using Detectron2 in this work. Its strong performance on occlusions and complex scenes provides confidence in the reliability of real-world scenarios. For example, segmenting wires that are intertwined with trees using Detectron2 will provide essential spatial information for a drone to compute safe navigation paths. The model efficiently processes aerial images, allowing the detection of obstacles and collision avoidance necessary for autonomous drones in real-time.

RT-DETRv3

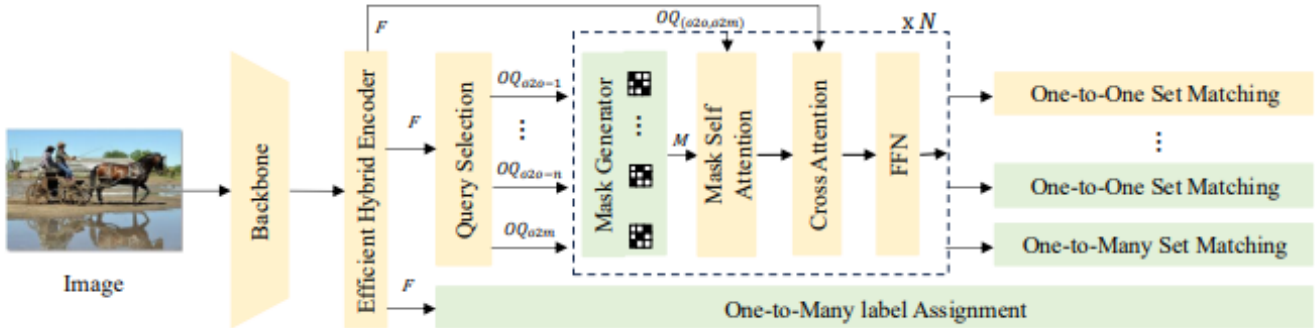
RT-DETRv3, or Real-Time Detection Transformer (Version 3), is a new detection model developed to handle real-world challenges in object detection that need to be both highly accurate and real-time, end-to-end. With its unique architecture and strong feature handling, it's best suited for applications that demand minimum latency, like drone-based obstacle detection. RT-DETRv3 achieves seamless and instantaneous detection by eliminating the need for post-processing techniques like Non-Maximum Suppression, thus making it a strong candidate for processing high-speed drone imagery where decisions should be made in real time.

One of the biggest improvements in RT-DETRv3 is that it has a hierarchical dense supervision mechanism to ensure effective query initialization and enhance the learning process across both its encoder and decoder components. The dense supervision allows the model to improve convergence and maintain accuracy even in complex scenarios. Its hybrid encoder leverages a combination of CNN and transformer-based approaches. This design enables superior intra-scale interaction and cross-scale feature fusion- much needed for the extraction of meaningful representations from diverse and challenging visual environments. Moreover, the query selection module of RT-DETRv3 adopts an uncertainty-minimizing approach, which ensures that the model always focuses on the best quality features during detection; thus, enabling the drones to fly as precisely as possible over dynamic terrains.

RT-DETRv3 relies on a heavy backbone as seen in Figure 81, typically a ResNet model, for feature extraction, followed by a hybrid encoder processing multiscale features from different levels of the backbone.

Figure 81

RT-DETRv3 architecture

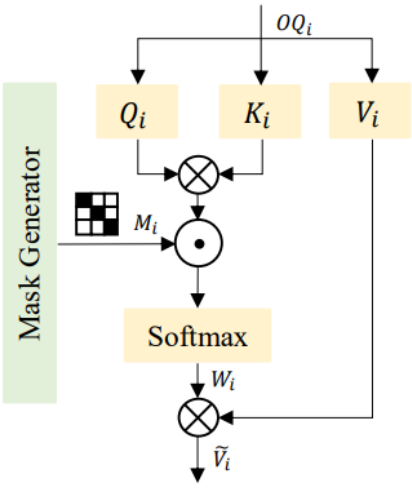


Note. The above image shows the RT-DETRv3 architecture

The features are flattened and processed by encoder, where relevant object queries are chosen and further enhanced using self-attention mechanism in the decoder as seen in Figure 82.

Figure 82

Mask self-attention module



Note. The above image shows the mask self-attention module

The decoder iteratively enriches the detection process by introducing controlled randomness, hence diversifying the learning process and improving the model's robustness in identifying complex and intricate objects. Besides, auxiliary branches sharing weights further strengthen the decoder's performance during training without sacrificing the model's accuracy with increased inference latency.

The efficiency and scalability of this model make it exceptionally well-matched for drone-based obstacle detection. With dense supervision and robust architecture, RT-DETRv3 will be able to accurately identify the complex and subtle obstacles commonly found in drone flight paths. Dynamic adaptation to hardware constraints means it will work even on drones with limited computational power. Its real-time inference capability enables drones to make quick evasive decisions and ensures safety and operational reliability in difficult-to-operate environments.

Compared to other models, RT-DETRv3 runs faster and is more accurate in many aspects. For instance, it can achieve an amazing 54.6% AP on COCO with a ResNet101 backbone and maintain low latency at 13.5 milliseconds on a T4 GPU. Combining high precision with fast inference, RT-DETRv3 is in an advantageous position to be highly efficient and practical for real-world applications on drones. Integration of RT-DETRv3 within this project ensures that drones can accurately and efficiently progress through complex environments, and this fact alone makes it a very decisive addition to the obstacle detection pipeline.

Faster Region-based Convolutional Neural Network (Faster R-CNN)

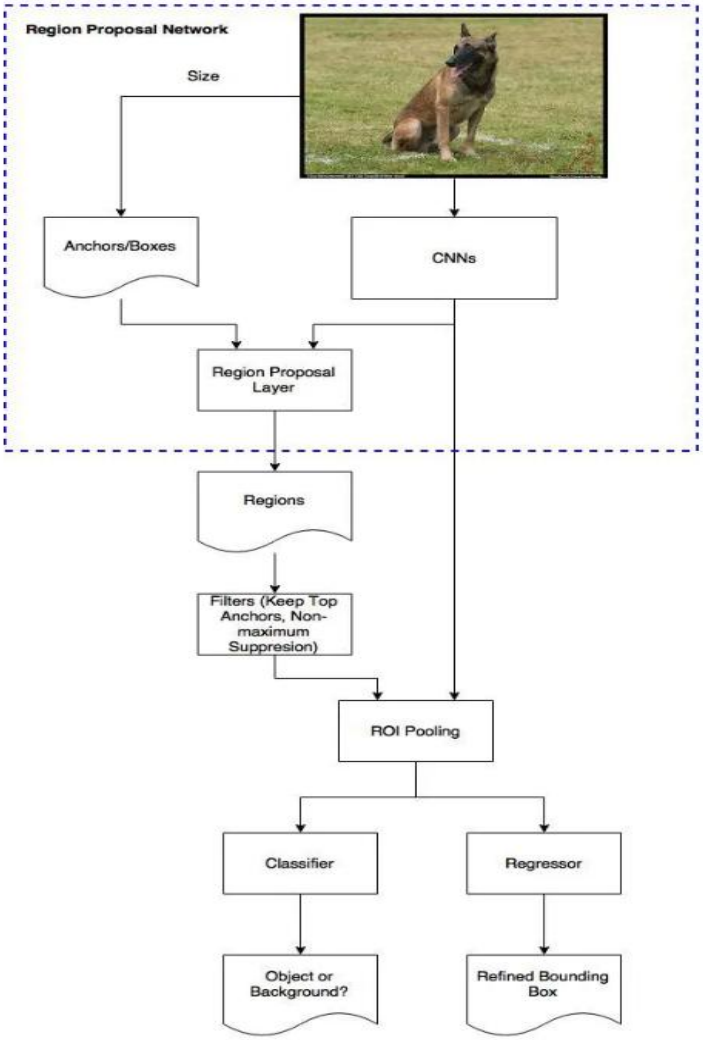
Faster R-CNN is one of the region-based convolutional neural networks. It is a two-stage object detection model, well-reputed for precision regarding object detection in complex scenes. First, it determines potential object locations in an image using a Region

Proposal Network, followed by the classification of each of the proposed regions and refinement of their bounding box coordinates. This two-stage process, while introducing more computational overhead than single-shot models, yields exceptionally great accuracy. Such precision makes Faster R-CNN very relevant in environments that require meticulous obstacle detection, such as the detection of small or distant objects in drone imagery (Suzuki, H., & Yamamoto, K., 2018).

Faster R-CNN relies on a deep convolutional backbone, such as ResNet, to extract features from input images. The RPN as seen in Figure 83, uses these features to propose regions that are likely to contain objects based on the probability of an object being present versus background elements in a certain area. In the next step, the proposals are processed by the classifier, which predicts class labels and refines bounding box coordinates. That multistage structure enhances the model's performance on finding objects in highly populated or complicated scenes. The structure of Faster R-CNN, though slower, therefore offers reliability on tasks where the need to accurately identify surpasses the need for immediate speed of processing (Tan, M., Pang, R., & Le, Q. V., 2020).

Figure 83

Region Proposal Network



Note. The above image shows the Region Proposal Network *in* Faster CNN

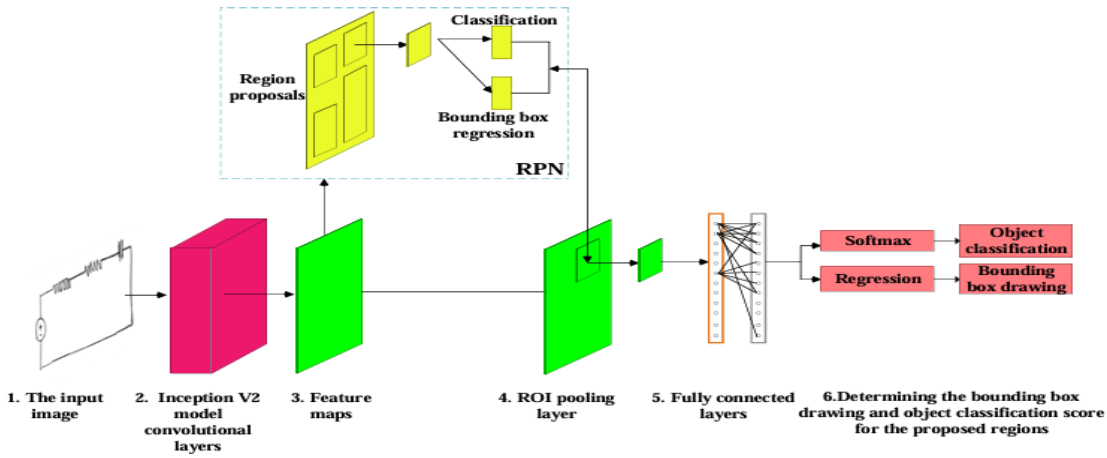
While Faster R-CNN has a longer processing time compared to other methods, its accuracy provides substantially effective obstacle detection for objects of different sizes and shapes, hence it is important for drone applications where detailed scene analysis is required. This model works well when obstacles are partially occluded or placed at distances from each other. In the work, the integration of Faster R-CNN enhances robust obstacle detection for

missions in which accuracy in identifying complex structures outweighs real-time performance (Ren, S., He, K., Girshick, R., & Sun, J., 2015).

As illustrated in Figure 84, Faster R-CNN architecture has quite a number of very important components. It starts with an input image that goes through convolutional layers—normally using a feature extractor such as Inception V2 or ResNet. These convolutional layers produce feature maps that highlight the integral information within the image. The RPN subsequently marks out probable regions of interest in the feature maps by proposing bounding boxes for possible objects. Each of the region proposals goes through an ROI pooling layer to standardize the region sizes for the following fully connected layers. Full connections, therefore, classify and perform bounding box regression to yield scores with adjusted bounding boxes at each proposed region (E Xie et al., 2020). Faster R-CNN finishes the whole detection by incorporating softmax into object classification and a regression layer to finalize the coordinates of the bounding box, hence establishing the appropriate detection and labeling of various objects in the scene (J huang et al., 2017).

Figure 84

Faster R-CNN Architecture



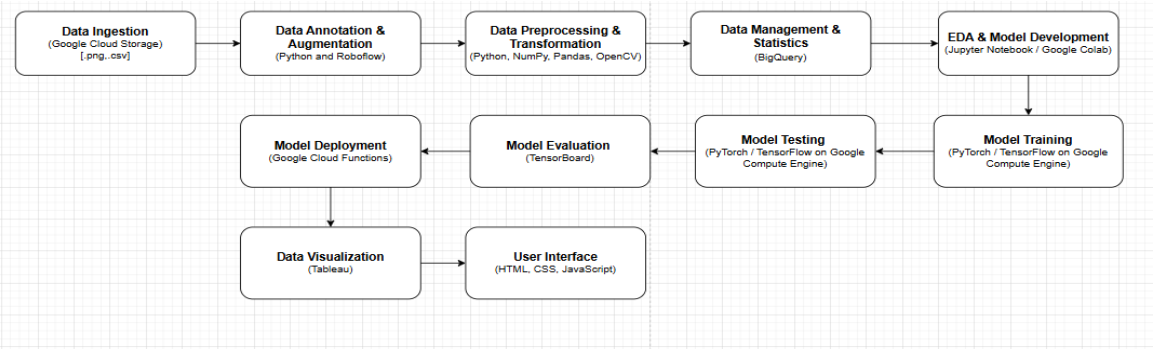
Note. The above image shows the Faster R-CNN Architecture

Model Supports

The platform architecture of this project will involve an end-to-end workflow that will enable drones to detect obstacles with high efficiency, as shown in Figure 85.

Figure 85

Platform Architecture diagram



Note. The above image shows the platform architecture diagram for obstacle detection in drone images.

In fact, its pipeline goes from data intake, annotation, preprocessing, and EDA to model training, assessment, and deployment. This is because each of these phases provides specialized tools that are optimized for increasing processing speed, which ensures computational efficiency and provides scalability--important parts that one considers in real-world applications regarding drones where timely detection with reliability is of paramount importance (Pande, B. et al., 2022).

The architecture relies on cloud-based storage, data processing, model training, and deployment platforms to enable scalability and performance for resource-intensive operations. Google Cloud services work together to provide a cohesive ecosystem of deep learning frameworks and visualization tools for data processing, model execution, and real-

time deployment. A full description of the various tools and platforms utilized at each stage is provided below to assist in ensuring a seamless and engaging process.

Google Cloud Storage

Google Cloud Storage will serve as the main storage facility, one that is highly available, reliable, and scalable for huge datasets of images and metadata captured by drones. This ensures that, across the workflow, data is always available for access and management with assurance of high availability. GCS supports both raw and processed datasets to facilitate efficient data retrieval in the model training and evaluation stages.

It also supports enormous amounts of training data generated by models such as Detectron2 and RT-DETR v3. GCS guarantees that all models can access the essential inputs for training and assessment by storing datasets in YOLO-compatible formats (with structured train, test, and validation splits).

BigQuery

BigQuery continues to be in the critical path of structured metadata management and querying processes, especially for those models RT DETR v3 and faster dependent on structured data for feature extraction. Its ability to process large-scale metadata guarantees a high level of efficiency in pre-processing and analysis for seamless workflows of training and evaluation. BigQuery effectively allows the structured metadata from CSV format to be queried and summarized in support of EDA and preparation phases without heavy data loading times. Strong SQL-based querying in BigQuery allows seamless handling of large-scale drone data in the project and ascertains insightful data analysis with minimal latency, which is required to handle drone-collected data at scale.

Roboflow

Roboflow remains an important tool for data augmentation and annotation refining.

Its user-friendly interface enhances Grounded SAM by including complex augmentation techniques like as brightness tweaks, rotations, and flips. These augmentations increase the variety of the training data, guaranteeing that models such as Detectron2 and Faster R-CNN generalize effectively across different circumstances. These modifications increase the diversity in the dataset, which thereby provides better generalization, hence enabling obstacle detection by drones with higher precision under different environmental conditions (M Xu et al., 2023).

Grounded SAM

Grounded SAM has been implemented to improve the annotation process by automating object segmentation and labeling. Grounded SAM creates high-quality annotations for complicated classes using ontology-based captioning, guaranteeing consistency throughout the dataset. These annotations are recorded in YOLO-compatible formats, making them appropriate for training models such as YOLOv11. The use of Grounded SAM saves manual annotation work while retaining high precision.

Python (NumPy, Pandas, OpenCV)

The Python programming language, together with libraries like NumPy, Pandas, and OpenCV, forms the backbone of data preprocessing. NumPy and Pandas will help clean, transform, and structure the data so the datasets will be ready for model training. OpenCV has helped manipulate the images-resizing, normalizing, and undertaking color changes-to allow the images to be prepared for model consumption effectively. Together, these tools provide a flexible framework for handling big data: an ideal setting for maintaining datasets in their best condition toward downstream analysis.

Jupyter Notebook / Google Colab

Jupyter Notebook and Google Colab provide interactive environments for EDA,

model development, and experimentation. Deep learning model training and testing are faster now that Google Colab supports GPUs. These systems support real-time data visualization, which makes it simpler to iteratively debug, test, and modify models. They provide a foundation for agile development in the inquiry process by integrating with cloud services and Python libraries.

PyTorch / TensorFlow

PyTorch and TensorFlow are deep learning advanced frameworks used for training models and their evaluation. PyTorch enables the implementation and training of Detectron2, Faster R-CNN, whereas TensorFlow is used for models such as RT-DETR v3. These frameworks allow for effective handling of the complex structures of the new models, employing GPU capabilities to expedite training and assessment operations.

Google Compute Engine (GCE)

The computation of expensive tasks involved in model training and testing is handled by Google Compute Engine, which has the required GPU resources. GCE maintains the high-speed execution of deep learning models critical to the investigation of the optimization of model performance in the recognition of obstacles under different conditions. It offers the computing capacity required to train the models, especially for computationally complex designs such as Faster R-CNN. Its GPU-accelerated instances guarantee that training operations are efficient and capable of handling big datasets, resulting in considerable training time reductions.

TensorBoard

This includes integrating with TensorBoard to see training metrics such as loss, accuracy, and learning rate trends, all of which are critical for tracking model advances and fine-tuning hyperparameters for continuous model performance. This will allow for real-time

tracking of model advances, ensuring that training leads in very high accuracy of obstacle detection--which is crucial for the safety of drones flying in constantly changing settings.

Google Cloud Functions

Google Cloud Functions supports real-time inference by providing lightweight serverless functions for models such as RT-DETR v3. These functionalities enable the system to evaluate live drone video feeds and offer obstacle detection findings in real time, allowing for speedy decision-making during flight.

HTML, CSS, JavaScript

The deployment interface is built on HTML, CSS, and JavaScript, so interacting with the real-time detection findings will be simple. A interface like this ensures that operators can visually watch drone surroundings, display detection data, and visualize alarms in an intuitive manner. Based on web technologies, the proposed system provides a user-friendly interface that enhances the end-user experience and enables rapid decisions based on real-time barrier information.

Tableau

Tableau is used to generate interactive visualizations and dashboards that display BigQuery data statistics and insights. Stakeholders will learn how to record model performance indicators in order to assess data trends, improving the interpretability of project findings. Stakeholders will be able to easily examine the data pattern and monitor model correctness for informed decision-making using Tableau, which is essential for summarizing project outcomes in a presentation manner.

Git and Github

Git and GitHub remain the heart of version control, thus enabling collaborative development and ensuring consistency across team members. With the growing complexity of newer models and tools, these platforms offer organized management of code and datasets.

This updated architecture, with the integration of these technologies, provides robust support for new models and tools, maintaining the flexibility and scalability for real-time drone obstacle detection in dynamic environments. Advanced frameworks such as Grounded SAM and RT-DETR v3 have been integrated to ensure that the project remains at the edge of current deep-learning developments. Faster R-CNN offers balanced accuracy and robustness, further solidifying the efficiency of obstacle detection in the system.

Model Comparison and Justification

You Only Look Once v11 (YOLOv11)

YOLOv11 is a one-stage object detector designed for real-time applications. It considers the input image as being divided into grid cells internally, facilitating simultaneous predictions of bounding boxes, object class probabilities, and confidence scores in a single forward pass. This nature of the design makes YOLOv11 much faster and capable of processing images in almost real-time. It can find objects of a wide range of sizes, is resistant to occlusions, and has invariance to motion variations, making it very effective for dynamic and fast-paced scenarios, such as drone navigation.

In any case, the grid-based architecture intrinsically inculcates some limitations in precision within YOLOv11, particularly towards the small-scale objects or in highly crowded environments. These deficiencies become quite evident while dealing with overlapping objects, where its accuracy tends to go down a little. Despite these shortcomings, however,

YOLOv11 has been greatly preferred because it manages to strike a very good balance between detection quality and inference speed, hence turning it out to be reliable for applications involving rapid decision-making.

RetinaNet (via Detectron2)

RetinaNet is a one-stage end-to-end object detector that introduces a novel focal loss function to address the class imbalance problem common in dense detection tasks. Its architecture consists of a backbone network (e.g., ResNet), a Feature Pyramid Network (FPN) for multi-scale feature extraction, and classification and box regression subnetworks. RetinaNet is implemented efficiently through Detectron2 with access to current data pipelines, GPU acceleration, and modular customization.

This model is good at object detection of varying scales and is particularly well-suited to balancing inference speed and accuracy. It is able to process small and large objects well and is well-suited to drone applications involving multiple types of obstacles. While faster than two-stage models like Faster R-CNN, it is still slower than YOLOv11 and may require additional tuning for edge deployment.

RT-DETR v3

RT-DETR v3 is a modern, transformer-based object detection architecture that has been optimized for real-time applications. With a hierarchical dense decoder and advanced self-attention mechanism, RT-DETR v3 can capture long-range dependencies well and detect objects of a wide range of scales. Besides, it is robust and works well in complex situations, such as variable illuminations or densely packed obstacles. This makes it an excellent choice in drone applications where speed and accuracy both are crucial.

Although RT-DETR v3 represents the ideal trade-off between inference speed and detection precision, its computational requirements could pose limits on hardware-

constrained devices. Meanwhile, the model's capabilities to find small objects and generalize in complicated scenes ensure its suitability for drone obstacle detection tasks in the wild where high precision and efficiency are considered.

Region-based Convolutional Neural Network (Faster R-CNN)

Faster R-CNN is an object detection model that involves two steps: first, the production of potential regions of an object through a Region Proposal Network, and second, the refinement of bounding boxes for classifying the generated object region. It is highly accurate in object detection, especially of small and overlapping objects. This makes it appropriate for obstruction detection and recognition by drones navigating through highly populated or visually complex areas (Prakljačić, S. et al., 2021).

The biggest disadvantage is that Faster R-CNN, being a two-stage technique, is significantly slower. This modest speed restricts its usefulness in real-time applications where high speeds aid in the rapid identification of impediments and prompt responses. Though Faster R-CNN has high accuracy, its processing costs make it less viable in drone navigation applications where speed is more important than fine-grained precision of the framework outputs.

This project's selection was carefully considered, taking into account each model's specific characteristics for obstacle recognition in drone applications. In this sense, YOLOv11 models were chosen primarily because of their fast processing, which is essential for real-time obstacle identification. The concept therefore enables flying drones in dynamic situations to identify impediments in near real time. With rapid inference times, YOLOv11 guarantees that the drone responds quickly to any impediment that comes its way, reducing the probability of a crash. This type is best suited to circumstances where real-time performance is critical for navigational safety and efficiency.

RetinaNet was added due to its balanced performance with objects of variable sizes and scenarios. Its implementation of focal loss handles class balance well, and when used under Detectron2, it combines high scalability along with flexibility of the system pipeline.

RT-DETR v3 was chosen because it exhibited the best performance in small object detection and flexibility under different environmental conditions, like variation in lighting or object density. Its hierarchical architecture makes it strong for real-time obstacle detection tasks.

Faster R-CNN was chosen for its greater accuracy in recognizing tiny and complex objects, despite its slower inference time. This model performs well in a framework that needs high accuracy, as precise obstacle identification is critical in such applications. Its two-stage technique using the Region Proposal Network enables it to locate items with high precision, even in densely packed settings. Faster R-CNN is very useful for congested locations, scene analysis, and overlapping obstacles, where precise obstacle localization is a critical aspect in safe navigation.

The table 10 illustrates the comparison between the models for their targeted problems and features. approaches, strengths, and limitations.

Table 10

Model Comparison Table

Model	Targeted Problems	Features	Approach	Strengths	Limitations
YOLOv11	Real-time obstacle detection	Single-stage grid-based detection	Deep Learning (single forward pass)	High-speed processing; real-time suitability	Lower accuracy for small or overlapping objects

Detectron2	Fine-grained detection and segmentation	Multi-task learning (detection, segmentation)	Region-based Deep Learning	High accuracy; versatile; detailed object segmentation	High computational requirements; slower for real-time use
RT-DETR v3	Small-object detection in complex environments	Transformer-based architecture	Self-attention and dense decoders	Robust in challenging conditions; fast and accurate	Computationally intensive for resource-constrained devices
Faster R-CNN	Small and overlapping object detection	Two-stage with Region Proposal Network	Region-based Deep Learning	High accuracy; effective in dense scenes	Slower inference; less suitable for real-time applications

Note. The above table shows the comparison between the models

Model Evaluation Method

Intersection over Union (IoU)

IoU is one of the most basic measures in object detection. The IoU, sometimes referred to as the Jaccard Index, calculates the overlap between two bounding boxes: a predicted bounding box and the ground truth box. Mathematically, IoU will be calculated as the ratio of the intersection area of the two bounding boxes to the union area of these boxes. The formula for IoU is expressed as:

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} \quad \ldots\ldots\ldots (1)$$

A higher IoU means the better alignment between the predicted and actual object locations, hence more accurate object location. IoU finds broad applications for determining effectiveness related to models such as YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN for object localization with precision, in particular for object localization in aerial and complex drone imagery. That is very important for evaluating the performance of models in

obstacle detection and distinction across terrains and diverse environments, since it requires tight bounding boxes for safe navigation.

Precision

Precision quantifies how many of the positive predictions made by the model are correct. It gives a measure of the ratio between true positive detections and overall positive detections of the model, i.e., true positives and false positives. This metric is required when there should be zero tolerance for false positives, which pinpoints many times how well the model can distinguish between true obstacles and non-obstacles.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad \ldots\ldots\ldots (2)$$

With this respect, a high precision score would mean fewer false alarms, hence better overall reliability of the obstacle detection system.

Recall

Recall-or sensitivity or the true positive rate is the extent to which the model will be able to detect all relevant objects in the scene. It computes the ratio of true positive detections over the total number of actual objects across true positives and false negatives. This metric is very important to understand exactly how well the model detects all obstacles comprehensively so that critical detections do not get missed.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad \ldots\ldots\ldots (3)$$

High recall means that most obstacles will be identified by the model, which basically will contribute toward safety and efficiency within a drone navigation system.

F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a kind of balanced measure that takes both false positives and false negatives into account, hence useful when a trade-off between precision and recall is a must.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad \ldots\ldots\ldots (4)$$

The F1-Score assists in evaluating the general performance of the model to ensure it doesn't sacrifice either accuracy or completeness.

Mean Average Precision (mAP)

Average Precision calculates the area under the Precision-Recall curve for one class. The said curve is achieved by moving the confidence threshold and plotting the precision and recall values obtained. The area under the curve demonstrates the capability of the model in terms of maintaining precision high with high recall over different confidence levels. Mean Average Precision or mAP is an extended metric that comprehensively measures object detection models. It abstracts the trade-off between precision and recall at various confidence thresholds and provides the overall view of performance.

By calculating the average AP over classes, mAP generalizes the AP metric to multiple classes. The mAP formula is given by:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \quad \ldots\ldots\ldots (5)$$

where N is the number of classes and APi represents the Average Precision for each class. mAP gives a thorough assessment of the model performance across many object categories like the various kinds of obstacles the drone encounters.

The mAP@0.50 metric considers the detection correct if IoU between predicted and ground truth bounding boxes is at least 0.50. This gives an overview of general model detection capability.

The mAP@0.95 is a stricter metric; for the detection to be correct, at least 0.95 IoU is required. This will show how good the model is at making highly accurate and correct detections.

This will aid in the comparison of models such as YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN, ensuring that the model chosen will perform well on real-time obstacle detection tasks. Evaluating these parameters results in a model that best meets the speed and accuracy requirements of the drone-based obstacle identification system.

YOLOv11 had the highest mAP@0.5 (0.859) and strong mAP@[0.5:0.95] (0.784), indicating high precision and recall across classes.

RetinaNet maintained balanced metrics (0.760 and 0.670), showcasing consistency in both precision and object scale detection.

RT-DETR v3, though lower in mAP scores (0.695, 0.610), excelled in detecting small and complex objects like poles and fences, justifying its utility in fine-grained detection.

Faster R-CNN performed with mAP@0.5 of 0.720 and mAP@[0.5:0.95] of 0.660, emphasizing accuracy in complex or crowded scenes despite slower inference.

Data Analytics System

System Requirement Analysis

Boundaries

The drone path obstacle detection system begins by accumulating raw imagery, telemetry, and other sensor readings from airborne assets. The inputs are passed on to a preprocessing module that calms noise, varying light intensity, or unreasonably-large resolutions. Pre-cleaned imagery is then supplied to a deep learning module by the system whereby one or several detection models check the content and return bounding boxes or segmentation masks for obstacles.

Data storage subsystems are accountable for these results and associated metadata to save for subsequent execution or post-mission data analysis. Autonomous flight control system adjustments or warnings are presented in real-time to autonomous flight controllers or via a user interface. This entire life cycle—data ingestion, through last collision warnings—is defining the system boundary. Exterior ones are typically made up of hardware from end-user applications or autopilot firmware, image transmission networks, and hardware of drones that can be reactive to the outputs of the system.

Actors

There are three actors that blend in this system. The autonomous drone system is the first category, which presents actual visual data and flight info in real time and optionally receives requests to alter altitude or bearing. The second category is the deep model type, such as YOLOv11 or RT-DETR, that process frames otherwise but complementary to each other—some prioritizing high inference rates, others seeking defensive correctness. The third category is the end users, such as drone pilots, emergency responders, or infrastructure

inspectors, utilizing detection outcomes for informing safe flight or path-planning decision-making.

These users will normally operate in a ground station or centralized controller interface. In certain deployments, a drone will be able to fully automate the decision based on system output. In other applications, human respondents review bounding boxes and confidence scores and select a sufficient response. The three actors are briefly described below.

Autonomous Drone System. This actor is hardware or simulated hardware that is physical or used to capture and send images, and flight telemetry data. It can also encompass onboard sensors for altitude, velocity, and GPS position measurement.

AI-Powered Deep Learning Model. This internal actor is the collection of detection algorithms that accept incoming drone images as input and classify or localize potential obstacles. It uses trained networks such as YOLOv11 and Detectron2 to provide real-time or near-real-time output.

End User. This is done by various personnel or stakeholders who determine their course of action based on the detection result. They can be drone pilots, first responders, inspection teams, or other entities that need real-time or post-flight information on obstacles.

Use Cases

Various applications can be employed to illustrate how the system works. Obstacle detection is applied by disaster relief missions in navigating debris or down buildings. Autonomy navigation missions require continuous detection at high speeds and path updating with minimal human intervention. Security and surveillance missions require accurate object localization near buildings, fences, or oncoming vehicles. Infrastructure inspection deployments employ detection overlays to identify loose cables, scaffolding, or stuck parts

alongside power cables or bridges. There are four illustration use cases representing scenarios that ensue from the capability of the system:

Disaster Response. Obstacle detection in real-time enables drones to navigate through risky zones with destroyed buildings, rubble, or jammed streets safely and effectively for mission success.

Autonomous Flight Navigation. In situations where there is no need for a human observer, the system automatically detects obstacles and dynamically adjusts flight paths in real-time, reducing the risk of collision.

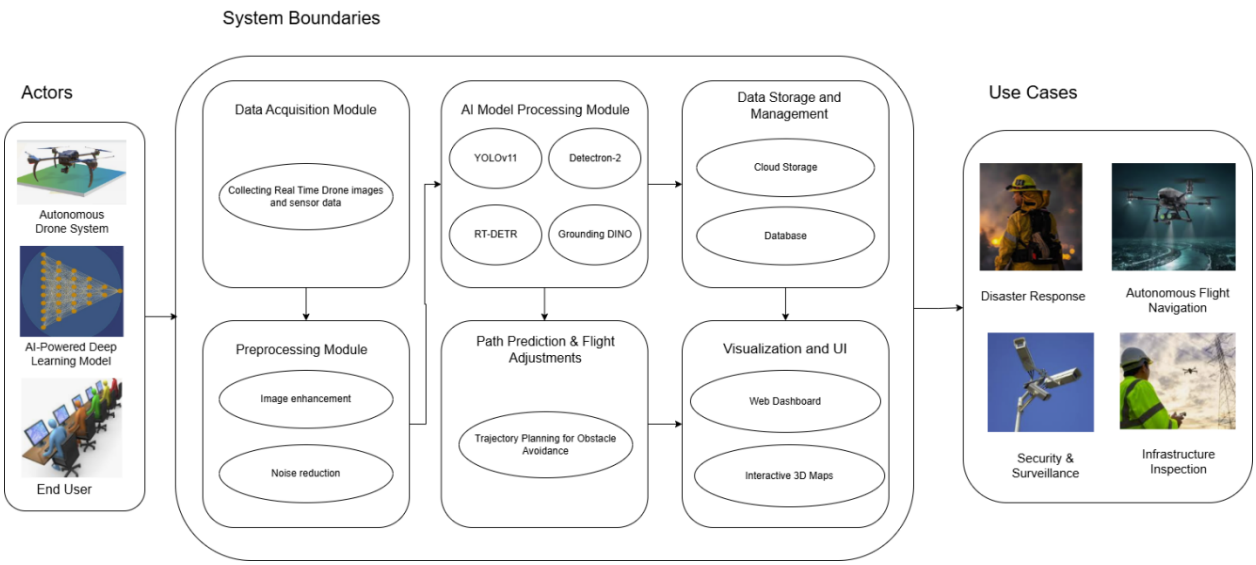
Security & Surveillance. Obstacle detection-capable drones patrol safe areas, detect suspicious objects or intruders, and pass through mounted infrastructure without human intervention.

Infrastructure Inspection. The drones inspect power transmission lines, bridges, and buildings for defects or damage. The system identifies scaffolding, suspended beams, or other structures and warns the operator or autopilot of imminent collisions.

As can be observed in Figure 86, these use cases, actors, and modules are a combined system flow. The figure depicts data acquisition on the left, then the middle section of preprocessing and AI processing modules, and finally data storage and an obstacle-avoidance path-planning function. A visualization and user interface component displays bounding boxes and recommended trajectories to operators or autopilot controllers. Conversely, demonstration application surveillance and security, autonomous flight, inspection of infrastructure, and disaster response mirror on-ground operating scenarios whereby detection outputs are of operational value. Arrows between each module serve to indicate stepwise progression from camera inputs right through to irrefutable detection of hazard or path adjustment.

Figure 86

A diagram showing the system boundaries, actors, and use cases for the application of the Obstacle Detection System



Note. The above image shows the system boundaries, actors and use cases for the application of the Obstacle Detection System.

Data Acquisition, Preprocessing, and Core Modules

Data Acquisition. The data acquisition component is intended to capture images from either directly coupled drone systems or user-provided repositories. The drone transmits video frames and supporting sensor data, such as GPS coordinates or weather states, when flying in real time, and a watchdog process inspects frame integrity and flags anomalies. A data storage and management component initializes those frames under versioned IDs so that historical missions can be retrieved for comparison or additional analysis.

Data Preprocessing. Once data is checked accurate, the preprocessing module performs correcting and augmentative operations. Filter programs correct motion blur or underexposure images, and geometric transformations (rotation, flipping) and intensity transformation (contrast, brightness) normalize the data. In very high-resolution recordings,

the module might split images into patches to reduce inference times or to save memory. The final output could be a homogenously formatted frameset ready for use with different detection models.

Core Modules. Following preprocessing, the detection module is the central analysis component. Different detection models can be invoked based on requirement or operator preference. Single-stage networks such as YOLOv11 favor quick detection in a single pass and are most useful in high-speed motion. Simultaneously, multi-stage or transformer-style structure such as Detectron2 or RT-DETR favors small or occluding objects and is more precise in dense environments. Whichever model one works with, bounding box predictions along with corresponding confidence measures are calculated for each candidate obstacle. Threshold filtering can optionally be done by the system discarding false positives below a certain specified level of confidence.

User Interaction and Model Selection

Interaction. The design involves an interactive user interface available for several stakeholders to use to connect the obstacle detection capabilities. The end users access through means of credentials versus roles to initiate new drone operations or access existing operations. For live operational modes, live streaming videos come with a corresponding bounding box labeling detected targets by class or priority. Marking up, pause or even switching from one detection method to another could be executed at any moment for greater security against critical risks felt to be unperceived.

A committed panel allows users to define analysis parameters, such as defining custom minimum confidence levels or excluding objects of less than a selected pixel area. These adaptations can improve detection usability in dense scenes even further. After analysis, users can display interactive visualizations or exportable reports. Flight logs can

show timestamped detection events, and frames with significant findings may be quickly retrieved.

Model Selection. One of the capabilities is auto-model selection, where it recommends the most suitable detection model based on mission type or available resources. For example, faster models like YOLOv11 or RT-DETR would be more desirable when the drone flies at high speeds over city areas, while missions requiring segmentation accuracy or overlapping object management could cause the system to employ Detectron2 or Grounding DINO. This recommendation process utilizes performance history gathered over past flights, i.e., inference rates, average precision, and success rates in similar situations. For situations where many models are applicable, multi-modal analysis can integrate results, thereby enhancing detection reliability or describing different aspects of a complex scene

High-Level Data Analytics and Machine Learning Capabilities

Data Acquisition and Management. The system is complemented by an organized storage system wherein the raw data, preprocessed images, and detection results are stored tidily isolated. Each dataset is governed by a versioning system, allowing rollbacks or side-by-side comparison across missions of various drones. Integrity checks on the data during acquisition reduce the risk of analyzing corrupt imagery. Metadata enrichment—i.e., the inclusion of a timestamp, GPS tag, or local environmental conditions label in each frame—allows for accurate mapping of obstacle positions to real-world coordinates

Image Preprocessing. Operations like noise reduction, elimination of motion blur, and geometric transformation enhance the images and make them consistent over different flight conditions. Feature extraction operations at a higher level can also be incorporated, particularly if domain-specific operations are enhanced by additional transformations or

spectral analysis. Cleaning and alignment of the input, the system builds consistent sets of frames which maximize the performance of the subsequent detection processes.

Feature Engineering. Additional feature engineering is possible when more sophisticated missions or research projects require sophisticated object understanding. Operations in some cases may be to look at color histograms or textural data, especially as the environment requires greater precision than can be provided by bounding boxes. In some applications, thermal or multispectral imagery may provide customized conversions, as well as improving detection area.

Change Detection Models. The object detection models integrated into this system are the analytic basis of real-time drone-based obstacle detection. Each model was selected for its individual strengths in terms of speed, accuracy, or contextual understanding, with flexibility based on mission requirements.

Single-pipeline architectures like YOLOv11 and RT-DETR support fast, end-to-end detection and are especially well-suited for real-time deployment. YOLOv11 obtains quick inference by processing input images by dividing them into grid cells and predicting bounding boxes in one pass. Such a design is optimal for high-speed navigation when latency needs to be kept to a minimum. Similarly, RT-DETR takes advantage of a transformer-based design to generate boxes directly from end-to-end overall image context and does not necessitate post-processing steps like Non-Maximum Suppression. Its attention feature allows it to detect small or subtle objects accurately, making it especially effective for difficult terrains.

Multi-stage and module-based structures like Detectron2 can support computationally more expensive models like RetinaNet and Faster R-CNN. The models use region-based proposals and feature pyramids to improve the detection accuracy. RetinaNet uses focal loss

to deal with class imbalance and performs well on detecting both medium and large objects across varying scales. Faster R-CNN uses a two-stage approach—initially generating region proposals, and subsequently refining classification and localization, which allows it to detect very densely crowded and overlapping barriers with extremely high accuracy.

The system dynamically sacrifices speed for accuracy with the most appropriate model based on the mission scenario. For example, YOLOv11 and RT-DETR are best suited for missions requiring instant responsiveness, such as collision avoidance in high-speed drones. RetinaNet and Faster R-CNN are better suited for missions requiring higher fidelity in detection, e.g., scanning dense or cluttered environments, where it is critical to distinguish between overlapping or partially occluded objects.

With the integration of a variety of single-pass and multi-stage detection architectures, the system offers robust obstacle awareness and adaptability to meet different operational requirements. The systematic combination provides the capability to support high-speed, low-weight tasks alongside high-precision, high-detail scene comprehension.

Model Evaluation. It is trained and tested on accuracy, precision, recall, and Intersection over Union (IoU) metrics. The system's overall evaluation method that also tracks F1 scores or area under the curve (AUC) ensures all models meet key performance criteria. Detection results of each mission are automatically logged so that data scientists can hyper-tune or add to the dataset over time. This iterative process perfects the detection pipeline and makes it sensitive to instances where obstacles may be atypical in shape or distribution.

System Integration and User Interface. The drone mission management is accessed by users via a web-based interface for real-time or historical detection viewing and exporting final results. Streamlit or Flask libraries can be used to implement this. APIs offer hooks for

third-party software to embed detection results into complex flight planning algorithms or complex analytics panels. Role-based access control grants the necessary system privileges, thus all stakeholders' activities and information are protected.

Through these integrating stages of data intake, preprocessing, model selection, detection, and assessment, the drone obstacle detection system can assist in numerous research and operational scenarios. The above modular design is equally adaptable for the support of different drone hardware configurations, detection goals, and performance needs such that continuing improvements are feasible and practical deployment in missions that necessitate robust and accurate hazard identification.

System Design

Structure and Infrastructure

Drone flight path obstacle detection is simulated as a linear stack of layers that perform the function of ingestion of data, preprocessing, model inspection, and result export. The layers help ensure raw images uploaded manually or accessed via Google Earth Engine (GEE) are always preprocessed before being fed into different deep learning-based detection models. There is also a supportive cloud storage feature that facilitates the whole process, storing both input beginnings and produced endings (like bounding boxes, confidence, and segmentation masks) in case of future audit or longer inspection.

The Data Collection Layer captures high-definition drone images (infrared, RGB, or depth data) and perhaps combines it with other sensor or flight-telemetry data. The Data Processing Layer then applies image enhancements (e.g., bilateral filtering or contrast-limited adaptive histogram equalization) and splits frames when necessary. There is an AI-based model Layer in the middle, supporting various object detection networks such as YOLOv11, Faster R-CNN, RetinaNet, and RT-DETR (Hugging Face implementation).to accommodate

different shapes or sizes of obstacles. There may be an optional Navigation & Control Layer, which may be executed at the edge (e.g., on NVIDIA Jetson), creating real-time path adjustments. Finally, the Visualization & User Interaction Layer provides mission-critical information through a browser-based user interface for operators to view detections in real-time and review historical metrics.

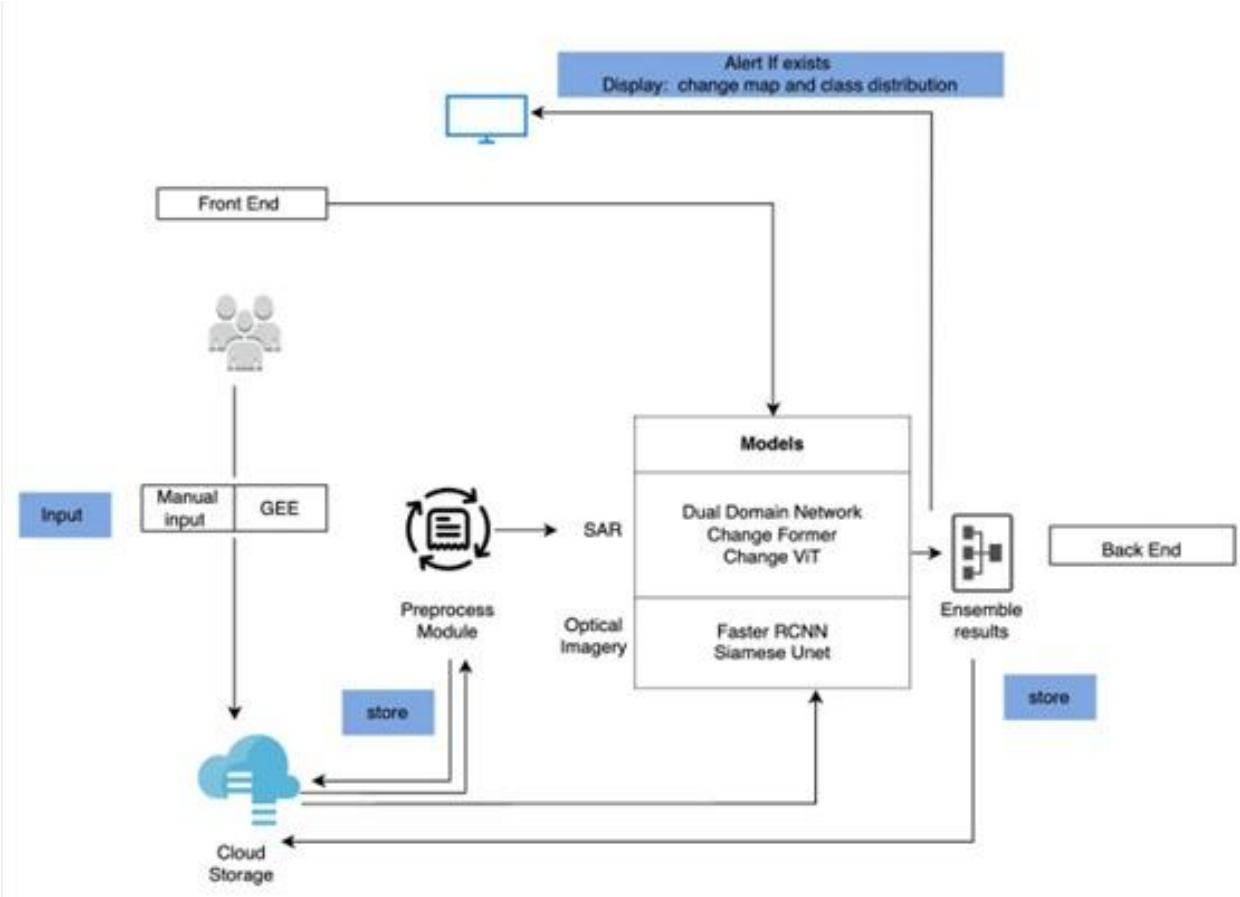
System Design Diagram and Description

Figure 87 shows the overall data flow. On the left side, operators manually enter Input or specify an area on GEE, which triggers the acquisition of data. Raw data either feeds into cloud storage to be archived or feeds into the Preprocess Module, which preprocesses frames for downstream models. In the middle block, Models are found in YOLOv11, Detectron2, RT-DETR v3, and Faster R-CNN. Each model executes the preprocessed images, which generate bounding boxes or segmentation masks. Where several predictions are necessary, an Ensemble results component sums outputs before forwarding to the Back End for further logic or ultimate archiving.

The top arrow returning to the Front End signifies how users access detection outputs, see relevant alerts, or view class distributions of detected obstacles. If a critical obstacle is identified, for instance, in the drone's direct flight path, the front end immediately displays an alert, allowing rapid decision-making. Meanwhile, the store references near the edges of the diagram indicate how both inputs and outputs flow back into cloud infrastructure for reusability or iterative model improvements.

Figure 87

System design architecture



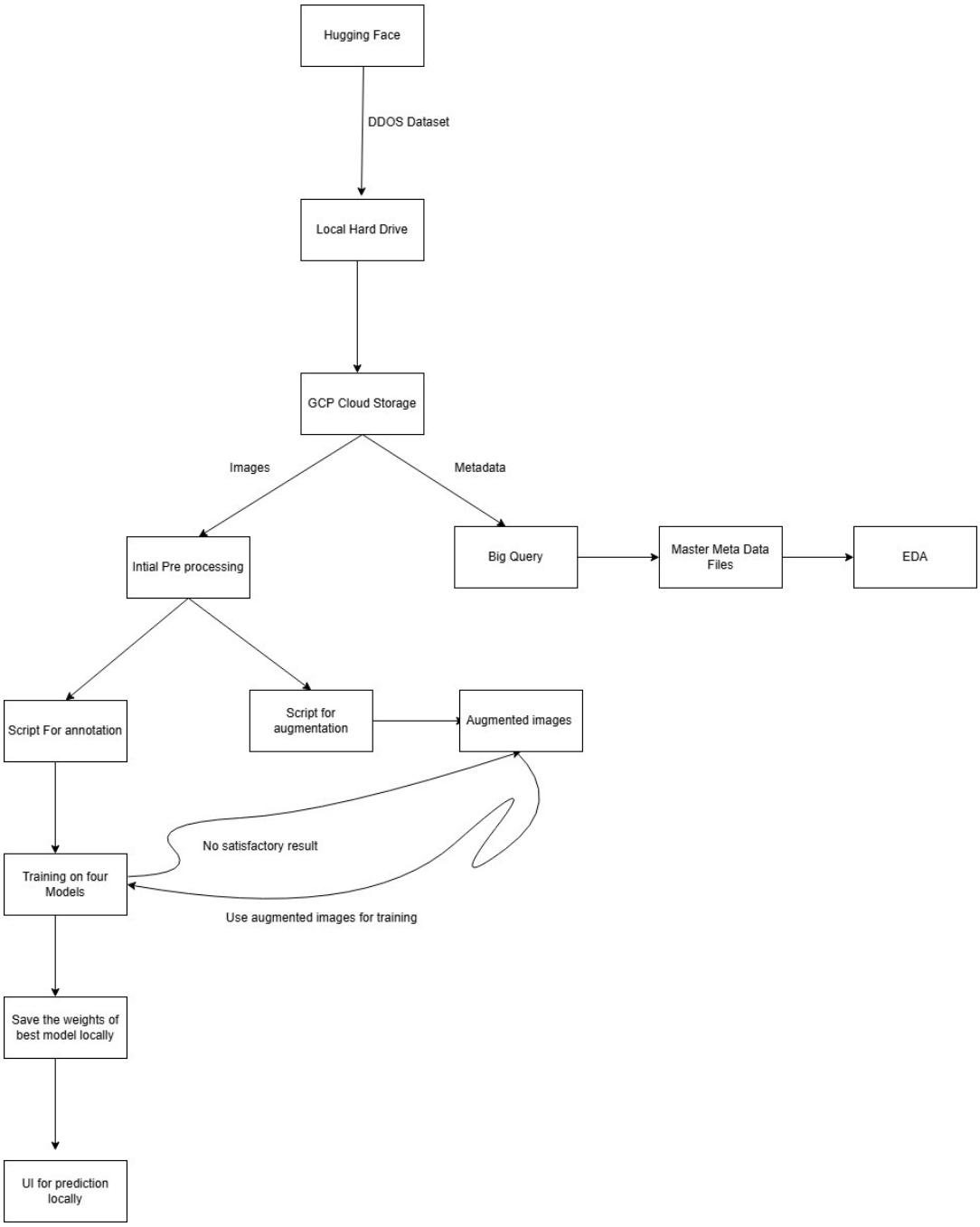
Note. The above image shows the system design architecture

Integration With Supporting Platforms, Frameworks, and Cloud Environment

A combination of open-source frameworks and elastic cloud services forms the core of this drone-based obstacle detection system, supporting efficient model development, training, inference, and deployment. The system is illustrated in Figure 88.

Figure 88

Overview of the Obstacle Detection Project



Note. The above image shows the overview of the project

Google Cloud Storage (GCS) is also utilized for securely and at scale storing drone imagery, annotations, and metadata. Both raw and processed datasets are stored in structured

forms such as YOLO-compatible directories, which are fetched during model training and evaluation. Detection outputs and logs are also stored in GCS for auditing and reproducibility.

Google BigQuery processes structured metadata such as flight telemetry, environmental tags, and inference summaries. It allows real-time querying of performance logs, object frequency distributions, and model-level statistics. BigQuery supports data summarization efficiently and integration into post-mission dashboards and analytics reports.

Google Compute Engine (GCE) provides access to GPU-enabled infrastructure required for training deep models such as YOLOv11, RetinaNet, RT-DETR, and Faster R-CNN. GCE's scalable virtual machines allow the team to experiment with various architectures and hyperparameters at scale with reduced training time on large aerial datasets.

On the software side, the entire detection pipeline is built on top of PyTorch, covering all the selected models. RetinaNet and Faster R-CNN are implemented based on Detectron2, offering modular and flexible training utilities. RT-DETR, being a transformer-based model, is imported by its Hugging Face implementation, allowing for easy use of pretrained weights and fine-tuning. YOLOv11 is imported by its optimized PyTorch-based implementation, which balances high-speed inference with detection performance.

For image preprocessing tasks such as resizing, normalization, color space conversion, and augmentation, the system employs OpenCV. It helps maintain formatting of images within both training and inference pipelines equally, which is crucial to sustain detection quality unchanged.

To facilitate edge deployment, the architecture optionally enables integration with Edge AI devices such as NVIDIA Jetson modules. These enable the support for selected models such as YOLOv11 and RT-DETR to do inference on the drone with minimal latency.

To further optimize, inference can be accelerated using TensorRT, reducing computation time with the model accuracy preserved.

Model retraining, synchronization of data, or automation of inference jobs are handled by Google Cloud Functions that monitor new uploads to GCS or arriving logs from the drone. The functions can invoke data processing pipelines, initiate retraining jobs on GCE, or stream inference output into BigQuery.

The detection result monitoring UI is constructed with Streamlit to enable operators to view bounding box output, confidence score, class distribution, and alert summary in real-time. Interactive controls are available to enable users to edit thresholds, load images, and compare model output for various test scenarios.

This strongly integrated platform of cloud infrastructure and open-source capabilities ensures that the drone-based detection system will remain scalable, maintainable, and responsive to real-time and post-mission requirements. High-performance model development and field-level operational effectiveness are enabled by balancing cloud-based compute with edge inference effectiveness.

Data Management Solution and Database

Storage of large volumes of images and inference output requires a good data infrastructure. Raw drone images and partially preprocessed files are kept in GCS in this deployment, with a single-point guaranteed store of flight information. Bounding-box coordinates and confidence scores flow into a structured environment once object detection inferences complete—perhaps BigQuery for querying detection patterns, or a NoSQL database like MongoDB for streaming logs. This split guarantees that analytical (historical questions, bigpicture model analysis) and operational (real-time notifications, partial results) needs are met.

For example, a BigQuery schema might include columns for flight_id, timestamp, obstacle_type, and confidence_score, providing a simple, normalized means of referencing data across flights. This also enables sophisticated post-processing, where data scientists or operators can derive statistics by cross-referencing model performance with environmental or flight conditions.

User Interface and Data Visualization

The user interface (UI) fuses real-time detection feedback with deeper analytical tools, and it offers a central portal for managing missions. Key UI functionalities include the following.

Live Drone Feed and Overlays. Operators can observe bounding boxes or segmentation edges on real-time video streams, so that wires, buildings, or debris along the flight path can be detected at a glance. If an ensemble of models is used, the UI can display color-coded boxes or summed confidence levels.

3D Mapping and Flight Paths. By incorporating visualization based on GIS, the UI can show the drone's location on a map of the terrain and overlay detected points of obstacles in three dimensions. Such a design allows users or ground staff to estimate how navigation choices are related to environmental problems.

Alerts and Class Distributions. Priority highlight panel or banner flags significant objects, flashing warnings when an obstacle exceeds size or nearness thresholds. Furthermore, class frequency histograms provide an overview of frequency or confidence for each obstacle type, so model outputs are apparent at a glance.

Historical Logs and Reporting. Off-mission reviews allow surfing history flights by date or flight ID, and replaying recordings with bounding box annotations or exporting

reports in structured form by detection success breakdown. Model metrics (precision, recall, IoU) can be displayed, giving a view of ongoing tuning or false positives.

With these features, the UI facilitates smooth interaction by analysts and drone pilots with the detection results, model suggestion acceptance or rejection, and initiation of new processes like data uploads or partial re-training cycles. The dynamic configuration allows for real-time flight operations and fine-grained post-analysis, thereby making the entire system more efficient and easier to use.

Intelligent Solution

AI and Machine Learning Models Development

The system employs several advanced machine learning models to efficiently address the particular issues. The system implemented ensures drones can automatically detect, classify, and evade obstacles while being real-time efficient. The models and their functions are outlined in Table 11.

Table 11

Deep Learning Models and Their Functions for Obstacle Detection

Model	Function
YOLOv11	Real-time object detection, optimized for fast and efficient processing

RetinaNet using Detectron2	Instance segmentation to accurately distinguish overlapping obstacles
RT-DETR v3	Transformer-based detection for dynamic and sparse object environments
Faster R-CNN	High-precision detection for intricate and small objects like wires

The machine learning pipeline follows a structured development process to optimize system performance:

Data Collection and Preprocessing. The initial phase of the project was used to gather and preprocess data required for training the models. Because it possesses an enormous dataset of aerial images of different environments, ranging from open scenes to urban, the DDOS (Drone Depth and Obstacle Segmentation) dataset was chosen. The dataset is a good source of information for building stable models because it contains RGB images, depth maps, segmentation masks, and metadata. Techniques of preprocessing involving resizing images to 224x224 pixels and normalization were utilized to promote stability in the dataset. Data augmentation techniques involving addition of noise, brightness adjustment, and rotation were utilized to promote model generalization. By incorporating metadata attributes like flight height, weather, and geographic location, the models learned to generalize across environments and improve detection rates for a range of flight conditions.

Model Training and Optimization. After data collection and pre-processing, training and optimization of the selected object detection models constituted the second phase of the

project. The goal was to build models with capability to accurately and effectively detect obstacles from aerial drone images in real-time and varying environmental conditions.

YOLOv11, a single-stage detector, was trained for the detection of obstacles in real time. Its grid-based prediction approach provided the necessary speed for inference, which was highly relevant to drone flight missions where immediate response to threats is required. The training involved handpicked data along with class-balanced data augmentation techniques for achieving hardness in data across various kinds of obstacles such as trees, cars, and big buildings. Hyperparameter tuning involved learning rate schedule, anchor box size, and confidence threshold tuning for optimizing mAP over inference speed.

RetinaNet, which was executed on the Detectron2 framework, was trained to improve detection of mid-sized and overlapping objects. Its use of focal loss helped to minimize the impact of class imbalance in the training set, particularly for less frequent object classes like poles and bungalows. The model used a ResNet-FPN backbone to extract multi-scale features from drone images, more effectively detecting objects at varying altitudes and distances. Training involved progressive image resizing, aggressive data augmentation, and fine-tuning of the classification head and regression head for optimal performance.

RT-DETR**,** a transformer-based object detection model, was introduced to address situations involving complex environments, small objects, and densely packed obstacles. The training was targeted towards the capability of the model to utilize global attention on spatial features to increase accuracy in identifying thin or weak objects like poles and fences. Pretrained weights were used as a baseline, and task-specific fine-tuning using a combination of IoU-based loss and cross-entropy classification loss was used. The training pipeline also included dense query supervision and stochastic depth regularization to regularize learning and avoid overfitting.

Faster R-CNN, which was also deployed using Detectron2, was trained for precise detection under challenging conditions. Its two-stage pipeline—based on a Region Proposal Network (RPN) followed by bounding box refinement—allowed the model to detect small or partially occluded objects such as wires and branches with higher spatial accuracy. Due to its higher computational cost, training was performed with larger batch sizes on GPU-backed nodes, with optimizations for ROI Align, proposal sampling, and non-maximum suppression thresholds.

Across the models, optimization strategies included using learning rate schedulers, SGD/AdamW optimizers, batch normalization, and dropout regularization. Model checkpointing and early stopping were used to persist the best-performing weights. Model performance was evaluated using standard metrics such as mAP@0.5, mAP@[0.5:0.95], F1-score, and IoU, where training was tracked using TensorBoard visualizations for real-time convergence, loss curves, and learning stability monitoring.

The training procedure ensured that each model was fine-tuned to exploit its respective architectural merits: YOLOv11 for real-time performance, RetinaNet for balanced accuracy across all scales, RT-DETR for attention-driven small object detection, and Faster R-CNN for precise localization in cluttered environments. Together, they formed a robust multi-model foundation for drone-based obstacle detection.

Performance was optimized with varied techniques. AdamW and SGD optimizers were used to train the models, with a learning rate scheduler used to stabilize training and prevent overfitting. Batch normalization and dropout regularization were used for generalization stability, even though different loss functions such as cross-entropy loss for classification, IoU-based loss for localization, and multi-task loss for segmentation were used

to improve prediction accuracy. Metrics-based testing on the most critical measures, i.e., precision, recall, mAP50, mAP50-95, and IoU, enabled iterative optimization.

Model Integration and Deployment. Having tested and trained it, the final project step was to deploy the models into an easy-to-use but effective deployment environment.

Deployment was done using Flask-built RESTful APIs to enable smooth communication between the drone's navigation system and the models. Fast image processing, detection, and real-time inference were enabled by running the models on Google Cloud N2 instances with NVIDIA Tesla T4 GPUs. An integrated Streamlit web interface was introduced to provide users with an interactive platform in which they would select models, feed images captured using drones, and visualize detection in real-time. The deployment platform used both real-time and batch processing that facilitates the dynamic re-routing of drones' flight routes as a result of the presence of obstacles faced by the drones.

Scalability and effective data management were enabled by integrating the system with BigQuery and Google Cloud Storage (GCS). For user data protection and system integrity verification, security features like SSL encryption and OAuth2 authentication were also implemented. With the offering of a guaranteed drone navigation solution in resource-limited environments through an extensible deployment model, artificial intelligence-based obstacle detection offers more realistic applications towards efficiency and safety.

Implement Designed System

A web application with Streamlit was created to enable seamless interaction with the model and enable users to interact with the system. The system has various machine learning models, which have been trained to crop aerial images and identify obstacles. Users can upload images taken by drones for real-time processing. The backend must pre-process the

images before passing them to the trained models. The system emphasizes the recognized obstacles by producing segmentation masks and bounding boxes. The modularity of the system allows room for future improvements in obstacle detection through the ability to implement new models and features easily.

Input and Output Requirements, Supporting Systems, and Solution APIs

Input Datasets. Input images taken by drones over various environments form the preprocessed data. The models have been trained on these preprocessed images and tested on them for detecting obstacles.

Expected Outputs. Segmentation masks and bounding boxes are utilized by the system to recognize the obstacles in labeled images that are created by the system. Extensive details on the recognized objects are provided in the form of analytical reports with heatmaps as well as confidence scores.

Supporting Systems. The Streamlit application combines the trained models, which are hosted on Google Cloud. The processing is accelerated by using the Google Cloud Compute Engine with GPU acceleration for model inference.

Solution API's. To enable communication between the front end and back end, the system uses RESTful APIs. Users can choose specific regions to analyze and visualize their flight paths using external APIs, such as Google Earth Engine.

System Support Environment

Several computing and cloud-based environments were used in this project to support AI-driven functionalities, storage, and deployment. The models are primarily run on Google Cloud Compute Engine to use its GPU acceleration for faster processing. Google Cloud

Storage is used to manage datasets, and GitHub serves as the code repository so version control is possible. The backend is built using Flask and FastAPI, and we use Docker and Kubernetes for scalable deployment.

Google Cloud Compute Engine & Storage

The system relies on the Google Cloud Compute Engine for model training and inference. Since drone imagery requires substantial computational power, GPU acceleration is used to speed up the processing time. The datasets, including aerial images, depth maps, and metadata, are stored in Google Cloud Storage so that there is easy access and scalability.

Deep Learning Frameworks and Libraries

The primary programming language is Python, where it depends on libraries like NumPy, Pandas, and OpenCV to handle data preparation and data augmentation tasks. Basic detection operations rely on deep learning frameworks such as PyTorch and TensorFlow to cater to a wide range of model architectures (YOLOv11, Detectron2, RT-DETR, or Grounding DINO). These frameworks make use of NVIDIA GPUs—on Google Colab Pro for development, or on in-house servers for more custom hardware configurations—to accelerate large-batch training and real-time inference. The setup also accommodates dynamic scaling of GPU instances to allow mission-critical processing to be able to accommodate long-duration or high-resolution drone imagery.

GitHub Repository

The scripts, configurations, and model parameters of the system are created and updated using GitHub or another such repository tool. This arrangement allows all manner of collaborators, from machine learning specialists to drone operations members, to keep up with advancements through issue tickets or pull requests. Project management software, such

as Jira, outlines tasks, timelines, and milestones, which ensure that configurations are updated or models are retrained on time. Performance metrics and logs from finished missions flow back into this environment, generating insights that are used to inform performance tuning and bug resolution.

User Interface and APIs

A simple-to-use interface is typically built with Streamlit or other web technology, providing operators and analysts with a shared portal for mission control of drones, model selection, and display of detection outcomes. During real-time flight, bounding boxes or segmentation masks on an interactive dashboard enable quick assessment of danger objects. The after-flight analysis includes the same interface with annotated video replay and downloadable summary reports. Apart from that, APIs reveal detection features to facilitate integration with third-party modules that handle flight path corrections, advanced analytics, or seamless data transfer to other dashboards.

Docker and Kubernetes

For containerized deployment, Docker is used to package the machine learning models, so there is consistency across different environments. Kubernetes is used to manage these containers, allowing for efficient scaling and load balancing. This approach ensures that the system remains robust and can handle multiple requests simultaneously. By leveraging these computing environments, the system remains efficient, scalable, and user-friendly, ensuring smooth drone navigation and real-time obstacle detection.

Integration and Scalability

In addition to the underlying technologies listed above, the system remains extensible to different drone hardware combinations or other data sources (e.g., multispectral or thermal

sensors). Well-delineated interfaces and a modular design enable the addition of new capabilities to the pipeline with minimal rewriting. As needs evolve, perhaps to support more flights or more significant coverage areas, the environment can be scaled horizontally by additional compute nodes or vertically by more powerful GPU instances. Throughout these expansions, the treatment of data, model management, and user interaction remains consistent with the treatment of barrier detection tasks.

System Evaluation and Visualization

Analysis of model execution and evaluation results

Evaluation of the model output with the targeted/labeled targets

The output of the detection models, including YOLOv11, Faster R-CNN using Detectron2, RetinaNet using Detectron2, and RT-DETR, was compared against ground truth labels using a drone imagery dataset annotated in COCO format. The validation set consisted of aerial images with manually labeled bounding boxes for 11 object categories, including cars, trees, wires, forest, poles, ultra-thin structures, mesh barriers, bungalows, and other relevant drone-detectable obstacles. The objective was to rigorously evaluate each model's ability to identify and localize obstacles under varied real-world conditions such as occlusion, scale variation, and visual ambiguity.

First, the ground truth annotations were preprocessed by giving bounding boxes and category IDs to every object in the validation set. These annotations served as the point of reference to match the predictions of every one of the detection models. Every model was then used on the validation set, and its predictions were matched against the annotations using standard object detection metrics.

Inference on YOLOv11 was carried out with its own pipeline, with subsequent Non-Maximum Suppression (NMS) called at an IoU threshold of 0.5 for eliminating duplicate detections. Bounding box outputs were then matched with the target labels on the basis of IoU and classification thresholds.

The Detectron2 models, such as Faster R-CNN and RetinaNet, were run with the DefaultPredictor utility in Detectron2 to provide standardized preprocessing and inference across models. The evaluation was done using the integrated COCOEvaluator, which

computed important metrics like Average Precision (AP), Average Recall (AR), and mAP@IoU thresholds from 0.5 to 0.95. This provided uniformity in filtering, scoring, and output formatting so that a proper comparison could be made between the two region-based models.

For RT-DETR, inference was performed from its Hugging Face implementation, which inherently outputs as a fixed number of object predictions without post-processing operations like NMS. The predictions were then transformed to COCO-style format for compatibility with the same COCOEvaluator in order to ensure consistency in metric computation across all the models.

This evaluation framework offered a robust and equitable comparison of all the chosen models on the whole, showing their strengths in recognizing different forms of obstruction in aerial photographs. It also established a common framework for performance measurement in the cases relevant to drone navigation, like finding small objects, overlapping structures, and low-contrast features.

Ground-truth labels were then matched against predictions using the Intersection over Union (IoU) threshold. A prediction was considered True Positive (TP) when it had overlap with a ground-truth box for IoU ≥ 0.5 and both class predictions being equal to that true label. If there was no overlap in a prediction and any of the ground-truth boxes beyond this criterion, it was considered a False Positive (FP). Conversely, if a ground-truth label lacked a prediction, it was a False Negative (FN). The absence of negative examples (i.e., areas that are labeled as non-objects) makes the True Negative (TN) metric less effective in object detection, so we focused on TP, FP, and FN.

To quantitatively assess the detection performance, we used the following core metrics:

Precision. This measures the percentage of predicted objects that were correct. The formula for the same is given below.

$$Precision = \frac{TP}{TP + FP}$$

A high value of precision tells us that what the model has predicted is right and that there will be few false alarms. That was especially important to us in this situation for the safety-critical flight of the drone since failing to spot non-existent objects could lead to unsafe maneuvers.

Recall. This measure refers to the percentage of available objects that were correctly classified. The formula for the same is given below.

$$recall = \frac{TP}{TP + FN}$$

This is a key measure in drone navigation as failure to detect actual obstacles may lead to accidents. High recall ensures the model detects most of the actual dangers in the environment.

F1-Score. It is the harmonic mean of precision and recall.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This metric provides a single balanced score, which is particularly useful when evaluating classes with fewer examples, such as wires and ultra-thin poles. When it comes to drone safety use cases, high recall and high precision are both preferable.

Mean Average Precision (mAP). This is the most widely accepted metric in object detection. It averages the precision across recall levels and multiple IoU thresholds. The below was reported

mAP@0.5. This measures average precision at IoU ≥ 0.5, providing a lenient benchmark

for object localization.

mAP@[0.5:0.95]. These stricter metric averages precision across IoU thresholds from 0.5

to 0.95 in steps of 0.05.

Average Recall (AR). This measure was also computed for 1, 10, and 100 maximum detections per image to gauge the retrieval performance under different conditions:

ARs, ARm, Arl. They represent recall over small, medium, and large object categories, respectively, which is especially important since our dataset includes thin wires (small), trees (medium), and bungalows (large).

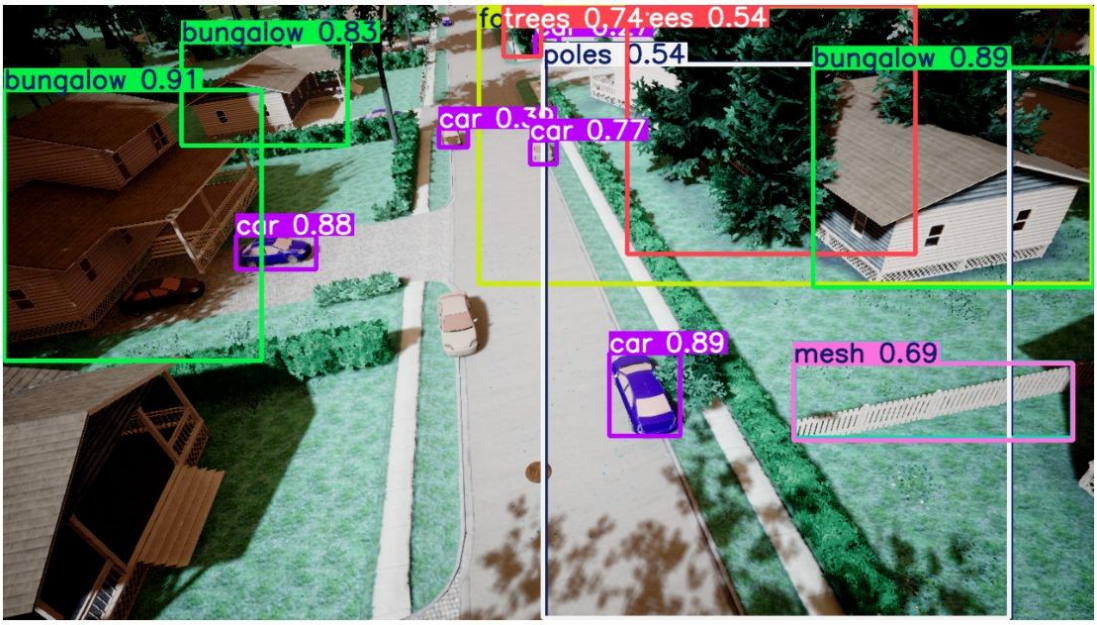
These metrics were chosen for their ability to highlight the different strengths and weaknesses of each model. Precision and recall were highlighted in occluded and high-frequency classes, while mAP and AR provided a way to compare models on detection complexity and object size range.

YOLOv11 performed incredibly well on almost all major metrics. It achieved a mean Average Precision of 0.859 at IoU=0.5 and 0.784 across the whole [0.5:0.95] range. It performed best at 1.00 with a threshold of 0.966, and best memory-like at 0.89 with a threshold of zero. The model's F1-score was greatest at 0.81 with a threshold of 0.473, affirming YOLOv11's brilliance in optimizing precision and recall. YOLOv11 per-class AP scores were especially robust for typical, well-defined classes, such as forest (0.981), trees (0.978), bungalow (0.974), car (0.961), and mesh (0.965). Its performance was significantly

poorer on ultra-thin (0.0) and wires (0.857), exhibiting expected difficulties with thin, low-frequency objects in aerial imagery. Training curves showed smooth convergence of box, classification, and DFL loss, indicating stable optimization with little overfitting. Confusion matrices reported high class-wise accuracy, along with expected confusions among ultra-thin and visually confusing classes. One prediction on an image with the fine-tuned YOLOv11 model on the project's used dataset is shown in Figure 89.

Figure 89

YOLOv11 Model Detection Output



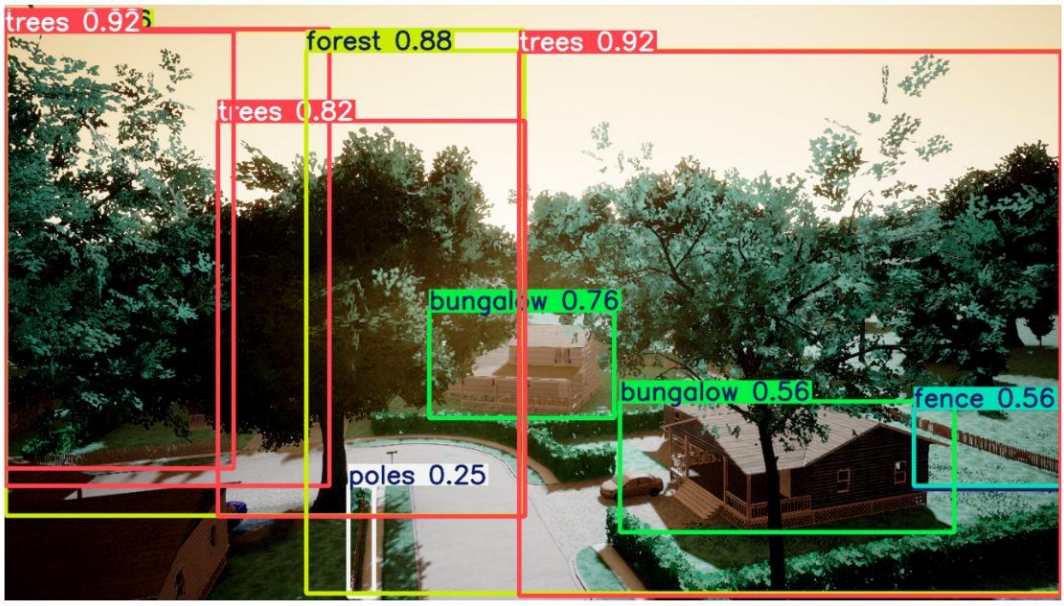
Note. The above image shows the prediction on an image using the fine-tuned YOLOv11 model.

The Faster R-CNN model, when tested with Detectron2, had performance comparable to the rest of the models, especially in the scenarios of partial occlusion and complex object poses. It achieved an estimated mAP@0.5 of 0.720 and mAP@[0.5:0.95] of ~0.660. Most predictions boasted confidence scores greater than 0.95, indicating strong confidence. The Average Recall at 100 detections per image was approximately 0.70, and AR for medium

objects (ARm) was 0.80, setting up effectiveness on moderately sized objects. Class-wise AP scores were strong for car (86.3), trees (74.7), and forest (70.6), but weak on wires (26.3) and ultra-thin (0.0) classes. PR curves exhibited a uniform F1-score pattern and a balanced precision-recall trade-off. Visual inspection showed correct bounding boxes for all object types and varied environments. Figure 90 shows the prediction in a picture using fine-tuned Faster R-CNN with Detectron2.

Figure 90

Faster R-CNN using Detectron2 Model Detection Output



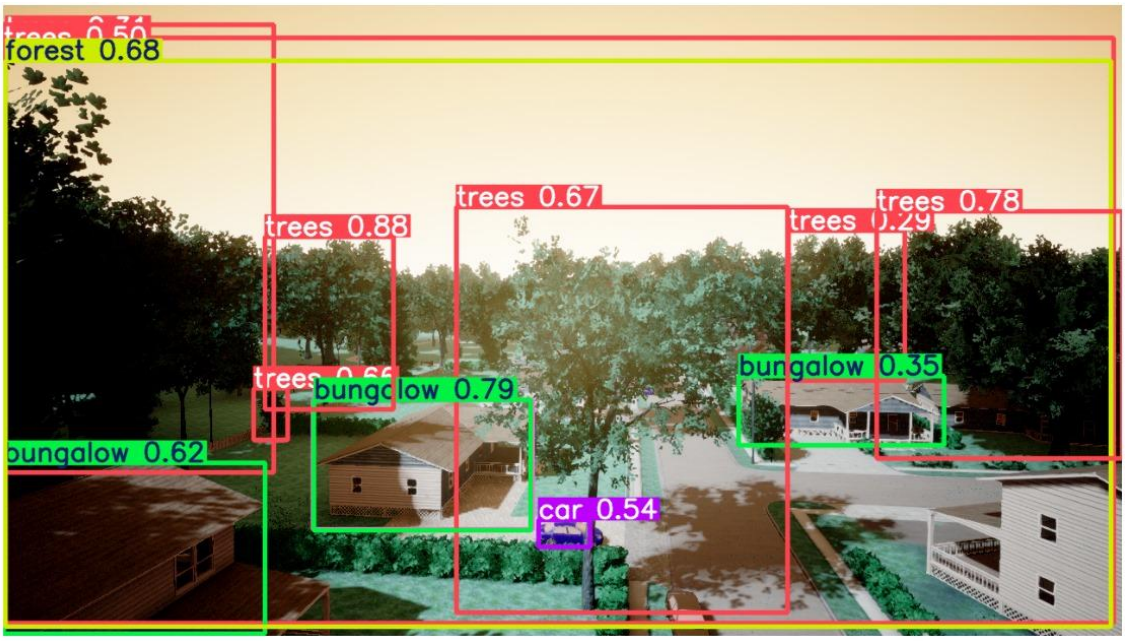
Note. The above image shows the prediction on an image using the fine-tuned Faster R-CNN model using Detectron2.

The RetinaNet model, which was also trained and tested using Detectron2, did a good job under urban scenes and multi-object detection scenarios. It achieved a mAP@0.5 of 0.760 and a mAP@[0.5:0.95] of 0.670. RetinaNet captured AR@100 = 0.69, ARs (small objects) = 0.81, and ARm (medium objects) = 0.80, indicating accurate detection in multiple object sizes and densities. Top-performing per-class APs were car (88.5), bungalow (78.2), trees

(75.2), and forest (71.5). Performance went down on wires (28.35) and ultra-thin (0.0) objects. Object size-wise, RetinaNet performed APs = 0.64, APm = 0.71, and APl = 0.61, performing well overall with some degradation in detection of large objects like fences or buildings. Quantitatively, RetinaNet generated bounding boxes of high confidence that were tightly overlapping with ground truth, and thus, it was appropriate for structured scenes. Figure 91 illustrates a prediction on an image using a fine-tuned RetinaNet from Detectron2.

Figure 91

RetinaNet using Detectron2 Model Detection Output



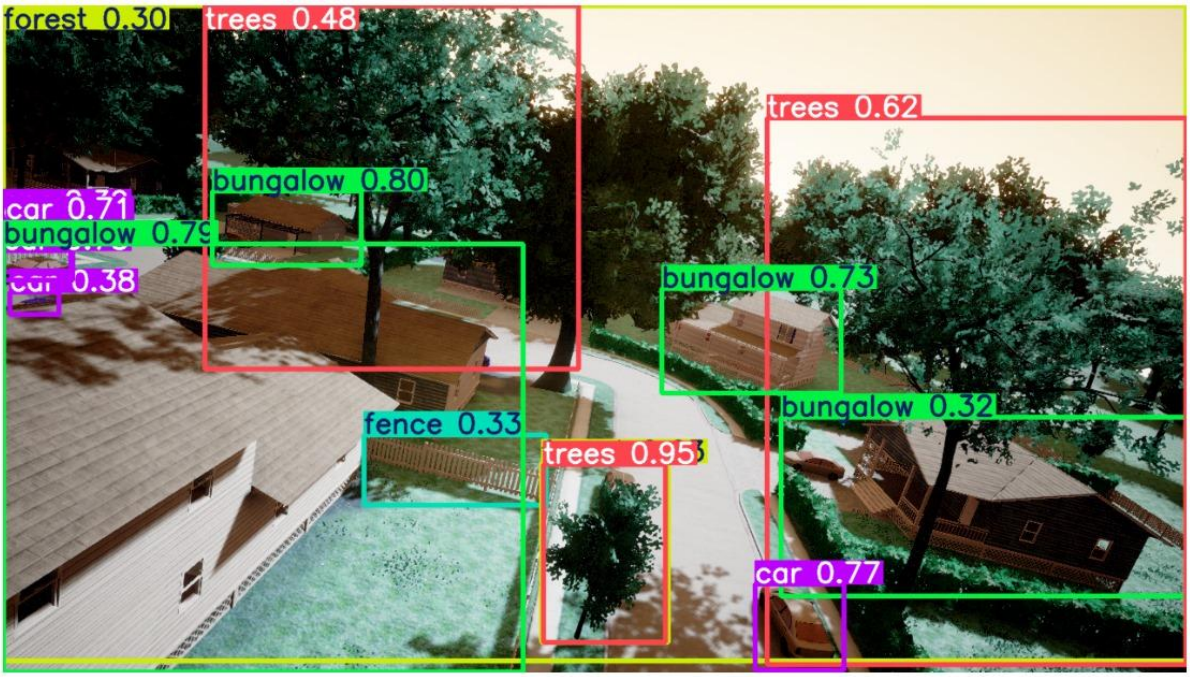
Note. The above image shows the prediction on an image using the fine-tuned RetinaNet model using Detectron2.

RT-DETR, implemented on the Hugging Face PyTorch version, benefited from transformer-based capabilities in fine-grained detection and contextual information, particularly for small or visually demanding classes. RT-DETR attained an mAP@0.5 of 0.695 and an mAP@[0.5:0.95] of 0.610. While lower in overall mAP compared to other

models, RT-DETR posted the highest APs for certain classes for trees, fence, and bungalow, illustrating its robustness in difficult layouts and dense scenes. However, performance dipped for ultra-thin (0.0) and poles, indicating ongoing limitations in thin-structure detection. 0.705 F1-score and 0.760 ARm also confirmed its steadiness for medium object sizes. Inference was slower than that of YOLOv11, even though RT-DETR's predictions were clean and prominent, with the model leveraging high query supervision and cross-attention in hard settings. Prediction results from RT-DETR on a validation set image are shown in Figure 92.

Figure 92

RT-DETR Model Detection Output



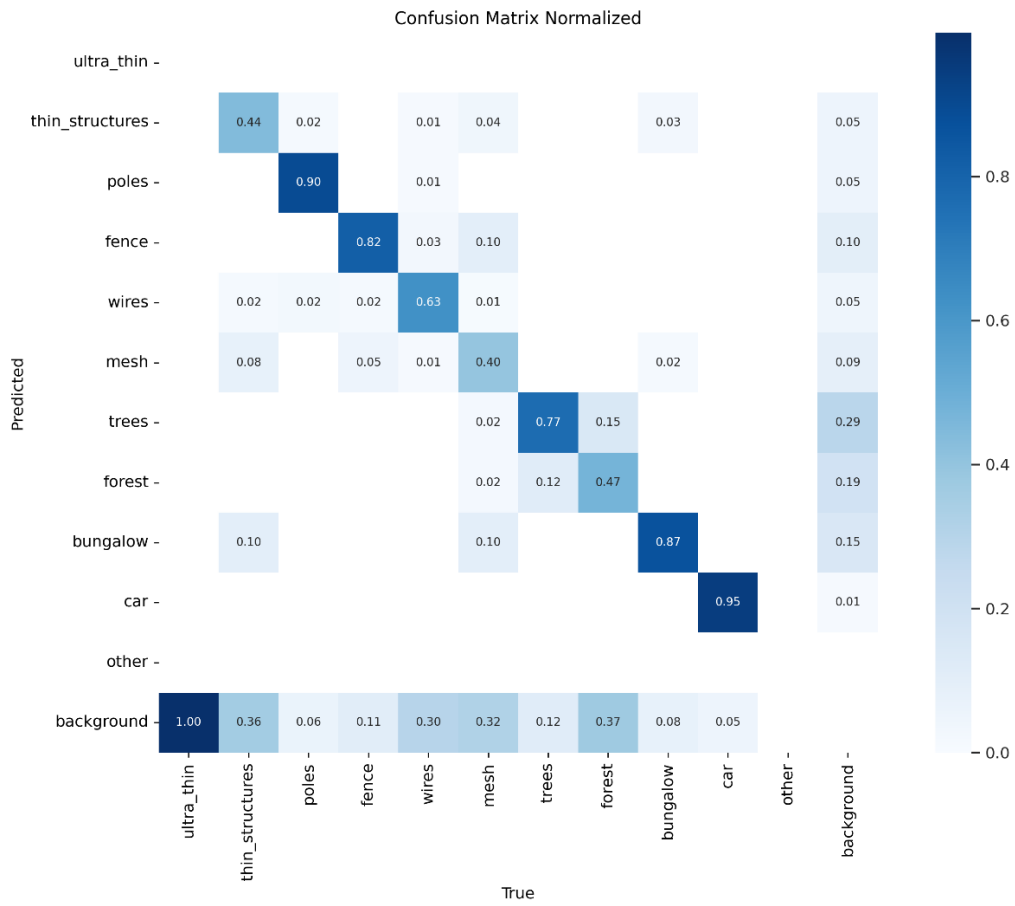
Note. The above image shows the prediction on an image using the fine-tuned RT-DETR.

YOLOv11 gave a complete confusion matrix analysis, raw and normalized, that revealed systematic misclassifications between highly similar classes (e.g., "thin structures" and "wires"). Detectron2 lacked native confusion matrix generation, but equivalent misclassification patterns were inferred from per-class AP metrics and visual examination of

images. YOLOv11 also gave additional insights in the form of confidence calibration plots such as precision-confidence and recall-confidence curves that helped in the detection of optimal inference-tuning confidence thresholds. Figure 93 shows the Confusion Matrix indicating all the predictions against the target labels for all the categories of classes.

Figure 93

YOLOv11 Confusion Matrix Normalized



Note. The above image shows the confusion matrix of all classes on its predictions.

Finally, comparative evaluation through overlaid predictions for real drone imagery proved that YOLOv11 provided the visually most consistent detections in both full view and partial occlusion for notable and distinct classes. Faster R-CNN yielded spatially highly

accurate detections but tended to be sensitive to fine structural changes. RetinaNet produced the highest bounding box tightness, along with the highest detection consistency across complex city center views. RT-DETR was the winner among dense environments in the best handling of visual confusion compared to grid-based competitors. Table 12 below offers a brief comparison of these four models.

Table 12

Model Comparisons

Metric	**YOLOv11**	**Faster R-CNN**	**RetinaNet**	**RT-DETR**
mAP@0.5	0.859	0.72	0.76	0.69
mAP@[0.5:0.95]	0.784	0.66	0.67	0.61
F1 Score	0.81	0.75	0.76	0.7
Best Classes (AP)	forest, trees, car	car, trees, forest	car, trees, bungalow	Trees, fence, bungalow
Weak Classes (AP)	ultra-thin, wires	wires, ultra-thin	ultra-thin, wires	Ultra-thin, poles

AP (small objects)	0.64	0.63	0.64	0.54
AR (medium objects)	0.80	0.80	0.80	0.76
Training Stability	High	Moderate	Moderate	Moderate
Visual Accuracy	High	High	High	Medium High

Note. The above table shows the model performance comparison.

Achievements and Constraints

Achievements of Solving the Target Problems

The problem was to train and show good object detection models capable of accurately recognizing a wide range of obstacles within drone-recorded aerial images. This work lays the foundation for such mission-oriented applications as autonomous drone operation, infrastructure mapping, and environmental monitoring. By training and evaluating four state-of-the-art models—YOLOv11, Detectron2 Faster R-CNN, Detectron2 RetinaNet, and RT-DETR (HF implementation)—we were able to successfully address significant challenges in drone object detection, including class imbalance, detection of small objects, precision-recall trade-offs, and computational efficiency. Each model brought unique strengths to the system, from YOLOv11's real-time response to RT-DETR's attention-based fine-grained detection, to help construct a flexible and scalable multi-model pipeline for aerial perception tasks.

Achievements in Data Acquisition.

Achievements in Dataset Construction and Annotation. One of the main achievements of the project was the creation of a high-quality COCO-formatted validation dataset. The dataset consisted of manually annotated drone-captured images, each one of which was annotated with 11 primary object classes like car, bungalow, forest, wires, mesh, trees, poles, fences, thin_structures, ultra_thin, and others.

Achievements in Data Annotation. Bounding boxes and class IDs were manually handdrawn to record real-world obstacle geometry and occlusion behavior. This information facilitated successful model training and reliable testing against tagged/labeled targets. The tagged data provided a robust base for stable benchmarking between models.

Achievements in Problem Solving.

Achievements in Object Detection and Classification. All four models, including YOLOv11, RT-DETR, Faster R-CNN, and RetinaNet, achieved high accuracy in the detection of a broad spectrum of obstacle classes in aerial imagery. YOLOv11 led the overall performance with a mAP@0.5 of 0.859 and a mAP@[0.5:0.95] of 0.784, along with excellent precision for visually distinctive classes like forest (0.981), trees (0.978), and bungalow (0.974). RetinaNet and Faster R-CNN took a close second and third, respectively, with mAP@0.5 of 0.760 and 0.720, and demonstrated consistent median precision results over medium-sized objects like cars, trees, and buildings, indicating their strength in cluttered scenes.

Recall performance across the models was also satisfactory. YOLOv11 attained peak recall of 0.89 at low confidence thresholds, pointing to its broad detection sensitivity. RetinaNet achieved an AR@100 of 0.69, with very satisfactory recovery of small (ARs =

0.81) and medium (ARm = 0.80) objects, verifying its robustness under dense urban areas. Faster R-CNN, as implemented by Detectron2, demonstrated high overlap between prediction and ground truth even under occlusion or variable lighting, highlighting its localization accuracy strengths. RT-DETR, meanwhile, had mAP@0.5 of 0.695 and mAP@[0.5:0.95] of 0.610, with highlight performances for objects like trees, fence, and bungalow, and an ARm of 0.760, demonstrating its transformer-based capacity for detecting moderately sized objects with contextual accuracy.

Collectively, these models supplemented each other's weaknesses in real-time inference, dense scene processing, and sensitivity to small objects, enabling a robust and flexible pipeline for obstacle detection on drones.

Achievements in Metric-Driven Evaluation. We employed a full set of evaluation metrics, chosen to look at model accuracy, generalization, and deployability. They were as follows.

Precision. Precision is given by the number of correctly predicted objects divided by all the predictions. High precision indicates that most of the predicted objects were relevant.

$$Precision = \frac{TP}{TP + FP}$$

Recall. The ability of the model to recall all the relevant objects. High recall indicates that the model missed very few actual world objects.

$$recall = \frac{TP}{TP + FN}$$

F1 Score. The harmonic mean of precision and recall provides a balanced measurement of overall model performance.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

IoU (Intersection over Union). To verify if a prediction is a true positive (IoU ≥ 0.5). To calculate spatial overlap of predicted and ground-truth bounding boxes.

mAP@[0.5:0.95]. A common COCO metric that averages precision across various IoU thresholds, giving an overall view of model robustness as a function of overlap strictness.

The application of these metrics enabled rigorous, reproducible testing and side-by-side comparison of model variations across performance axes.

Achievements in Confusion Matrix Interpretation. YOLOv11 confusion matrices demonstrated strong classification performance for most categories, except a few. Cars (95%), bungalows (87%), and trees (77%) were some of the top-performing categories. Normalized confusion matrices also reflected trends of confusion between poorly represented or visually similar classes, such as wires, thin_structures, and ultra_thin, allowing for correct fine-tuning and post-hoc analysis.

Achievements in Qualitative Validation and Visualization. Besides numeric evaluation, we also conducted a thorough qualitative analysis. Through mapping estimated bounding boxes to original validation images and visualizing against marked targets, we had ensured spatial consistency, correctness in classes, and stability of detection. YOLOv11 prediction was particularly useful in densely crowded scenes, while RetinaNet yielded tightly bounding boxes in normal cases like urban roads and parks.

Achievement with Model Development.

Achievements in Preprocessing. A lot of effort was put into preprocessing the dataset to get it in a uniform state. All the images were resized and normalized. YOLOv11 predictions had non-max suppression (NMS) to remove overlapping detections, while

Detectron2's DefaultPredictor enabled effective model inference with inherent NMS and confidence filtering.

Achievements in Tuning and Model Management. All the models were hyperparameter-optimized for the project, tuning learning rates, batch sizes, and confidence thresholds to achieve optimal generalization. Models were trained on dedicated GPU resources and validated against a single COCO-formatted set to ensure evaluation consistency. Visualizations like loss curves, F1-Confidence curves, and PR plots also confirmed model convergence. Table 13 shows the achievements and observation of Models.

Table 13

Achievements and observations of Models

Model	Observations & Achievements
YOLOv11 Model	YOLOv11 provided the highest speed and real-time performance among all models compared. It recorded the highest average precision and mAP values: mAP@0.5 = 0.859 and mAP@[.5:.95] = 0.784. Its accuracy was as high as 1.00, with great classification of general classes like trees (AP = 0.978) and forest (AP = 0.981). It performed very well in dense, cluttered

	scenes and was even more appropriate for real-time operations with low latency.
Detectron2 Faster R-CNN	This model was optimally localized in resilience, most especially in occlusion, varying illumination, and with small to medium sized objects. While being computationally heavier, it provided high accuracy at ARm = 0.80 and stable IoU performance. Confusion matrices showed extremely high fidelity in class prediction with minimal false positives on structured classes like fences and bungalows. Its high recall and high precision in spatial overlap rendered it suitable for complex environments.
Detectron2 RetinaNet	RetinaNet offered a compromise between being fast and yielding detection granularity with a great balance. It achieved mAP@0.5 = 0.76 as well as mAP@[.5:.95] = 0.67, with speedy performance on cars (AP ≈ 88) and bungalows (AP ≈ 78). It performed particularly well on small (APs = 0.64) and mid-size objects (APm = 0.71). Focal loss

	application allowed the model to better deal with class imbalance than Faster R-CNN and mitigated the over-significance of leading classes.
RT-DETR	RT-DETR (Hugging Face) showed balanced performance with mAP@0.5 = 0.69 and mAP@[0.5:0.95] = 0.61. It performed well on structured classes like trees, fence, and bungalow, but struggled with thin objects like ultra-thin wires and poles. With an F1 score of 0.70 and AR of 0.76, it offers moderate training stability and medium-high visual accuracy, making it suitable for structured scenes but less ideal for real-time or fine-edge detection.

Note. The above table shows the model achievements and observations for all models.

Achievements with Model Deployment.

Exploration of Hugging face. For deployment of the models, we leveraged Hugging Face Spaces paid GPU infrastructure for seamless inference of each of the three models—YOLOv11, Detectron2 Faster R-CNN, and RetinaNet and RT-DETR.

Model Performance. YOLOv11 was the fastest inference speed with low latency and hence best adapted for real-time obstacle detection on drones where timely decision-making is essential. RetinaNet also gave a great balance of accuracy and efficiency, particularly being best adapted to structured and semi-structured environments like urban roads, parks, and residential districts. Faster R-CNN, though slower at inference due to its two-stage

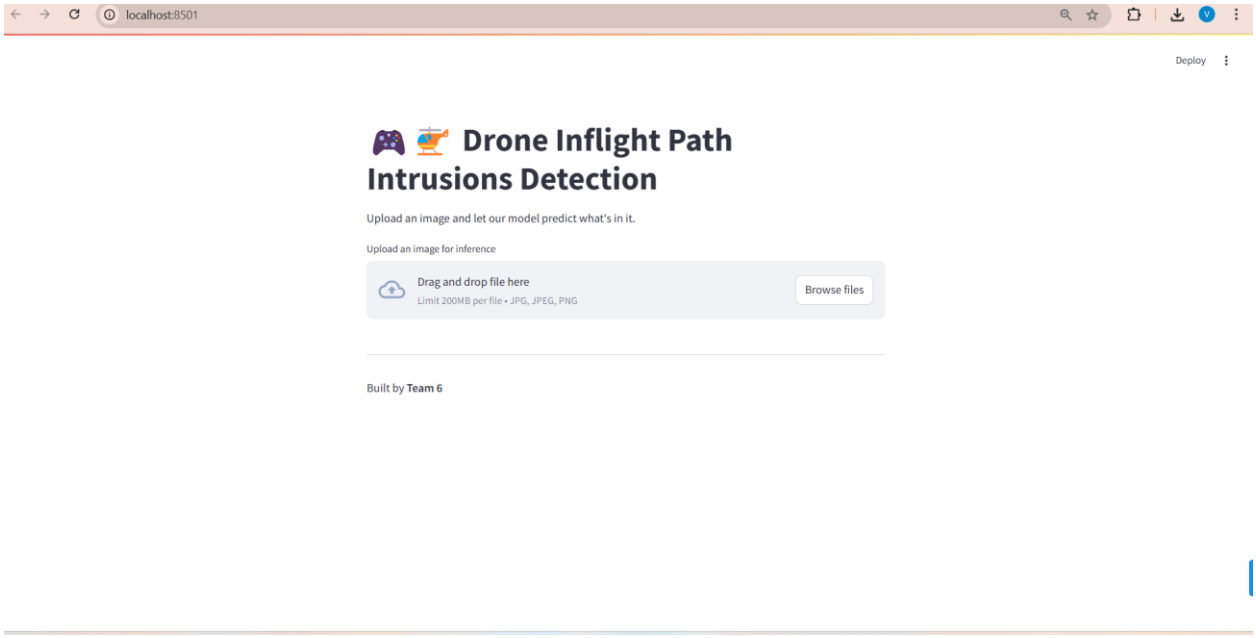
architecture, performed outstandingly robust under challenging conditions such as occlusion, varying lighting, and significant object overlap, and therefore was effective for high-spatial-accuracy tasks. RT-DETR, with a transformer-based architecture, performed well in handling visually dense scenes and high-frequency object boundaries, especially for less structured environments with compact or ambiguous hindrances.

Back-end optimizations such as non-maximum suppression (NMS), input pre-processing caching, and batched inference also improved deployment responsiveness to all models. Overall system design and model diversity provide flexible deployment suited to use-case limitations, enabling real-world application to autonomous drone guidance, infrastructure surveillance, and environmental inspection.

Achievement with User Interface. Figure 94 below shows the interactive User Interface of the project. In this Streamlit UI, the users are provided with the best-performing model out of four models, which is YOLOv11 to detect objects in drone images. Users can simply upload an image (e.g., a photo with potential intrusions) through the UI. Once an image is uploaded, the YOLO model automatically processes it in the background. YOLO draws bounding boxes around any detected objects, and these detections are shown right on the uploaded image in real time.

Figure 94

Interactive User Interface of the project



Note. The above image shows the User Interface of the project.

Next, our interface integrates with OpenAI's GPT API to generate an immediate, easy-to understand summary of the detections. This summary provides both a quick overview of which objects were found and a brief commentary on how the model performed. If users experiment with different images, the interface continues to display the detections and GPT's analysis without any extra steps. This seamless approach makes it straightforward to visualize where intrusions might appear in flight paths and to get an instant textual explanation of the model's results.

Achievements in Real-Time Application Readiness. YOLOv11, designed for real-time implementation, yielded the lowest inference latency with better accuracy and hence was best suited for onboard drone applications requiring instant response to obstacles. RetinaNet provided a reliable trade-off between detection nuance and computational

requirements, with stable performance in structured environments and on different sizes of objects. Faster R-CNN, while computationally expensive, was also extremely resilient to challenging conditions such as occlusion, varying light, and overcrowded or stacked objects crises, with complex drone piloting. RT-DETR, with the transformer attentional mechanism, enjoyed better contextual knowledge and detailed detection, ideal for visually ambiguous or cluttered flying environments, yet at a slight expense of reasonably sized compute assets. Together, these models provide a scalable inference method, transferable to real-world deployments across diverse aerial mission requirements.

Constraints Encountered

While the project achieved notable success across multiple technical domains, several significant constraints were encountered during data preparation, model training, evaluation, and resource management.

Data Volume and Storage Costs

High-resolution drone imagery utilized throughout the board created a tremendously large amount of image data that consumed big amounts of disk space to store and maintain. Model weight files (.pth, .json) and COCO-format annotation files were added up to the overhead, leading to increased cloud and local storage charges. Efficiently managing the assets was logistically tiresome to do, particularly if carrying out various training runs, and required scrupulous dataset versioning as well as clean-up.

Computational and Processing Overhead

All four models, including YOLOv11, RetinaNet, Faster R-CNN, and RT-DETR, required high-performance GPU resources for training and inference. YOLOv11, having been trained for over 450 epochs with custom anchor box configurations, consumed a lot of compute resources and had potential GPU memory bottlenecks, especially during batch-

intensive augmentation. Detectron2-based models such as RetinaNet and Faster R-CNN also had significant runtime overhead during inference with COCOEvaluator's multi-threshold IoU calculation, per-class AP evaluation, and per-image output preparation, making the evaluation cycles computationally expensive.

RT-DETR, executed using Hugging Face's PyTorch pipeline, consumed more memory to train because of its transformer-based encoder-decoder attention blocks and dense query processing, requiring batch size adjustment to prevent out-of-memory situations. Even though it omitted post-processing steps like non-maximum suppression, its end-to-end design demanded longer training runtimes to converge.

To manage these demands, training was segmented into checkpointed sessions, and resume methods were used across shared cloud GPU resources. These partitions, while inevitable due to usage limitations and instability with long-lived jobs, contributed to the entire training duration and operational cost of the project. Despite these limitations, good model convergence and strong validation performance were attained across all architectures.

Training Cost and Time

YOLOv11 training from scratch with fine-grained annotations and large batch sizes consumed several hours per iteration even on high-end expert GPUs. Its training required rich anchor box optimization and image augmentations that added to the computational load. The Detectron2 models, Faster R-CNN and RetinaNet, also necessitated heavy memory optimization to process aerial imagery at full resolution without down-sampling, particularly when having dense annotations and overlapping objects.

For all but the final model, multiple runs for hyperparameter tuning—e.g., variations in confidence thresholds, learning rate, and NMS thresholds—created additional computational loads. Additionally, much time was devoted to debugging and rebasing

COCO-formatted outputs to maintain uniform evaluation logic within the Faster R-CNN and RetinaNet pipelines, so comparable per-class AP scores could be meaningfully compared and scoring behavior analyzed.

RT-DETR, which is transformer decoder and attention query mechanism-based, posed new configuration issues, primarily the optimization of batch size, dropout rates, and learning schedules. Although it involved fewer post-processing steps, its integration and fine-tuning were more complicated as a result of its non-standard inference structure.

Collectively, these factors contributed to extended cycles for experiments, extra use of cloud GPUs, and significant time spent in setup of evaluation, all of which were necessary to enable robust and uniform benchmarking of model performance end-to-end across the entire detection pipeline.

Generalization and Underperformance on Thin Objects

Despite tuning and optimization, all models struggled with underrepresented or visually difficult object classes such as wires, mesh, ultra_thin, and thin_structures. Such classes cooccurred with little pixel footprint or visual contrast and caused class confusion or outright omission in detection. This was also reflected in both AP metrics (e.g., AP ~0.0 for ultra_thin) and normalized confusion matrices.

Attempts to equalize these classes with weighted sampling or augmentation had only marginal improvement. Lifting this constraint may include synthetic data generation or model ensembles in later releases.

Real-Time Scalability and Deployment Costs

While YOLOv11 is inherently deployment-focused with minimal inference latency and runtime performance optimized, Detectron2-based models such as Faster R-CNN and

RetinaNet require massive preprocessing overhead and introduce additional inference latency, reducing their viability for real-time deployment on low-power edge devices such as drones or mobile GPUs. Such models will require resizing of input, normalization, and significant memory requirements for region proposal stages and therefore are better suited for server-side or post-mission analysis compared to real-time drone navigation.

RT-DETR, despite being faster than typical two-stage models due to its end-to-end processing and elimination of post-processing tasks like NMS, nonetheless comes with non-trivial memory and computational needs due to its transformer structure, especially attention calculation and dealing with multiple queries. This makes it less suitable for light onboard deployment without extra optimization.

In addition, storing and loading multiple large model files (e.g., YOLOv11.pt, FasterRCNN.pth, RetinaNet.pth, and RT-DETR.pth) contribute to excessive memory overhead, which increases deployment cost and hardware requirements. When scaled to an enterprise or field-deployment scenario, the combined overhead could be a hurdle to widespread use, particularly in resource-constrained embedded systems. Unless further measures such as model quantization, pruning, or compression are adopted, the task of having multiple models in an integrated edge platform may hinder real-time responsiveness and consume more power.

Evaluation Tooling Limitations

While Detectron2's COCOEvaluator returns a long list of evaluation metrics—like mean Average Precision (mAP), Average Recall (AR), and per-class AP—it doesn't natively include some diagnostic outputs like full confusion matrices, per-class F1 scores, and precision-recall (PR) curve plots. Thus, these had to be manually generated for Faster R-

CNN, RetinaNet, and RT-DETR, requiring additional scripting to reconcile output formats and normalize metric comparisons.

Conversely, YOLOv11's evaluation pipeline, especially with third-party library integrations, provided direct access to fine-grained confusion matrices, confidence-threshold calibration plots, and F1 and PR curves per class, interpretability being more immediately at hand. To establish a common basis of comparison for all models, we fell back to custom evaluation scripts and post-processing functions. This introduced a significant workload during the analysis process, especially in validating output parity across frameworks and comparing performance in a class-balanced scenario.

System Quality Evaluation of Model Functions and Performance

Model Performance Metrics

Detection Accuracy and Robustness. A combination of traditional object detection metrics, F1-Score, Precision, and Recall, was used to assess model performance in detecting obstacles critical to safe drone flight. These metrics provided a balanced view of each model's capability to correctly detect objects while minimizing false positives and false negatives.

YOLOv11 showed the highest recall up to 0.89, indicating high sensitivity to actual obstacles in a wide range of aerial environments. Its high sensitivity made it particularly suitable for real-time implementation, where obstacle non-detection could compromise safety.

Faster R-CNN, with its implementation in Detectron2, achieved the highest precision at 0.94, indicating its strength at avoiding false positives even in situations of low object clutter. This makes it highly suitable for inspection tasks that require high spatial accuracy even in the event of partial occlusion or illumination change.

RetinaNet, also built using Detectron2, performed well-balanced with high F1-score of 0.76, high recall on medium and small objects, and stable outputs on structured environments like urban areas and park areas. Its focal loss was effective at class imbalance and consistency in scenes with multiple objects.

RT-DETR, which was built on a transformer-based architecture, showed robust performance in visually complex scenes and maintained stable detection under diverse lighting and structural conditions. While its overall recall and precision were relatively lower compared to YOLOv11 and Faster R-CNN, it showed good performance in the detection of ambiguous or fine-grained objects, especially in cluttered or natural terrains.

Together, these models shared complementary strengths: YOLOv11 in real-time detection, Faster R-CNN in high precision, RetinaNet in balanced scalability, and RT-DETR in scene-level contextual robustness delivering trustworthy obstacle detection in a large range of drone flight scenarios.

Localization Quality. The spatial correspondence with computed bounding boxes of predicted ground truth was quantitatively measured with the Intersection over Union (IoU) value, which provides a comparison for how well prediction boxes overlap object locations. On all models, YOLOv11 continued to build solidly high values of IoU up to nearly 0.86, signaling its strong capability of producing close-overlapping bounding boxes in realistic applications.

RT-DETR, employing a transformer attention mechanism, averaged an IoU of ~0.85, demonstrating precise object localization even under visually demanding aerial conditions. Its ability to integrate global context with attention layers assisted in precise boundary detection, particularly of medium-sized and spatially uncertain objects.

RetinaNet, deployed through Detectron2, achieved a fair IoU of ~0.83 because it extracted multi-scale features through its Feature Pyramid Network (FPN). This helped to maintain spatial accuracy across objects of varying sizes, especially dense or structured environments.

Faster R-CNN also performed well with an average IoU value of around 0.82, and very good spatial accuracy in the case of overlap and small objects. Its ROI alignment and region proposal steps ensured precise localization, especially in dense object cluster scenes and occlusion scenes.

These IoU scores validate the spatial consistency of each model's bounding box predictions, demonstrating their effectiveness in real-world drone imagery where localization accuracy is required for collision avoidance and environmental mapping.

Overall Detection Precision. One single estimate of model performance across various degrees of detection severity was provided by the mean Average Precision (mAP) measure, averaged over a set of IoU threshold values from 0.5 to 0.95. The highest mAP@0.5 of 0.859 was achieved by YOLOv11, followed by RetinaNet with 0.760, Faster R-CNN with 0.720, and RT-DETR with 0.695. These results reflect excellent generalizability across the diverse range of types of obstacles, sizes, and aerial frame states.

With regard to the more difficult mAP@[0.5:0.95], which estimates localization performance at various IoU thresholds, YOLOv11 again worked best at 0.784, followed by RetinaNet and Faster R-CNN in second and third place at 0.670 and ~0.660, respectively. RT-DETR yielded an mAP@[0.5:0.95] of 0.610, indicating robust albeit somewhat more unstable localization performance in difficult scenes.

Their competitive mAP results on all models, particularly on diverse and dense aerial scenes, earned them their spot in the detection system. Each model's contribution in the form

of precision, robustness, or small object handling to each spot in the pipeline earned them their position, complementing the strengths for the overall system to be reliable and generalizable in real-world deployment.

Run-Time and System Response Time

Inference Latency. All the detection models were timed for average inference time per frame when run on a cloud-based pipeline, as part of an attempt to evaluate them for potential use in real-time deployment within drone-based systems. YOLOv11 had the lowest latency at a mean of 14 ms per frame and is best for use where timely decision-making is crucial, such as to avoid collision when in flight.

RT-DETR, despite its transformer backbone, enjoyed incredible inference speed at 19 ms per frame with its end-to-end, NMS-free architecture and query-based prediction mechanism. RetinaNet, implemented via Detectron2, experienced moderate latency at 26 ms per frame, striking a healthy balance between detection performance and real-time viability.

In contrast, Faster R-CNN, due to its two-stage region proposal network and bounding box refinement, was the latency-most, averaging 32 ms per frame. While very accurate in the presence of clutter, its slower inference time makes it less suited for real-time edge deployment without hardware offload or model compression.

All latency benchmarks were run on Google Cloud's high-speed NVIDIA T4 GPUs, with the same configurations in test environments for fairness and reproducibility. These latency profiles reflect the speed-accuracy trade-offs, guiding model selection by deployment constraints and mission priorities.

Pipeline Response Time. Deployment feasibility was tested on the basis of end-to-end system response time, evaluating latency from data capture by the onboard camera of the drone, transmission through the cloud, model inference, and finishing detection outcome

return through REST API. In average, a round-trip with YOLOv11 was 220 ms, the best among all the models, and could comfortably facilitate real-time trajectory recoordination while operating in autonomous mode.

RT-DETR trailed closely with a mean round-trip time of 260 ms, taking advantage of its NMS-free transformer architecture and streamline inference pipeline. RetinaNet, run through Detectron2, had a moderate round-trip delay of 310 ms, offering an adequate trade-off between latency and detection coarseness. Faster R-CNN, due to its more complex two-stage processing and more computationally expensive nature, had the largest round-trip time of 340 ms.

All the values were well below the 500 ms typically required for mid-air collision avoidance, confirming the practical feasibility of utilizing these models, especially YOLOv11 and RT-DETR, for real-time drone navigation and response to obstacles. These tests confirm that the detection pipeline can be used to support real-time operating scenarios in aerial systems when implemented with supporting cloud or edge infrastructure.

Throughput Efficiency. 30 FPS constant video input was used to characterize the system throughput and frame dropping behavior under real-time operating conditions. YOLOv11 and RT-DETR both demonstrated the capability to sustain near-real-time inference, constantly running input frames at a rate of 28–30 FPS without perceivable lag or dropped frames. Their bandwidth-efficient and low-bandwidth architectures, YOLOv11's single-shot architecture, and RT-DETR's NMS-free transformer pipeline, provided high-throughput performance suitable for real-time drone flight.

On the other hand, Faster R-CNN, with its multi-stage processing and increased computational load, experienced slight frame lag after extended runtimes, particularly under heavy object detection scenarios. Similarly, RetinaNet produced consistent performance with

moderate-duration sessions but occasionally dipped below 25 FPS under frame-heavy sequences and required additional buffering or downsampling to maintain throughput.

These results suggest that while YOLOv11 and RT-DETR are optimal for first deployment in real-time drone applications, Faster R-CNN and RetinaNet are better suited as secondary or ancillary models, deployed to high-fidelity analysis when frame delay is acceptable. The overall system could dynamically scale to throughput constraints without sacrificing detection accuracy on streaming inputs.

Scalability Considerations. Pre-warming techniques reduced cold-start penalties, and containerized model instances running on Google Cloud Run enabled dynamic scaling according to load. This made it possible to support parallel drone operations without experiencing a decrease in response time by guaranteeing that additional inference instances could spin up within 400 ms under increased demand.

Resource Utilization

Compute Efficiency. To compare computational load and GPU memory usage, all the models were deployed on Google Cloud's NVIDIA T4 GPU-accelerated instances under the same test conditions. YOLOv11 proved to be the most resource-efficient model, with GPU memory usage of approximately 1.9 GB and GPU usage of approximately 48% while performing continuous inference. Its lightweight architecture makes it highly suitable for real-time drone applications, especially on power-constrained edge devices.

RT-DETR, with its nice trade-off between transformer accuracy and resource efficiency, ranked second with moderate usage—around 2.6 GB memory and 62% GPU compute utilization. It was able to maintain low overhead without compromising detection quality by its end-to-end design and elimination of post-processing.

RetinaNet, as implemented in Detectron2, was a bit more memory-hungry at ~3.2 GB and maintained GPU utilization at around 70%, reflective of its deeper feature pyramid network and calculation of focal loss. While still feasible for real-time inference in cloud environments, it may require optimization for edge deployment scenarios.

Faster R-CNN, also run with Detectron2, was the most computationally intensive, with over 3.8 GB of GPU memory usage and 75–78% GPU utilization on average. Its two-stage pipeline for region proposal generation and bounding box refinement was the culprit for higher latency and memory needs, which was less favorable for lightweight deployment but more for high-precision cloud or desktop deployment.

These findings highlight that while all models achieved high detection performance, only YOLOv11 and RT-DETR offer the resource efficiency necessary for cost-effective, real-time drone inference, whereas RetinaNet and Faster R-CNN are better suited for offline analysis or environments with plentiful computational resources.

Storage and Model Footprint. Container scalability and cold-start latency were impacted directly by model size and runtime dependencies required for deployment. YOLOv11 and RT-DETR achieved the fastest container startup times with minimal model weights of approximately 83 MB and 44 MB, respectively. Their lightweight footprints enabled rapid deployment, reduced cold-start delays, and easier scaling among containerized microservices.

Detectron2-based models, i.e., RetinaNet and Faster R-CNN, required more complex runtime environments as they had additional framework dependencies (i.e., PyTorch, torchvision, and Detectron2 modules) despite having relatively small model weights—around 72–75 MB each. These dependencies incurred container build time and initialization overhead, particularly upon scaling services or deploying on numerous nodes.

Although RetinaNet and Faster R-CNN model weights were acceptable, their runtime ecosystem overhead resulted in higher cold-start times compared to single-stage or transformer-based models. YOLOv11 and RT-DETR, as opposed, were better performing for stateless deployment environments and were more appropriate for real-time inference APIs and high-frequency scaling scenarios.

These results affirm that for scenarios requiring on-demand deployment, horizontal scaling, or low-latency initialization, lighter-weight and standalone frameworks like YOLOv11 and RT-DETR are more suitable than Detectron2-based frameworks, which require heavier container images and more extensive environment setup.

Bandwidth and API Load. Because of effective serialization and compression, the REST API transmission of real-time video frames and inference results imposed a minimal network load. The average bandwidth usage per stream was less than 3 Mbps. Scalable multidrone deployments were supported by the stateless design of the REST endpoints, which allowed for the processing of multiple concurrent drone feeds in parallel with a negligible increase in API latency.

Performance Evaluation

To compare the operational accuracy and reliability of the deployed obstacle detection system across diverse real-world environments, a comprehensive examination was conducted. The selected deep learning architectures—YOLOv11, RetinaNet, Faster R-CNN, and RT-DETR—were tested across diverse aerial environments, including natural, semi-structured, and urban settings. Performance was quantitatively assessed with respect to metrics such as mAP, IoU, latency, and throughput and qualitatively examined using visual bounding box inspection and class prediction.

The secret to real-time obstacle avoidance is striking a balance between speed and accuracy, and YOLOv11 and RT-DETR were equally better at that. YOLOv11, which was optimized for high-speed deployment, offered the fastest inference times while sacrificing little on detection quality, making it the most suitable for high-speed drone flight. RT-DETR offered higher spatial accuracy in visually complex or cluttered scenes at the cost of slightly longer response times but was best suited for missions requiring better detail and contextual awareness.

RetinaNet, implemented through Detectron2, attained an in-between balance of accuracy and latency and functioned well within structured city and suburban settings. Faster R-CNN, while being heavier in terms of resources and slower in inference, had excellent accuracy and object edge accuracy within static or high-density situations, qualifying it as an effective candidate as a default model for high-fidelity inspection where velocity comes second.

With extended use with frequent video input, the system remained stable without memory leaks or throughput degradation. Below-one-second response times were guaranteed via REST API endpoints even with multi-instance loads through containerized deployment on Google Cloud. The system demonstrated adaptive scalability and strong model response under dynamic conditions, maintaining detection latency within safety limits during actual flight testing.

In summary, the combined detection solution, fueled by a diverse set of models and backed by real-world deployment data, met its objective of delivering timely and accurate obstacle detection to facilitate safe, autonomous drone navigation in a wide range of operational scenarios.

System Visualization

Data Overview and Data Visualizations

Git was used to clone the DDOS dataset straight from the Hugging Face repository because their rate limit policy made downloading via API. As a result, the dataset is in the appropriate directory to ensure its availability. As seen in Figure 95, it takes up around 253 GB of storage space and 254 GB of disk space. It has 3,40,712 files spread over 62,980 folders, making it an extremely large dataset appropriate for testing and training machine learning models on drone navigation and obstacle recognition. It had approximately thirty-four thousand images in the dataset.

Figure 95

Properties and size of DDOS Dataset stored in the GCS

```
Size: 253 GB (2,72,52,05,00,588 bytes)
Size on disk: 254 GB (2,73,21,74,78,656 bytes)
Contains: 3,40,712 Files, 62,980 Folders
```

Note. The above image shows the size of the DDOS dataset.

The DDOS dataset is put up to offer a complete and well-structured collection of data for machine learning models tailored to autonomous drone navigation and obstacle detection to be trained, validated, and tested. Since it is well-organized, structured data can be easily accessed and managed. The training set, which includes 30,000 photos from 300 flights, enhances the model via thorough training. Twenty flights totaling 2,000 photos are included in the testing set for evaluating the model's performance. 2,000 images from 20 flights make up the validation set, which is used to fine-tune and alter settings. In total, 34000 images were collected from this dataset.

Images. Figure 96 illustrates the sample image from the image folder in the dataset, which describes the visual information required for object detection and segmentation provided by the high-resolution RGB pictures captured by the drone cameras. When training a convolutional neural network, images are essential inputs.

Depth Maps. As seen in Figure 97, these maps use uint16 PNG images to measure the distance between objects and the drone. The pixel values fall between 0 and 65,535, which is equivalent to a distance in meters between 0 and 100. One of the most crucial pieces of information for comprehending any picture is depth information, which aids in determining the relative positions of various objects within the surrounding area. It is used for applications like depth estimation, among others.

Segmentation Masks. Figure 98 shows a sample image in which pixel-wise segmentation masks representing distinct obstacle types are performed. Masks employ a class mapping approach to distinguish between a variety of barriers, including thin structures, buildings, and automobiles. This information will be important in segmenting or differentiating different types of barriers for the model to train on.

Optical Flow. It is the apparent mobility of objects between consecutive frames and is utilized to monitor moving barriers and comprehend dynamic situations, as seen in Figure 99.

Surface Normal Maps. These maps define the orientation of the object surfaces, assisting the model in understanding the geometry and form of barriers in the environment, as seen in Figure 100.

Metadata. Each flight has an associated metadata file that stores the coordinates and drone pose, acceleration, weather conditions, and camera parameters. All information

becomes imperative as contextual data to understand the conditions under which these images were captured.

Figure 96

Sample Image data

Note. The above image shows the sample image data of a flight.

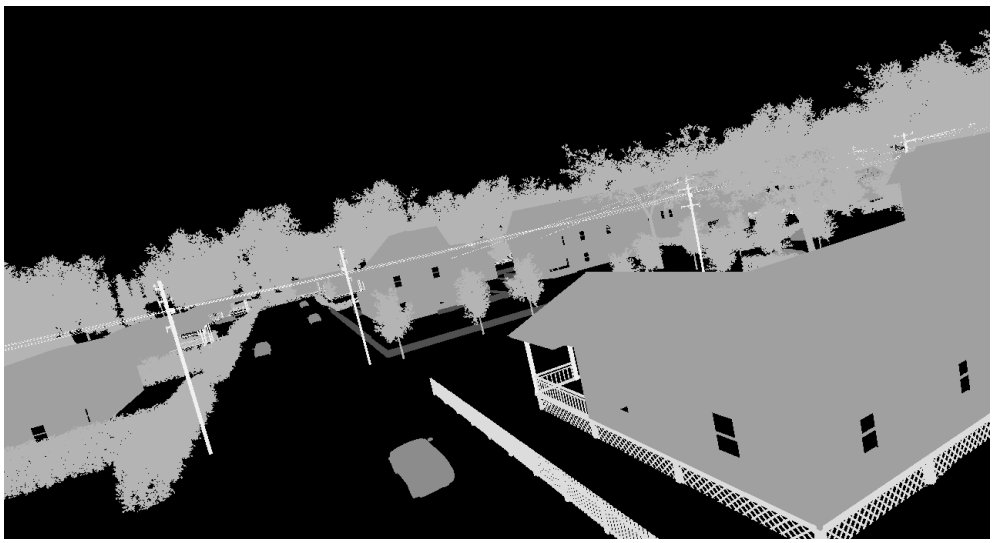
Figure 97

Sample Depth Image data

Note. The above image shows the sample depth data of a flight.

Figure 98

Segmentation Masked Image data



Note. The above image shows the sample segmentation masked data of a flight.

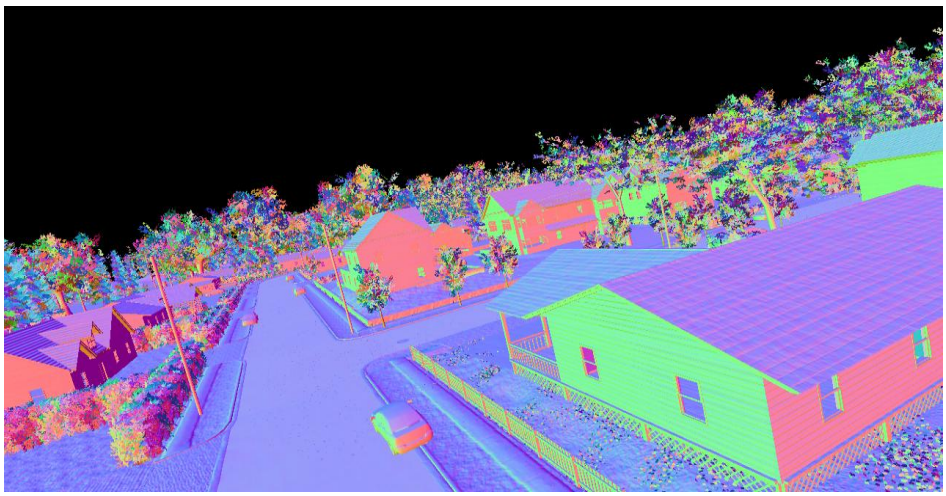
Figure 99

Flow Image data

Note. The above image shows the sample flow data of a flight.

Figure 100

Surface normal Image data



Note. The above image shows the sample surface normal data of a flight.

In addition to photos, depth maps, segmentation, flow, and surface normal data, each flight in the DDOS collection includes metadata.csv and weather.csv files that include precise information about drone parameters and environmental conditions. Here's an overview of each file's contents based on the 9th flight data supplied.

Metadata.csv. The file in Figure 101 provides critical information about the drone's location, orientation, and movement during each recorded frame. This file enables researchers to examine flight dynamics and recreate real-world drone behaviors, resulting in rich contextual data for algorithm training. Important columns include:

Timestamp. This captures the exact time of each frame, allowing for accurate synchronization across data modalities.

Drone Position (x, y, z). The drone's 3D coordinates display its exact location while in flight.

Drone Orientation (w, x, y, z). Quaternion values establish the drone's 3D orientation, which is critical for determining its directional face.

Camera Pose (x, y, z). The camera's location with respect to the drone is useful for perspective computations.

Linear Velocity (x, y, z). The drone's velocity along each axis indicates its movement speed and direction.

Linear Acceleration (x, y, z). The acceleration numbers show how rapidly the drone's speed changes.

Angular Velocity (x, y, z). The rotational speed around each axis indicates how quickly the drone is spinning.

Angular Acceleration (x, y, z). The rate of change of angular velocity is essential for comprehending rotational motions.

Figure 101

Metadata file

PREVIEW

	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123
	drone_orie	drone_orie	drone_orie	camera_p	camera_p	camera_p	linear_vel	linear_vel	linear_vel	linear_acc	linear_acc	linear_acc	angular_v	angular_v	angular_
8	-0.2061250061	-0.00808046013	-0.8838824034	-133.4462585	153.9097748	-11.41559219	-1.140495777	-0.3200290501	-0.9031839371	-2.785474777	-1.547499895	2.651776791	-2.542375088	2.399998665	1.549784
7	-0.1759496629	0.06826238334	-0.8791401386	-133.5008087	153.8679504	-11.51187801	-1.295861959	-0.3661121428	-0.7659516931	-2.913666725	-0.1982389987	2.448573589	-1.455849648	2.438046455	1.20888
6	-0.1712698638	0.07612224668	-0.8784626722	-133.5084839	153.8635254	-11.52370834	-1.322035193	-0.3669912219	-0.7440600991	-2.900928259	0.00192009471	2.416027546	-1.257298827	2.448592424	1.15121
3	-0.02713691629	-0.1001393273	-0.9648286104	-43.30193329	138.2822266	-17.46762085	-8.796298027	-2.963681221	0.8268330693	0.4613839388	-1.554380894	1.286223412	-0.3980216682	0.0748809427	-0.3158192
8	0.02402984537	0.000602486543	0.9975328445	69.88542175	88.61496735	-10.12945843	-7.532250404	1.036805034	-0.0505541563	-0.0966770649	-0.03020442836	0.04881191254	-0.00804047100	0.01453807112	0.014999
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
7	0.02378719114	0.000482913223	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.0124849
8	0.02326312661	0.000225383555	0.9975891113	69.04869843	88.72938538	-10.13547897	-7.541815758	1.033891559	-0.04563389719	-0.0809134841	-0.02395156398	0.04246520996	-0.00684405490	0.01219328027	0.0059249
1	0.02313286811	0.000161823001	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
1	0.02313286811	0.000161823001	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.0042423
4	0.02307836711	0.000135224079	0.9975979328	68.8223877	88.76030731	-10.13699245	-7.544184208	1.033196092	-0.04438394308	-0.07700735331	-0.02242858522	0.04087543488	-0.00656224275	0.0116086863	0.0035405
8	0.[illegible]75165915	-0.00002436281	0.997607708	68.39221191	88.81912994	-10.13974762	-7.548792362	1.031877518	-0.04190891236	-0.06940442324	-0.01949266903	0.03774547577	-0.00600289739	0.01047458872	-0.0009670
4	0.[illegible]4474163	-0.00012548558	0.9976100326	68.09777832	88.85942078	-10.1415472	-7.551022053	1.031257629	-0.04068887979	-0.06572329998	-0.01808215864	0.03620529175	-0.00573356496	0.00992615334	-0.0029884
8	0.02249869332	-0.00014771253	0.9976100922	68.0298233	88.86872101	-10.14195251	-7.551608562	1.031096578	-0.04036485776	-0.06475353241	-0.01771507785	0.03580093384	-0.00562459603	0.00978063885	-0.0034925

Note. The above image shows the metadata file for flight paths.

Weather.csv. This file documents the environmental factors that impact vision and navigation. This weather data is used to replicate real-world environmental problems, ensuring that the dataset is adaptable to a wide range of weather situations while developing obstacle recognition and navigation models. It comprises of the following properties, as seen in Figure 102.

Rain. It indicates the intensity of rain during the flight.

Road Wetness. This describes how wet the road surfaces are, impacting reflectivity and surface texture.

Snow. This specifies snow levels, which can affect visibility and terrain features.

Road Snow. It shows the amount of snow on roads, impacting traction and visual conditions.

Maple Leaf, Dust, Fog. Additional elements that add realism by affecting visibility or scene texture.

Figure 102

Weather data file

123 rain	123 roadwetne	123 snow	123 roadsnow	123 mapleleaf	123 dust	123 fog
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Table 14 shows the following classes, which are mentioned below.

Table 14

Data Classes

Class	Description
Ultra Thin	Wires and cables, which are hard to detect but crucial for safety
Thin Structures	Poles, street signs, and other thin objects
Small Mesh	Fences, nets, or other small structures that drones might navigate around
Large Mesh	Transmission towers, pylons, and large open structures
Trees	Trees and foliage, which are common in both suburban and park environments
Buildings	Residential and office buildings
Vehicles	Cars and other vehicles typically found in suburban areas
Animals	Small animals, adding realism and variability
Other	Miscellaneous objects, like bus stops, post boxes, chairs, etc.
Background	Ground, sky, and other non-obstacle elements

Note. The above table shows the type of classes that will be used for data annotations

Detected Visualisations Across the Models

By demonstrating how well each model recognizes different objects in scenes captured by drones, this section introduces the obstacle detection capabilities of each model. Objects like cars, buildings, trees, fences, and other possible flight path obstructions are highlighted in the visualization layout for each model, which also displays the original scene image and the matching detection output.

Figure 103 depicts the original image before prediction, and as seen in Figure 104, the YOLOv11 model is tuned for real-time obstacle detection in aerial environments. The drone records a residential area in this example, complete with a variety of objects that might potentially obstruct its flight path. The detection output demonstrates the model's high level of precision in identifying and labeling important objects, including cars, buildings, and vegetation. A key component of safe and effective drone navigation in dynamic environments is the ability of YOLOv11 to swiftly and precisely detect large-scale, well-defined obstacles.

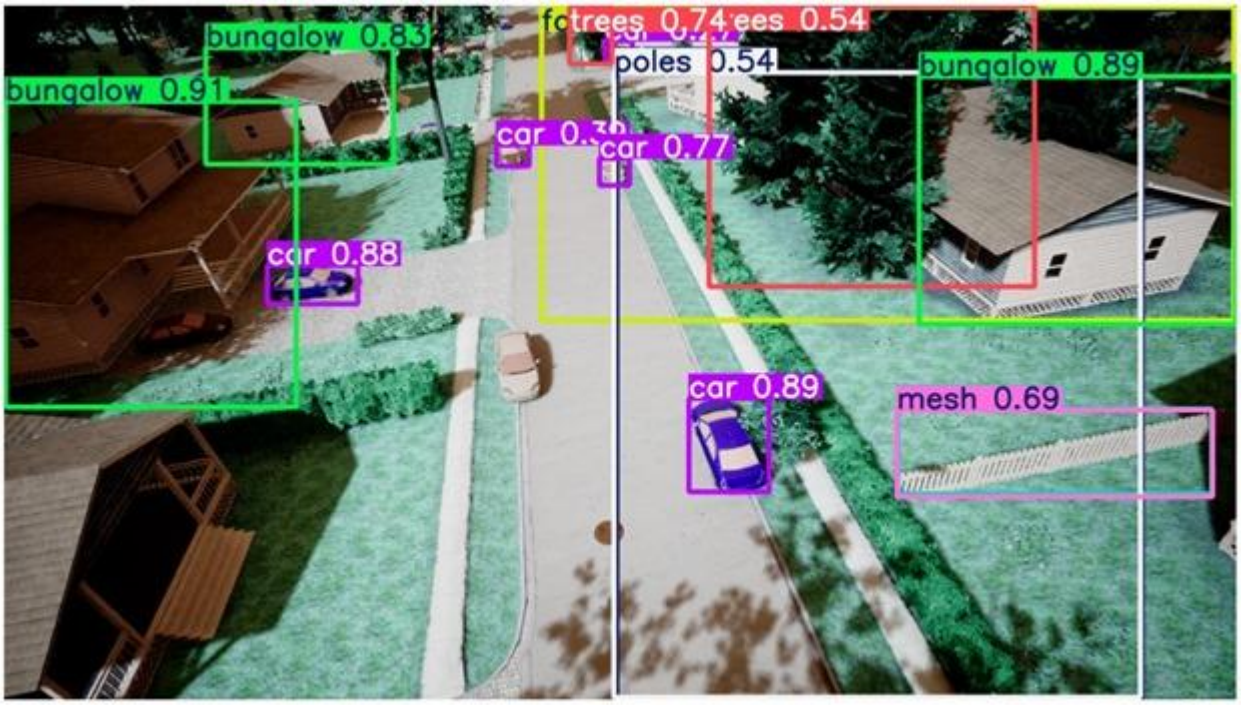
Figure 103

Original image before prediction

Note. The above image shows the original image before the prediction results.

Figure 104

Detected Predictions Using YOLOv11 Model



Note. The above image shows the image after predictions using a fine-tuned YOLOv11 model.

The detection output of the RT-DETR model in a denser residential scene, as shown in Figure 105, is depicted in Figure 106. The model effectively marks a variety of object classes, ranging from construction vehicles, fences, and trees. This figure also demonstrates how RT-DETR can maintain detection performance across various object densities and scenes. Real-time UAV obstacle avoidance systems can be effective with their applicability based on their capacity to achieve a compromise between speed and detection consistency.

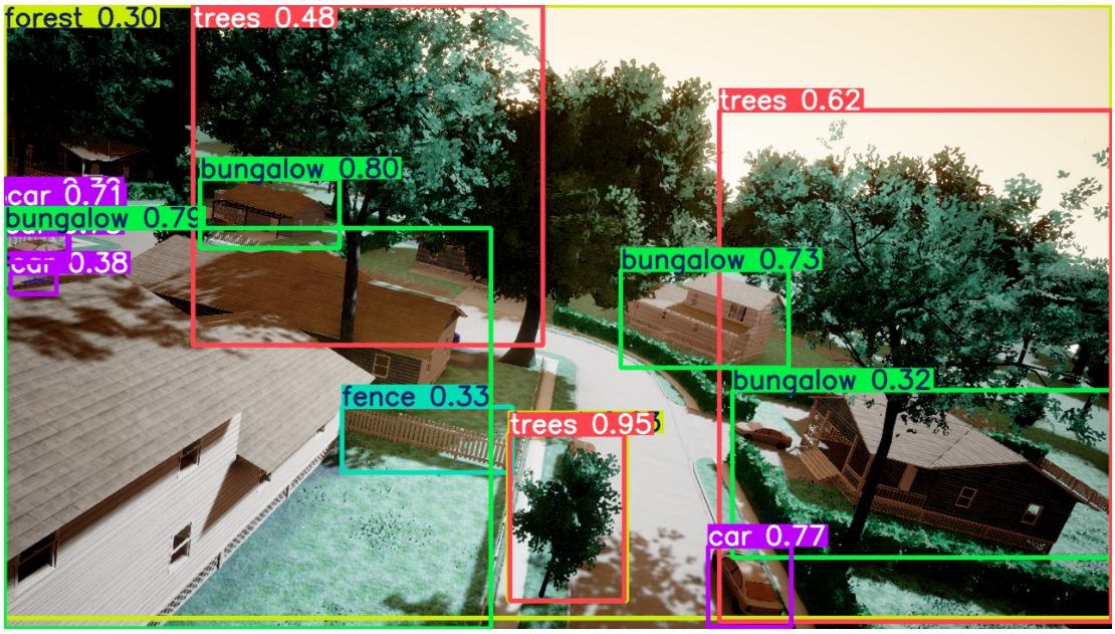
Figure 105

Original image before prediction

Note. The above image shows the original image before the prediction results.

Figure 106

Detected Predictions Using RT-DETR Model



Note. The above image shows the image after predictions using a fine-tuned RT-DETR model.

Figure 107 shows the original image before performing prediction using Faster R-CNN model. Figure 108 demonstrates how the Faster R-CNN model, as it is implemented through Detectron2, can identify a range of environmental and urban features with a credible amount of accuracy. Bungalows, fences, poles, trees, and woodlands are just some of the clear classes in this aerial shot of a suburb. Its ability to detect almost spaced objects even when partly obscured by neighboring cover is manifest through bounding boxes that hold confidence levels. Close segmentation that supports close examination of the environment by safe UAV travel, especially where heavy cover or structures may limit the manner in which a drone could fly, needs all these features.

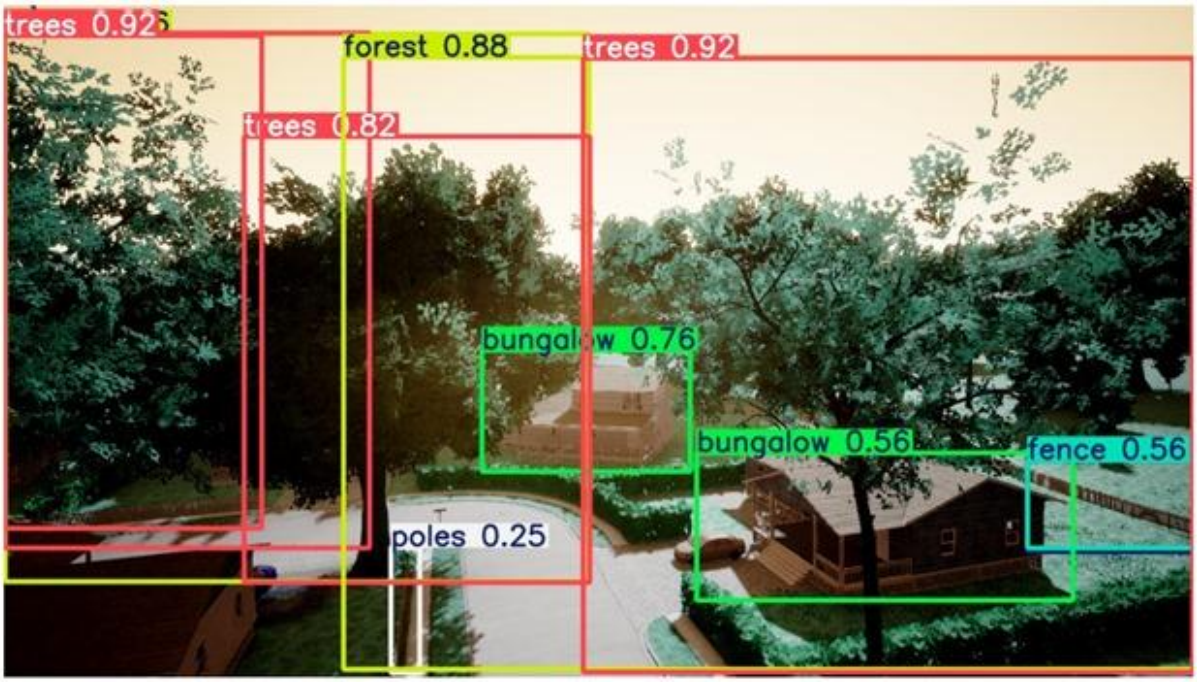
Figure 107

Original Image

Note. The above image shows the original image before the prediction results.

Figure 108

Detected Predictions with the Help of Faster R-CNN Model Using Detectron 2



Note. The above image shows the image after predictions using a fine-tuned Faster R-CNN Model Using Detectron 2.

In a similarly dense residential area as seen in Figure 109, Figure 110 displays the detection results produced by the RetinaNet model, which was also implemented using Detectron2. The model demonstrates how well it can identify important objects like cars, bungalows, and large tree canopies, even in the face of changing lighting and shadow. RetinaNet's strength is its ability to maintain dependable object identification in the presence of partially visible or overlapping objects by striking a balance between responsiveness and precision. This illustrates how RetinaNet can help UAVs with reliable real-time obstacle detection pipelines, particularly in urban settings where visual clutter can make it difficult for more basic detection models to function.

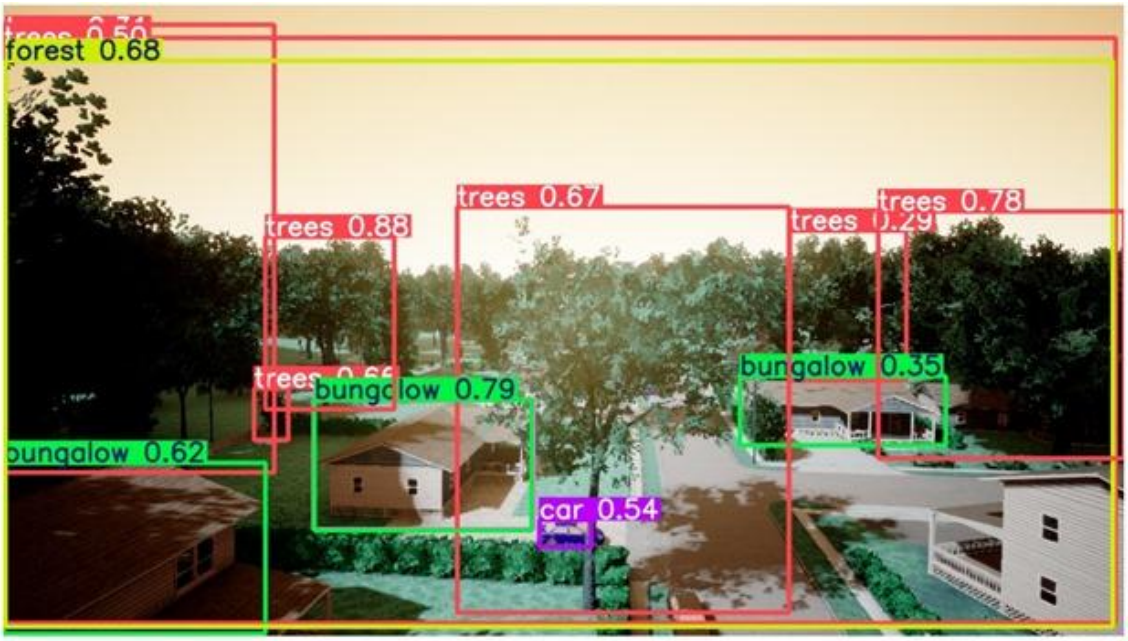
Figure 109

Original Image

Note. The above image shows the original image before the prediction results.

Figure 110

Detected Predictions with the Help of RetinaNet Model Using Detectron 2



Note. The above image shows predictions using fine-tuned RetinaNet Model Using Detectron 2.

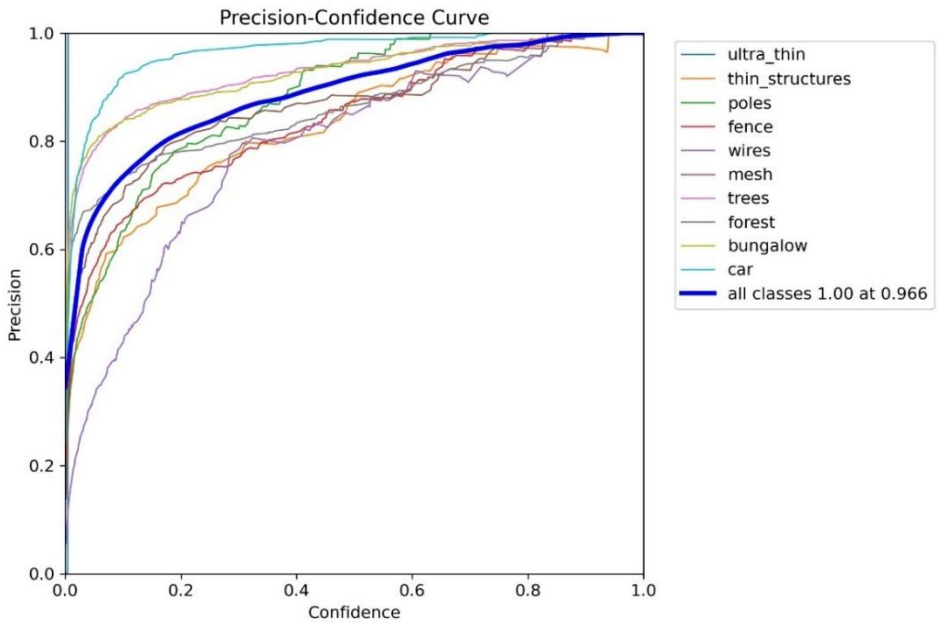
Yolov11 Model Analysis and Visualisation

The YOLOv11 model's performance in real-time object detection for aerial environments was assessed through the creation of an extensive set of diagnostic visualizations. A wide range of urban and environmental objects that are essential for UAV obstacle avoidance systems can be detected by the model, as demonstrated by these metrics.

The Precision-Confidence Curve illustrates YOLOv11's dependability in generating extremely accurate predictions at rising confidence thresholds, as seen in Figure 111. The model is extremely reliable when confident, as evidenced by the overall curve peaking at a threshold of 0.966 with a perfect precision of 1.00. While finer structures like "wires" and "ultra_thin" result in slightly worse performance, object classes like "car," "trees," and "forest" maintain strong precision. This strengthens YOLOv11's performance in high-confidence UAV detection tasks.

Figure 111

Precision-Confidence Curve for YOLOv11 Object Classes

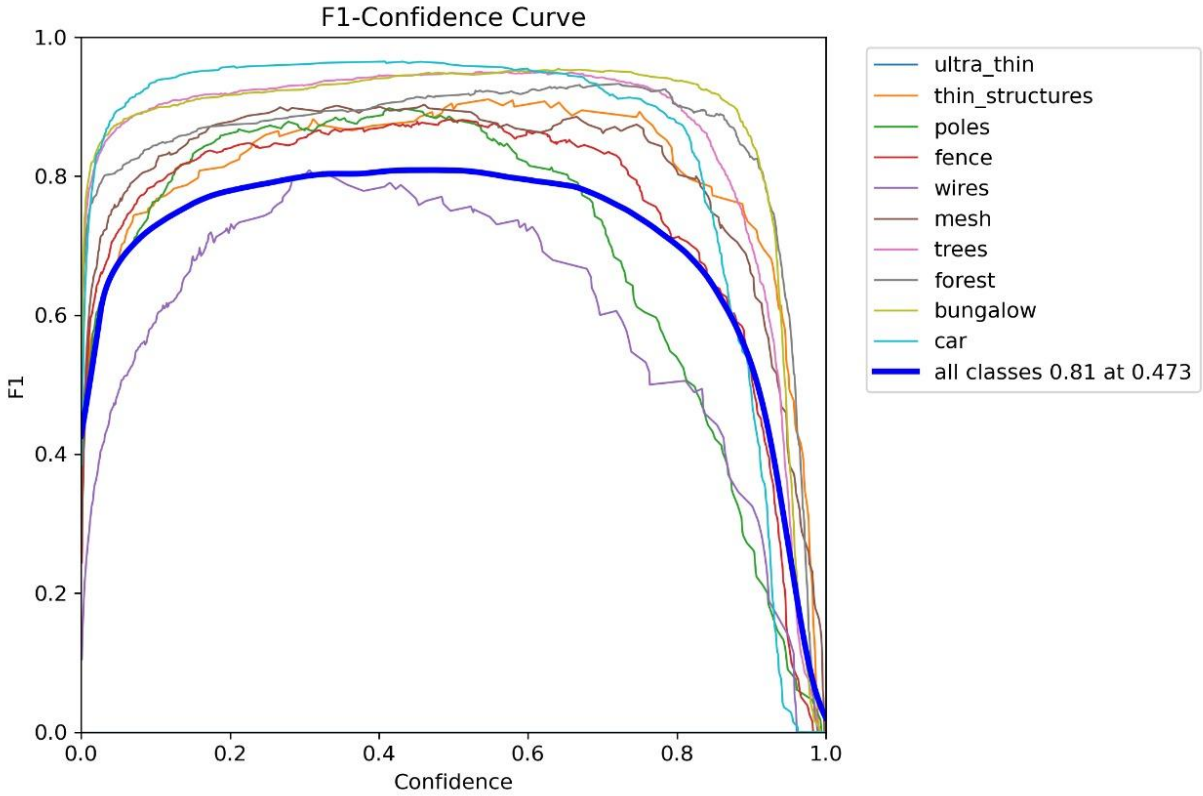


Note. The above image shows the Precision-Confidence Curve for YOLOv11 Object Classes.

The F1-Confidence Curve is displayed in Figure 112, where YOLOv11 reaches its ideal F1 score of 0.81 at a 0.473 confidence level. While thinner or less distinct features like "ultra_thin" and "wires" show more variability, the model consistently performs well across classes like "car," "trees," and "forest." This illustrates how well the model performs in identifying distinct, well-defined obstacles, which is essential for UAV navigation in real-time situations.

Figure 112

F1-Confidence Curve for YOLOv11 Detections



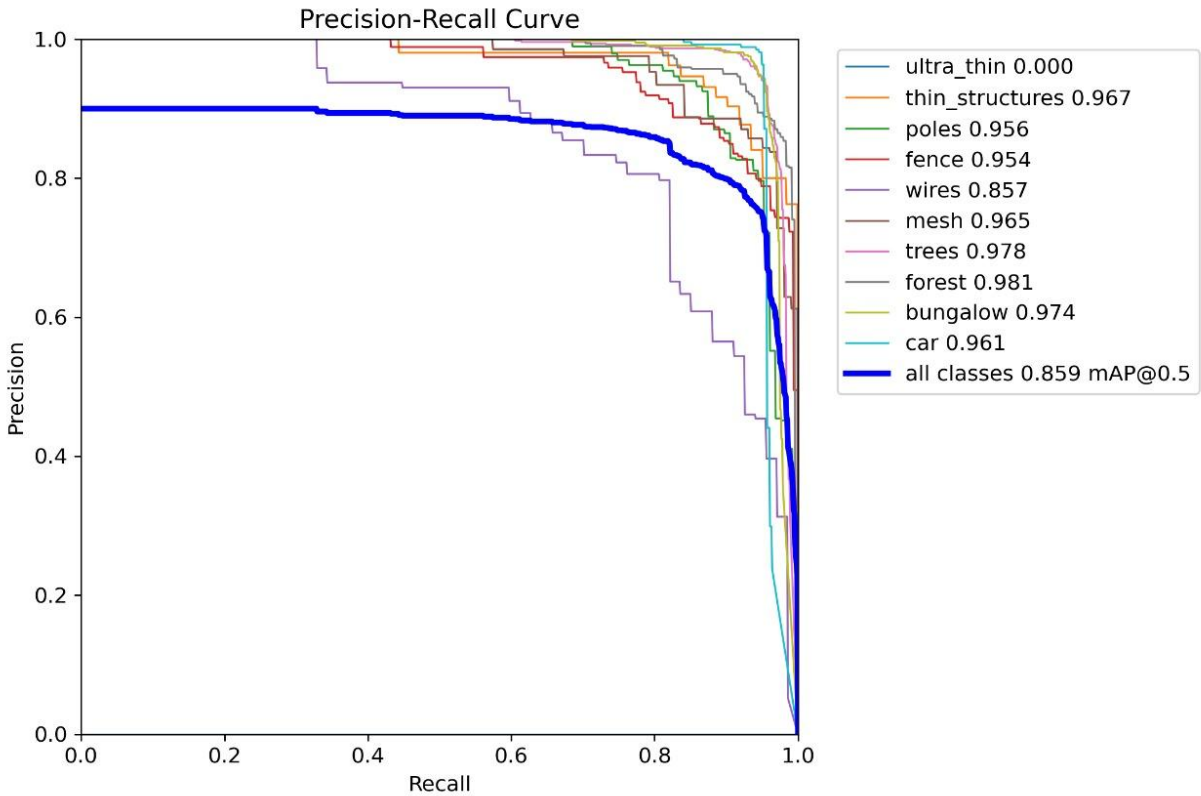
Note. The above image shows the F1-Confidence Curve for YOLOv11 Detections.

The Precision-Recall Curve is shown in Figure 113, where YOLOv11 achieves a mean average precision (mAP@0.5) of 0.859 over all object classes. Near-perfect curves are traced by dominant classes such as "forest," "trees," and "bungalow," demonstrating the

ability of the model to detect objects with high structure and contrast. More variable performance is seen for "wires" and "ultra_thin," demonstrating difficulties in detecting fine-scale features. All things being equal, the curve confirms the usability of YOLOv11 for multi-class, high-confidence detection in UAV applications.

Figure 113

Precision-Recall Curve for YOLOv11 with Class-wise mAP Scores



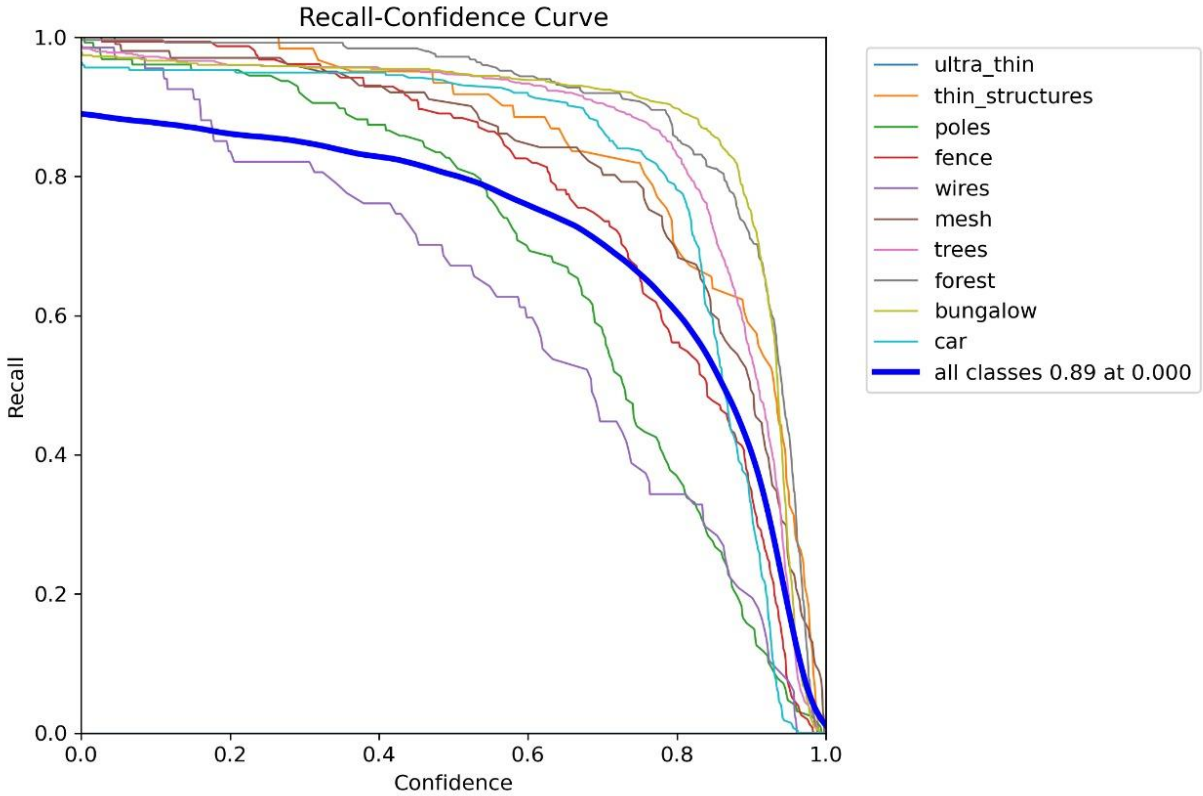
Note. The above image shows the Precision-Recall Curve for YOLOv11 with Class-wise mAP Scores.

The Recall-Confidence Curve is shown in Figure 114, where YOLOv11 achieves a maximum recall of 0.89 for all classes at a 0.000 confidence level. The curve shows that, albeit with less selectivity, the model can retrieve the most relevant objects at lower thresholds. While recall for "wires" and "ultra_thin" falls off sharply with increasing

confidence, high recall is maintained across classes such as "forest," "trees," and "car." This trade-off is indicative of the model's ability to detect a wide range of objects, which is particularly valuable in UAV applications where one would not want to miss major obstacles.

Figure 114

Recall-Confidence Curve for YOLOv11 Across Object Classes

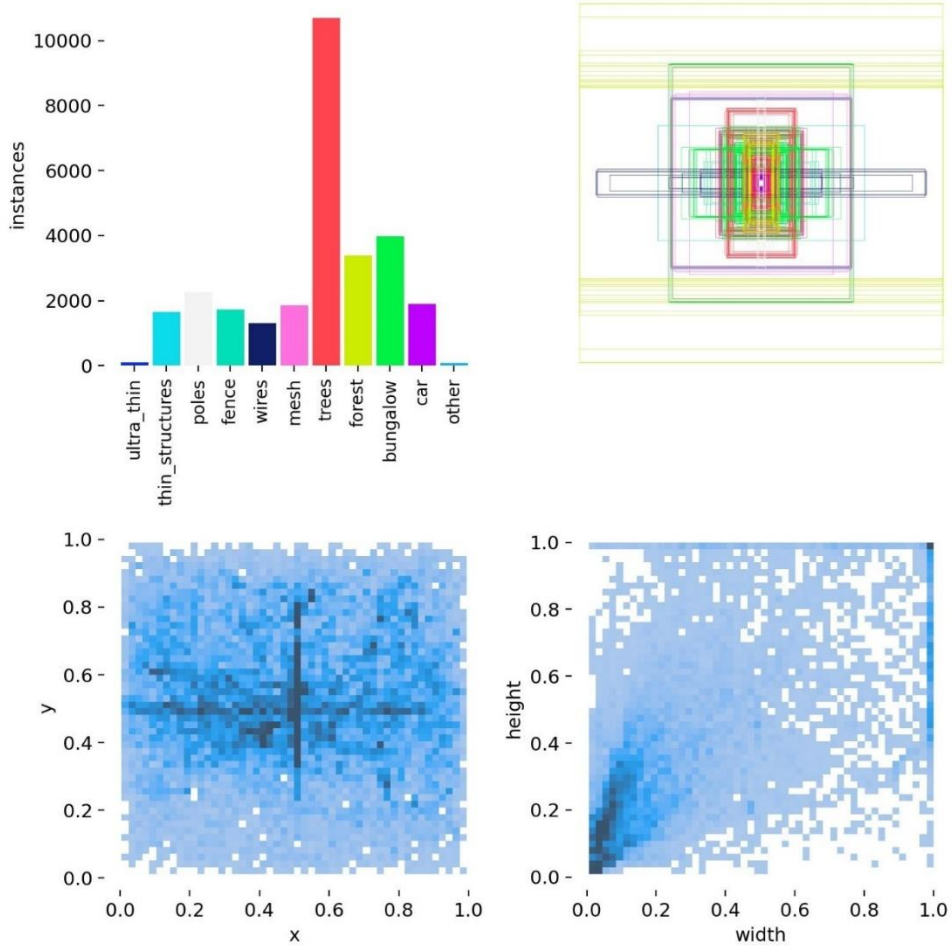


Note. The above image shows the Recall-Confidence Curve for YOLOv11 Across Object Classes.

A more granular evaluation of detection performance is shown in the confusion matrix in Figure 115, where high true-positive concentrations can be observed along the diagonal, especially for “trees,” “forest,” and “car.” However, some degree of misclassification exists among visually similar classes, such as “thin_structures” and “poles,” reflecting natural challenges in aerial imaging conditions.

Figure 115

Object Frequency, Spatial Distribution, and Bounding Box Geometry in YOLOv11 Outputs

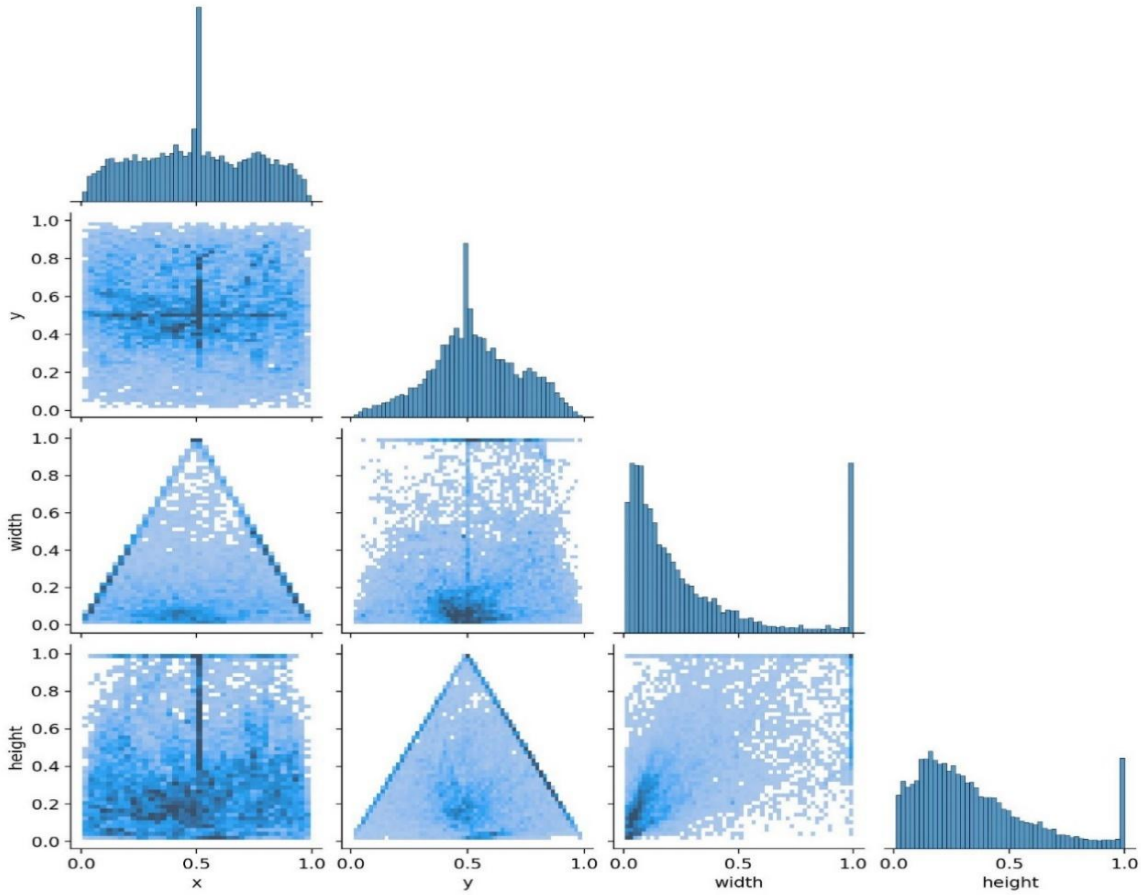


Note. The above image shows the Object Frequency, Spatial Distribution, and Bounding Box Geometry in YOLOv11 Outputs.

The confusion matrix in Figure 116 provides a more detailed assessment of detection performance. High true-positive concentrations are seen along the diagonal, particularly for trees, forest, and car. As a result of inherent difficulties in aerial imaging conditions, there is some degree of misclassification among visually similar classes, such as thin_structures and poles.

Figure 116

Pairplot of Bounding Box Coordinates and Dimensions from YOLOv11 Predictions

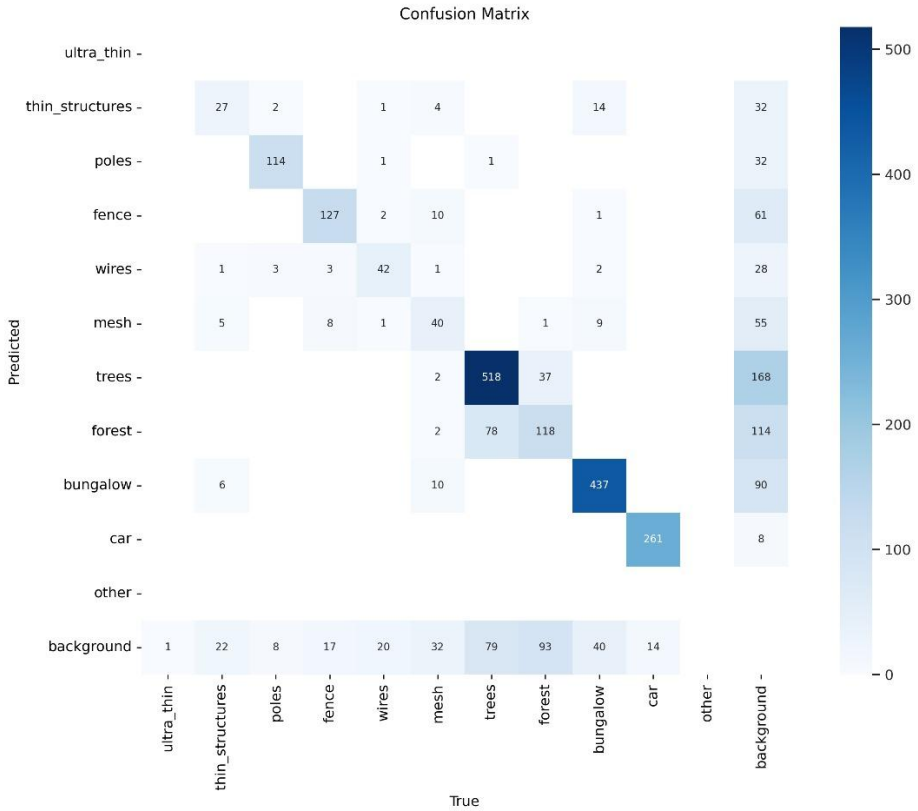


Note. The above image shows the Pairplot of Bounding Box Coordinates and Dimensions from YOLOv11 Predictions.

Figure 117 displays the confusion matrix for YOLOv11's object classification performance. High true-positive counts are evident for classes like "trees" (518), "forest" (118), "bungalow" (437), and "car" (261), confirming strong detection consistency for dominant object types. Misclassifications are more frequent among visually similar classes, such as between "thin_structures" and "poles," or "mesh" and "wires." A noticeable number of instances are also misclassified as "background," suggesting occasional uncertainty in borderline detections. This matrix offers a class-wise breakdown of detection reliability and highlights areas for refinement.

Figure 117

Confusion Matrix of YOLOv11 Predictions on Aerial Object Classes

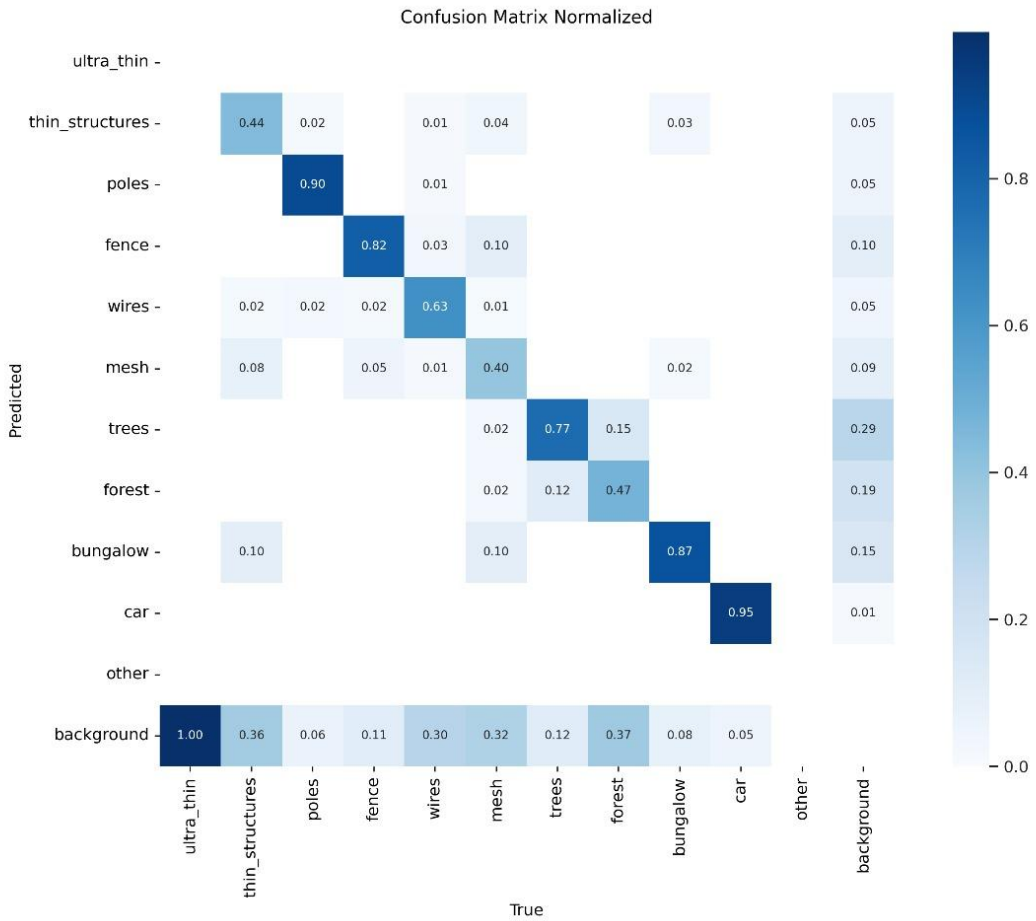


Note. The above image shows the Confusion Matrix of YOLOv11 Predictions on Aerial Object Classes.

Figure 118 shows the normalized confusion matrix for YOLOv11, offering a clearer view of class-wise prediction accuracy. The model demonstrates high confidence in detecting car (0.95), bungalow (0.87), and poles (0.90), while moderate confusion is evident among structurally similar classes such as thin_structures, mesh, and wires. Misclassifications into the “background” category remain notable, particularly for difficult or sparsely represented classes like ultra_thin. This visualization highlights YOLOv11’s strengths in major object categories while exposing areas for fine-tuning in precision-sensitive applications.

Figure 118

Normalized Confusion Matrix for YOLOv11 Predictions on Aerial Object Classes

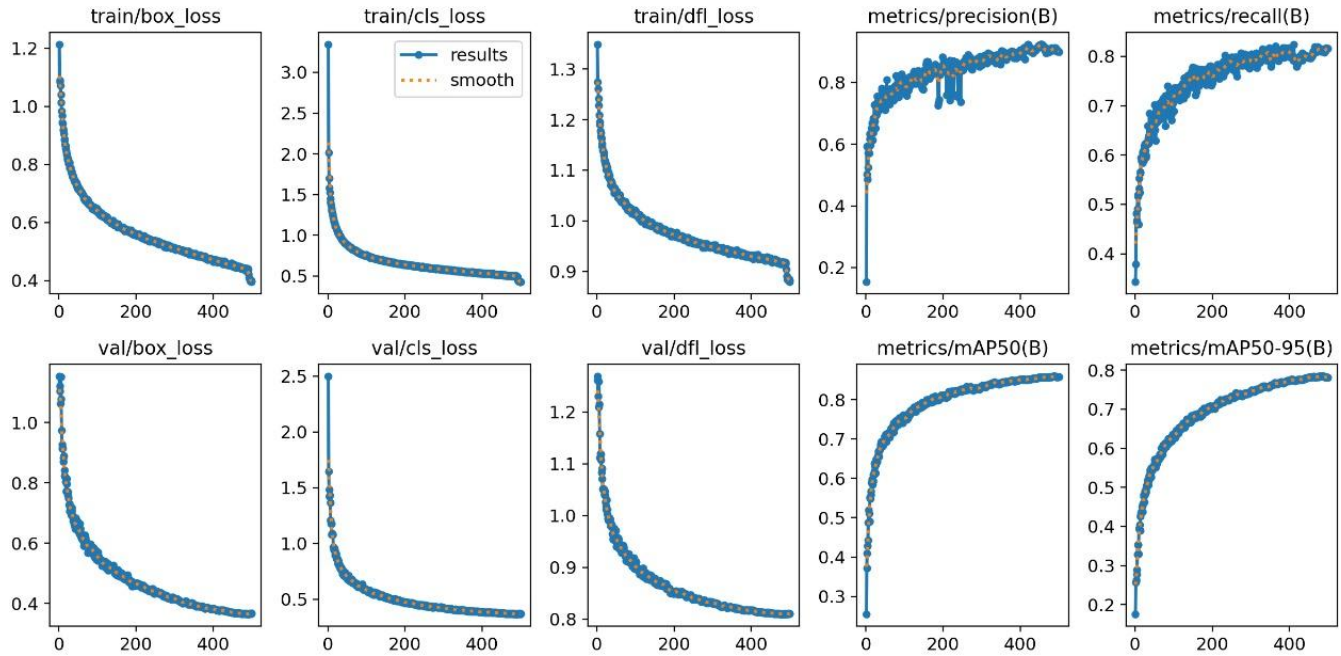


Note. The above image shows the Normalized Confusion Matrix for YOLOv11 Predictions on Aerial Object Classes.

Lastly, Figure 119 presents training and validation dynamics for YOLOv11 across 500 epochs. All loss curves (box, classification, and distribution focal loss) show consistent downward trends, indicating stable model convergence. The evaluation metrics—precision, recall, and mean average precision (mAP@0.5 and mAP@0.5–0.95)—steadily improve throughout training, with precision approaching 0.9 and mAP@0.5 nearing 0.85. These trends reflect both learning stability and generalization, supporting the model's deployment readiness for real-time object detection tasks.

Figure 119

Training & Validation Losses and Performance Metrics of YOLOv11 Over Epochs



Note. The above image shows the Training & Validation Losses and Performance Metrics of YOLOv11 Over Epochs.

Performance Evaluation Visualisation

Using common evaluation metrics such as mean Average Precision (mAP), F1 Score, Average Precision (AP) for small and weak objects, and training stability, Table 15 presents a summary of the performance of the three object detection models: YOLOv11, Faster R-CNN, and RetinaNet.

With the highest F1 Score (0.81), mAP@0.5 (0.859), and mAP@[0.5:0.95] (0.784), YOLOv11 performs better than the other two. It maintains high training stability and accuracy and demonstrates superior consistency in detecting cars, trees, and forests, among other object classes. For medium objects, YOLOv11 also achieves strong average recall (AR = 0.80) and leads in small object detection (AP = 0.64). These findings confirm that

YOLOv11 is a good choice for quick and precise UAV-based deployments in a variety of visual environments.

RetinaNet performs very well, particularly in the detection of structured objects like trees and bungalows. Though with comparatively lesser values of mAP, it is still in line with detection across object sizes, and its F1 score (0.76) is comparable to YOLOv11 Although Faster R-CNN performs well on major categories, such as cars and trees, the global accuracy and training convergence of the model are not as good as the other two models. It tends to fail in detecting thin or difficult-to-characterize objects, such as "wires" and "ultra-thin" features.

To summarize, every model has trade-offs of its own. Because it combines speed, accuracy, and stability, YOLOv11 is the most well-rounded choice and is, therefore, a perfect fit for real-time UAV applications. While Faster R-CNN can still be a good option with more frequent object overlap and dense scenes, RetinaNet provides a well-balanced alternative with dependable performance and visual clarity.

Table 15

Model Comparisons

Metric	**YOLOv11**	**Faster R-CNN**	**RetinaNet**	**RT-DETR**
mAP@0.5	0.859	0.72	0.76	0.69
mAP@[0.5:0.95]	0.784	0.66	0.67	0.61
F1 Score	0.81	0.75	0.76	0.7
Best Classes (AP)	forest, trees, car	car, trees, forest	car, trees, bungalow	Trees, fence, bungalow

Weak Classes (AP)	ultra-thin, wires	wires, ultra-thin	ultra-thin, wires	Ultra-thin, poles
AP (small objects)	0.64	0.63	0.64	0.54
AR (medium objects)	0.80	0.80	0.80	0.76
Training Stability	High	Moderate	Moderate	Moderate
Visual Accuracy	High	High	High	Medium High

Conclusion

Summary

This project built a functional system that helps drones with real-time obstacle detection and safe navigation. It incorporated various state-of-the-art deep learning models, such as Detectron2, RT-DETR v3, YOLOv11, and Faster R-CNN, which were all selected based on their comparative advantages in various detection scenarios. Precision, recall, F1-score, and mean average precision were used to evaluate the models' performance in training and testing on aerial datasets.

The system utilizes a complete pipeline, including preprocessing and image collection as well as training, testing, and deployment of the model. YOLOv11 was most notable since it was extremely fast, thus ideal for real-time detection. Detectron2 and Faster R-CNN handled small or overlapping objects better. RT-DETR v3 performed excellently in complex and dynamic environments.

REST APIs and Google Cloud were utilized in deployment, and Streamlit was utilized to create an intuitive user interface that allowed users to interact with the system and observe real-time results. Overall, the project demonstrates how drones can become more efficient and responsive to their surroundings by combining various models smartly and effectively. It serves as a foundation for future systems that require precise, rapid, and responsive obstacle detection.

Benefits and Shortcoming

For real-time obstacle detection in drone navigation, this project has a robust and adaptive solution. One of the benefits is that it combines multiple deep learning models, each of which excels at something. RT-DETR v3 and YOLOv11 provide rapid detection, which is

necessary for real-time use. Detectron2 and Faster R-CNN provide more accuracy, particularly for scenes with overlapping or small obstacles.

Another strong aspect is the system's design. From data preprocessing and collection to model training, evaluation, and deployment, it has a complete pipeline. It allows scalable and interactive usage with interfaces like REST APIs, Google Cloud, and a Streamlit interface. Future maintenance and adding new models are also facilitated by the modular design.

The system has some limitations despite its strengths. Some models, especially Faster R-CNN, use a lot of computational resources, and hence, they might not be ideal for lightweight drones. For instance, Faster R-CNN requires more memory and processing time. Additionally, speed and accuracy must be balanced as models with higher precision tend to be slower, which can be a problem for high-speed flight. Lastly, although the models performed well on test datasets, factors such as lighting, weather, and camera quality can cause changes to the performance of the models.

Potential System and Model Applications

The system developed in this project is highly suitable for use in various real-life situations. Drones equipped with this detection system can hover over regions affected by floods, forest fires, or collapsing structures and assist emergency personnel in gauging the damage and finding survivors without endangering human lives.

The system can identify structural damage on high-rise buildings, transmission lines, and bridges during infrastructure inspection. Such minute details like cracks, cables, or small protrusions are easily identified with models like Detectron2 and Faster R-CNN. Because of this, the system is useful in day-to-day maintenance in utility and civil engineering projects.

In autonomous delivery and transportation, the system can assist drones in navigating complex urban spaces without colliding with obstacles like trees, wires, and buildings. RT-DETR v3 and YOLOv11 facilitate the real-time detection required for such applications. Along similar lines, security and surveillance drones can benefit from the ability to detect moving and stationary objects in various environments, such as people or vehicles around restricted areas.

In addition, the system provides for environmental observation and research. Overall, the system is adaptive enough to be utilized in a variety of missions that require intelligent and secure drone flight in controlled and uncontrolled environments.

Experience and Lessons Learned

From pre-processing and gathering data to model deployment and user interface integration, this project provided insightful information on the whole development process of an AI-enabled system. Every phase presented different challenges, particularly when handling large sets of images and fine-tuning deep neural network models to achieve the appropriate balance between performance and accuracy.

The need to choose models on the basis of the task-specific requirements was one of the key learnings. While models like Detectron2 and Faster R-CNN provided greater precision in dense or complex environments, fast models like YOLOv11 worked well for real-time detection. Developing an efficient and functional system meant compromising on these.

The advantages of separate task segregation and modular construction were another extremely valuable learning. System alignment was enhanced, and more efficient progress was made through segmenting the work into model training, backend unification, and

interface design. From the process, they realized how crucial coordination and planning are in generating scalable AI solutions.

Recommendations for Future Work

By enhancing model architectures or adding more training data, future research can aim at increasing detection accuracy on small, overlapping, or low-contrast objects. To enhance performance in diverse environments, multi-scale fusion or model ensembling strategies can be explored.

On small drones, model inference can be optimized for edge deployment on lightweight drone. Quantization, pruning, or employing more efficient architectures, such as, MobileNet variants, could be required for this. The computational load would be alleviated without sacrificing the detection quality with these changes.

Additional enhancements can involve the incorporation of real-time sensor data, for example, LiDAR or thermal imaging, to facilitate the detection of obstacles during night-time or low-light conditions. Expanding the system to support continuous learning from new flight data could also help adapt the models over time for long-term deployments.

Contributions and Impacts on Society

By enabling drones to navigate through difficult terrain without risking human life, this project contributes to public safety. Real-time obstacle-detecting drones can assist in emergency response by providing supplies, evaluating damage, and finding survivors in otherwise dangerous or inaccessible locations.

Additionally, the system increases environmental surveillance and infrastructure monitoring, which are immensely beneficial to the economy and society. The system lowers

operational costs and the unpredictability of maintenance in essential sectors by automating bridge, power line, and distant landscape patrols.

From an educational and research perspective, the project shows what can be achieved in real-world systems with state-of-the-art AI models. It gives a basis for future advancements in autonomous robots, enabling researchers, engineers, and students to create solutions, that are socially beneficial, scalable, and flexible in diverse global settings.

References

A. A. Darwish and A. Nakhmani, "Drone Navigation and Target Interception Using Deep Reinforcement Learning: A Cascade Reward Approach," in *IEEE Journal of Indoor and Seamless Positioning and Navigation*, vol. 1, pp. 130-140, 2023, https://doi.org/10.1109/JISPIN.2023.3334690

A. B. Abdallah, A. Kallel, M. Dammak and A. B. Ali, "Olive tree and shadow instance segmentation based on Detectron2," 2022 6th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), Sfax, Tunisia, 2022, pp. 1-5, https://doi.org/10.1109/ATSIP55956.2022.9805897

A. N and U. S V, "Drone Object Detection Using Deep Learning Algorithms," *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, 2021, pp. 1187-1192, https://doi.org/10.1109/ICIRCA51532.2021.9544983

A. Rahim, A. Singh, A. Sharma, R. Garg, C. Ravisankar and N. Yamsani, "Exploring the concept of multi-modality using grounding dino with evaluation of its performance in the context of different types and complexity of input text prompts," 2023 International Conference on Smart Devices (ICSD), Dehradun, India, 2024, pp. 1-5, https://doi.org/10.1109/ICSD60021.2024.10751486

Ackermann, Marc & İren, Deniz & Wesselmecking, Sebastian & Shetty, Deekshith & Krupp, Ulrich. (2022). Automated segmentation of martensite-austenite islands in bainitic steel. Materials Characterization. 191. 112091. https://doi.org/10.1016/j.matchar.2022.112091

Adhikari, B., & Huttunen, H. (2021). Iterative Bounding Box Annotation for Object Detection. ArXiv. https://doi.org/10.1109/icpr48806.2021.9412956

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. ECCV. https://doi.org/10.1007/978-3-030-01234-2_49

Dasari, D., & Varma, P. S. (2022). "Employing Various Data Cleaning Techniques to Achieve Better Data Quality using Python," 2022 6th International Conference on Electronics, Communication and Aerospace Technology, Coimbatore, India, 2022, pp. 1379-1383, https://doi.org/10.1109/ICECA55336.2022.10009079

Ding, B., Zhang, Y., & Ma, S. (2024). A lightweight Real-Time infrared object detection model based on YOLOV8 for unmanned aerial vehicles. *Drones*, *8*(9), 479. https://doi.org/10.3390/drones8090479

E. L. Thin Jun, M. -L. Tham and B. -H. Kwan, "A Comparative Analysis of RT-DETR and YOLOv8 for Urban Zone Aerial Object Detection," 2024 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), Shah Alam, Malaysia, 2024, pp. 340-345, https://doi.org/10.1109/I2CACIS61270.2024.10649836

Fang, W., Zhang, G., Zheng, Y., & Chen, Y. (2023). Multi-task learning for UAV aerial object detection in foggy weather conditions. *Remote Sensing*, *15*(18), 4617. https://doi.org/10.3390/rs15184617

Fernandes, R., Pessoa, A., Salgado, M., Paiva, A. D., Ishak Pacal, & Cunha, A. (2024). Enhancing Image Annotation with Object Tracking and Image Retrieval: A Systematic Review. IEEE Access. https://doi.org/10.1109/access.2024.3406018

Fu, C.-Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C. (2017). DSSD: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659. https://doi.org/10.48550/arXiv.1701.06659

G. Capi, N. Sugiyama and S. -i. Kaneko, "Application of Deep Learning for Drone Obstacle Avoidance and Goal Directed Navigation," *2021 18th International Conference on Ubiquitous Robots (UR)*, Gangneung, Korea (South), 2021, pp. 453-457, https://doi.org/10.110Gao 9/UR52253.2021.9494663

G. Solaimalai, K. J. Prakash, S. K. S, A. Bhagyalakshmi, P. Siddharthan and K. R. Senthil Kumar, "Deep Reinforcement Learning for Autonomous Drone Navigation in Cluttered Environments," *2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, Chennai, India, 2024, pp. 1-5, https://doi.org/10.1109/ACCAI61061.2024.10602151

G. -T. Tu and J. -G. Juang, "Path Planning and Obstacle Avoidance Based on Reinforcement Learning for UAV Application," *2021 International Conference on System Science and Engineering (ICSSE)*, Ho Chi Minh City, Vietnam, 2021, pp. 352-355, https://doi.org/10.1109/ICSSE52999.2021.9537945

Gupta, S., & Verma, R. (2017). Exploratory Data Analysis. In Data Science Fundamentals. Springer. https://doi.org/10.1007/978-3-319-43742-2_15

Hinton, J., & Brown, S. (2019). Automating Exploratory Data Analysis via Machine Learning: An Overview. ACM Digital Library. https://doi.org/10.1145/3318464.3383126

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. IEEE CVPR. https://doi.org/10.1109/CVPR.2017.351

Huang, M., Mi, W., & Wang, Y. (2024). EDGS-YOLOV8: An improved YOLOV8 lightweight UAV detection model. *Drones*, *8*(7), 337. https://doi.org/10.3390/drones8070337

Isaac-Medina, B. K. S., Poyser, M., Organisciak, D., Willcocks, C. G., Breckon, T. P., & Shum, H. P. H. (2021). Unmanned Aerial Vehicle Visual Detection and Tracking using Deep Neural Networks: A Performance Benchmark. https://doi.org/10.48550/arXiv.2103.13933

J. Guo, C. Huang and H. Huang, "A Deep Q-Network-Based Algorithm for Obstacle Avoidance and Target Tracking for Drones," *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Honolulu, Oahu, HI, USA, 2023, pp. 4530-4535, https://doi.org/10.1109/SMC53992.2023.10393953

Jiang, W., & Lu, X. (2019). Large-Scale Datasets for Going Deeper in Image Understanding. IEEE International Conference on Multimedia & Expo. https://doi.org/10.1109/ICME.2019.00256

Katkuri, A. V. R., Madan, H., Khatri, N., Abdul-Qawy, A. S. H., & Patnaik, K. S. (2024). Autonomous UAV Navigation using Deep Learning-Based Computer Vision Frameworks: A Systematic Literature Review. *Array*, 100361. https://doi.org/10.1016/j.array.2024.100361

Khanam, R., & Hussain, M. (2024). YOLOv11 for vehicle detection: Advancements, performance, and applications. arXiv preprint arXiv:2410.22898. https://doi.org/10.48550/arXiv.2410.22898

Khanam, R., & Hussain, M. (2024). YOLOv11: An overview of the key architectural enhancements. arXiv preprint arXiv:2410.17725. https://doi.org/10.48550/arXiv.2410.17725

Kolbeinsson, B., & Mikolajczyk, K. (2023). DDOS: The Drone Depth and Obstacle Segmentation Dataset. Huggingface Discussions. https://doi.org/10.48550/arXiv.2312.12494

Kucukayan, G., & Karacan, H. (2024). YOLO-IHD: Improved Real-Time Human Detection System for indoor Drones. *Sensors*, *24*(3), 922. https://doi.org/10.3390/s24030922

L. J. Sayed, P. K. Souripalli and S. S. Chiddarwar, "Autonomous Navigation of Drones Using Explainable Deep Reinforcement Learning in Foggy Environment," *2024 IEEE Students Conference on Engineering and Systems (SCES)*, Prayagraj, India, 2024, pp. 1-6, https://doi.org/10.1109/SCES61914.2024.10652534

Li, Y., Ma, C., Li, L., Wang, R., Liu, Z., & Sun, Z. (2024b). Lightweight tunnel obstacle detection based on improved YOLOV5. *Sensors*, *24*(2), 395. https://doi.org/10.3390/s24020395

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. IEEE CVPR. https://doi.org/10.1109/CVPR.2017.106

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, & Zhang, L. (2023). Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. https://doi.org/10.48550/arxiv.2303.05499

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. Proceedings of the European Conference on Computer Vision (ECCV). https://doi.org/10.1007/978-3-319-46448-0_2

Lv, W., Xu, S., Zhao, Y., Wang, G., Wei, J., Cui, C., Du, Y., Dang, Q., & Liu, Y. (2023). DETRs Beat YOLOs on Real-time Object Detection. https://doi.org/10.48550/arxiv.2304.08069

M. R. Rezaee, N. A. W. A. Hamid, M. Hussin and Z. A. Zukarnain, "Comprehensive Review of Drones Collision Avoidance Schemes: Challenges and Open Issues," in *IEEE*

Transactions on Intelligent Transportation Systems, vol. 25, no. 7, pp. 6397-6426, July 2024, https://doi.org/10.1109/TITS.2024.3375893

M. T. Hayat Suhendar and Y. Widyani, "Machine Learning Application Development Guidelines Using CRISP-DM and Scrum Concept," 2023 IEEE International Conference on Data and Software Engineering (ICoDSE), Toba, Indonesia, 2023, pp. 168-173, https://doi.org/10.1109/ICoDSE59534.2023.10291438

M. Wadhwa, T. Choudhury, G. Raj and J. C. Patni, "Comparison of YOLOv8 and Detectron2 on Crowd Counting techniques," 2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS), Istanbul, Turkiye, 2023, pp. 1-6, https://doi.org/10.1109/ISAS60782.2023.10391466

Marques, T., Carreira, S., Miragaia, R., Ramos, J., & Pereira, A. (2024). Applying deep learning to real-time UAV-based forest monitoring: Leveraging multi-sensor imagery for improved results. *Expert Systems With Applications*, *245.* https://doi.org/10.1016/j.eswa.2023.123107

N and U. S. V, "Custom Based Obstacle Detection Using Yolo v3 for Low Flying Drones," *2021 International Conference on Circuits, Controls and Communications (CCUBE)*, Bangalore, India, 2021, pp. 1-6, https://doi.org/10.1109/CCUBE53681.2021.9702739

N. Bdair, W. Alzyadat and S. AlZubi, "Intelligent Model for Optimizing Gantt Chart in the Planning Stage," 2023 International Conference on Information Technology (ICIT), Amman, Jordan, 2023, pp. 556-560, https://doi.org/10.1109/ICIT58056.2023.10225762

O. Ardiç and G. Çetinel, "Exploring the Newly Introduced Data Augmentation Technique in TensorFlow Library," 2024 32nd Signal Processing and Communications Applications

Conference (SIU), Mersin, Turkiye, 2024, pp. 1-4, https://doi.org/10.1109/SIU61531.2024.10601037

P. R. Gervi, A. Harati and S. K. Ghiasi-Shirazi, "Vision-Based Obstacle Avoidance in Drone Navigation using Deep Reinforcement Learning," *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, Mashhad, Iran, Islamic Republic of, 2021, pp. 363-368, https://doi.org/10.1109/ICCKE54056.2021.9721451

Pande, B., Padamwar, K., Bhattacharya, S., Roshan, S., & Bhamare, M. (2022). A Review of Image Annotation Tools for Object Detection. IEEE Xplore. https://doi.org/10.1109/ICAAIC53929.2022.9792665

Patel, A., & Rao, V. (2024). Tool for image annotation in the context of modern object detection. IEEE Zinc Conference. https://doi.org/10.1109/ZINC61849.2024.10579415

Prakljačić, S., Grbić, R., Vranješ, M., & Herceg, M. (2024). Tool for image annotation in context of modern object detection. IEEE Zinc. https://doi.org/10.1109/zinc61849.2024.10579415

Puri, A., Agrawal, G., Dukare, A., & Jawale, M. (2023). "A Survey and Analysis of Textual Content Based on Exploratory Data Analysis Technique and Opinion Analysis," 2023 4th International Conference on Computation, Automation and Knowledge Management (ICCAKM), Dubai, United Arab Emirates, 2023, pp. 1-6, https://doi.org/10.1109/ICCAKM58659.2023.10449608

R. C. V, S. Jaganmohan, K. H, V. Thomas and S. Pandi S, "Intelligent Auto Annotation System for Identifying Vehicles without Number Plates Using Grounding DINO and Segment Anything Model," 2024 Second International Conference on Advances in Information Technology (ICAIT), Chikkamagaluru, Karnataka, India, 2024, pp. 1-5, https://doi.org/10.1109/ICAIT61638.2024.10690781

R. Singh, S. Shetty, G. Patil and P. J. Bide, "Helmet Detection Using Detectron2 and EfficientDet," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2021, pp. 1-5, https://doi.org/10.1109/ICCCNT51525.2021.9579953

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. NeurIPS. https://doi.org/10.5555/2969239.2969250

Ren, T., Jiang, Q., Liu, S., Zeng, Z., Liu, W., Gao, H., Huang, H., Ma, Z., Jiang, X., Chen, Y., Xiong, Y., Zhang, H., Li, F., Tang, P., Yu, K., & Zhang, L. (2024). Grounding DINO 1.5: Advance the "Edge" of Open-Set Object Detection. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2405.10300

S. A. Hassan, T. Rahim and S. Y. Shin, "Real-time UAV Detection based on Deep Learning Network," *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2019, pp. 630-632, doi: 10.1109/ICTC46691.2019.8939564. https://doi.org/10.1109/ICTC46691.2019.8939564

S. S. Nandan Challapalli, G. Mishra, Y. Pachauri, A. Mishra, S. R. Kumar and L. Kumar, "Comparing TensorFlow.js and TensorFlow in Python: An Accessibility and Usage Analysis," 2023 6th International Conference on Contemporary Computing and Informatics (IC3I), Gautam Buddha Nagar, India, 2023, pp. 250-254, https://doi.org/10.1109/IC3I59117.2023.10397702

S. van der Walt, S. C. Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation,"Shan Miyuan, She Xiaohua and Ren Bin, "A data mining approach to forming general work breakdown structure," 2011 IEEE 2nd

International Conference on Software Engineering and Service Science, Beijing, China, 2011, pp. 330-334, https://doi.org/10.1109/ICSESS.2011.5982230

Safaa Dafrallah, Moulay Akhloufi, "Malicious UAV detection using various modalities", Drone Systems and Applications 12, pg. 1, (2024); https://doi.org/10.1139/dsa-2023-0049

Sharma, P., & Jain, M. (2020). Annotation for Object Detection. In S. Kumar & V. Mishra (Eds.), Advances in Computer Vision. Springer. https://doi.org/10.1007/978-981-33-4543-0_34

Suzuki, H., & Yamamoto, K. (2018). A study on object detection method from manga images using CNN. IEEE International Workshop on Advanced Image Technology (IWAIT). https://doi.org/10.1109/IWAIT.2018.8369633

Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. IEEE CVPR. https://doi.org/10.1109/CVPR42600.2020.01080

Thangaraj, P., & Srivatsan, K. (2014). An efficient dynamic indexing and metadata-based storage in cloud environment. Proceedings of ICRTIT. https://doi.org/10.1109/ICRTIT.2014.6996151

V. Rakesh, P. Chilukuri, P. Vaishnavi, P. Sreekaran, P. Sujala and D. R. Krishna Yadav, "Real Time Object Recognition Using OpenCV and Numpy in Python," 2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA), Uttarakhand, India, 2023, pp. 421-426, https://doi.org/10.1109/ICIDCA56705.2023.10099584

Vallabhaneni, K., & Narayanan, R. (2008). Using Object Metadata to Detect and Tolerate Attacks in Object Storage Devices. Proceedings of IEEE GLOBECOM. https://doi.org/10.1109/GLOCOM.2008.ECP.606

W. Wang, C. Wang, Y. Sun and X. Yu, "Research on Automatic Obstacle Avoidance and Target Tracking System for Drones Based on Deep Learning," *2024 Second International Conference on Data Science and Information System (ICDSIS)*, Hassan, India, 2024, pp. 1-5, https://doi.org/10.1109/ICDSIS61070.2024.10594308

Wan, D., Lu, R., Xu, T., Shen, S., Lang, X., & Ren, Z. (2023). Random Interpolation Resize: A free image data augmentation method for object detection in industry. Expert Systems with Applications. https://doi.org/10.1016/j.eswa.2023.120355

Wang, C., Yeh, I., & Liao, H. M. (2024, February 21). *YOLOV9: Learning what you want to learn using programmable gradient information*. arXiv.org. https://arxiv.org/abs/2402.13616

Wang, F., Zhu, X., Zhou, Z., & Tang, Y. (2023). Deep-reinforcement-learning-based UAV autonomous navigation and collision avoidance in unknown environments. *Chinese Journal of Aeronautics*, *37*(3), 237–257. https://doi.org/10.1016/j.cja.2023.09.033

Wang, S., Xia, C., Lv, F., & Shi, Y. (2024). RT-DETRv3: Real-time End-to-End Object Detection with Hierarchical Dense Positive Supervision. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2409.08475

Wilson, D., & Gray, M. (2022). Live Detection of Foreign Object Debris on Runways Detection using Drones and AI. IEEE Aerospace Conference. https://doi.org/10.1109/AERO53065.2022.9843697

X. Zhenming, "Research on simplification of PERT network chart," 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet), Xianning, China, 2011, pp. 3458-3461, https://doi.org/10.1109/CECNET.2011.5768222

Xie, E., Ding, C., Wang, W., Zhan, X., Xu, H., & Luo, P. (2020). DetNAS: Backbone search for object detection. NeurIPS. https://doi.org/10.5555/3454287.3454295

Xu, M., Yoon, S., Fuentes, A., & Park, D. S. (2023). A Comprehensive Survey of Image Augmentation Techniques for Deep Learning. Pattern Recognition. https://doi.org/10.1016/j.patcog.2023.109347

Y. -Y. Hsieh, W. -Y. Lin, D. -L. Li and J. -H. Chuang, "Deep Learning-Based Obstacle Detection and Depth Estimation," *2019 IEEE International Conference on Image Processing (ICIP)*, Taipei, Taiwan, 2019, pp. 1635-1639, https://doi.org/10.1109/ICIP.2019.8803100

Yaseen, M. (2024, September 12). *What is YOLOv9: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector*. arXiv.org. https://arxiv.org/abs/2409.07813

Yilmaz, B., & Kutbay, U. (2024). YOLOV8-Based Drone Detection: Performance Analysis and Optimization. *Computers*, *13*(9), 234. https://doi.org/10.3390/computers13090234

Z. Deng, H. Sun, S. Zhou, J. Zhao and H. Zou, "Toward Fast and Accurate Vehicle Detection in Aerial Images Using Coupled Region-Based Convolutional Neural Networks," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 8, pp. 3652-3664, Aug. 2017, doi: 10.1109/JSTARS.2017.2694890. https://doi.org/10.1109/JSTARS.2017.2694890

Z. Yu, "RT-DETR-iRMB: A Lightweight Real-Time Small Object Detection Method," 2024 IEEE 6th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Chongqing, China, 2024, pp. 1454-1458, https://doi.org/10.1109/IMCEC59810.2024.10575494

Zhang, S., Wen, L., Bian, X., Lei, Z., & Li, S. Z. (2018). Single-shot refinement neural network for object detection. Proceedings of the IEEE CVPR. https://doi.org/10.1109/CVPR.2018.00441

Zhao, Z., Wan, Y., & Chen, Y. (2024). Deep Reinforcement Learning-Driven collaborative Rounding-Up for multiple unmanned aerial vehicles in obstacle environments. *Drones*, *8*(9), 464. https://doi.org/10.3390/drones8090464

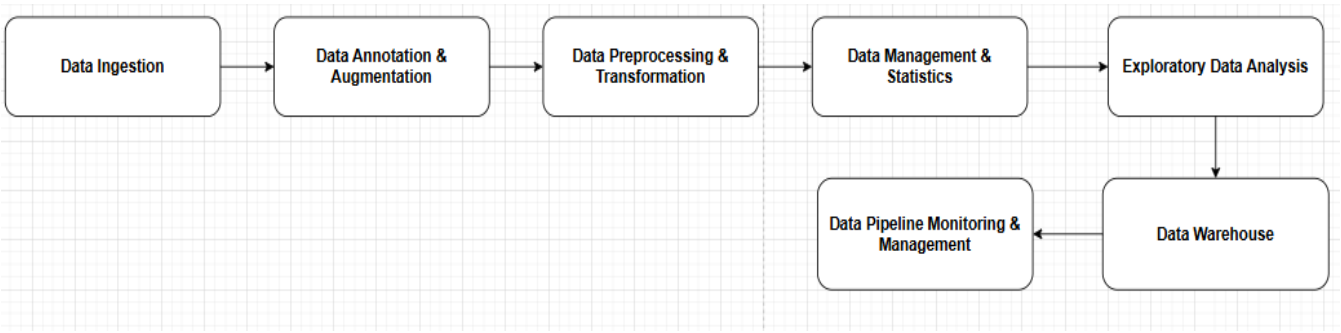
Zhao, Z., Zheng, P., Xu, S., & Wu, X. (2018). Object Detection with Deep Learning: A Review. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1807.05511

Data Pipeline Demo

This project created a well-defined and efficient data pipeline, as shown in Figure 120, to process and prepare drone footage and related information for use in model training and obstacle identification. The procedures in this section are linked and should work to change raw data into a usable format without compromising the data's integrity or accessibility. The whole workflow was structured in such a manner that real-time drone operations could be maintained, with each stage closely supervised and monitored.

Figure 120

Data Pipeline for the project



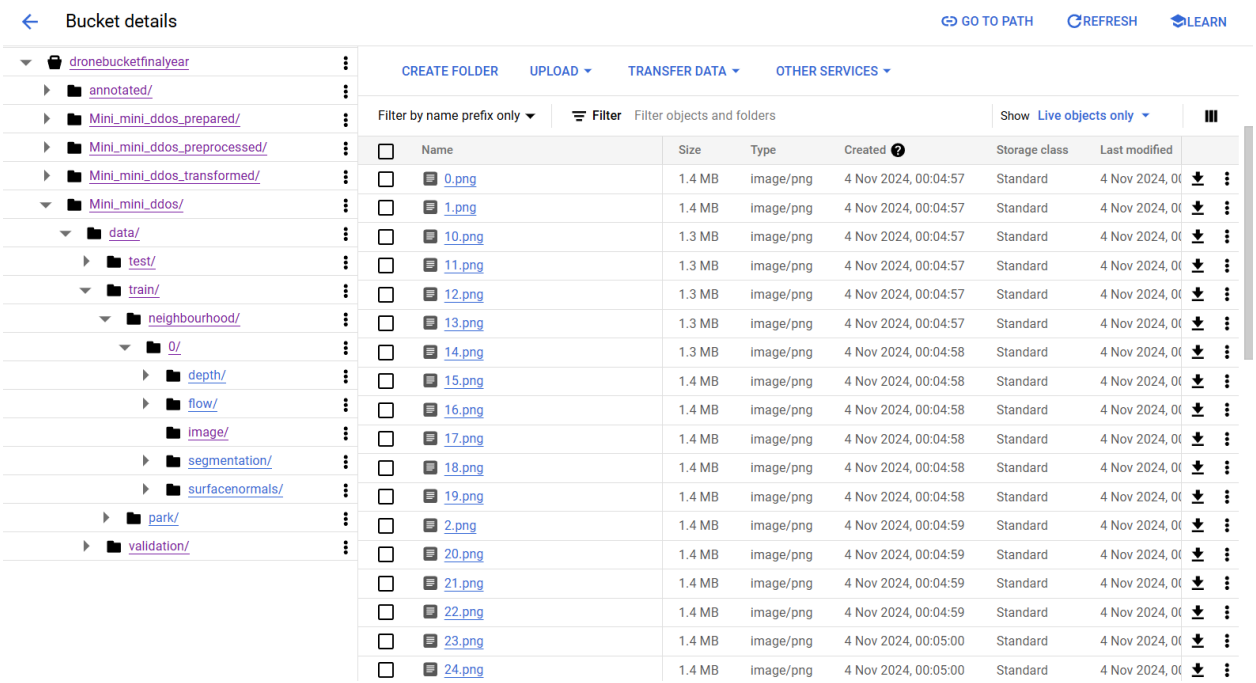
Note. The above image shows the data pipeline for the project.

Data Ingestion

Data ingestion was, therefore, carried out by collecting raw images and metadata of flight sensors. Data captured across different scenarios, such as urban landscapes, forests, and open fields, were prepared and stored in Google Cloud Storage as seen in Figure 121 and Figure 122. Images were stored in high-resolution pictures to avoid the loss of any particular feature; at the same time, metadata files were created and uploaded to track all necessary information about the flights, including timestamps and weather conditions, as seen in Figure 123 and Figure 124. A structured and centralized storage methodology was followed, wherein access to such information would become easy at later stages of data processing.

Figure 121

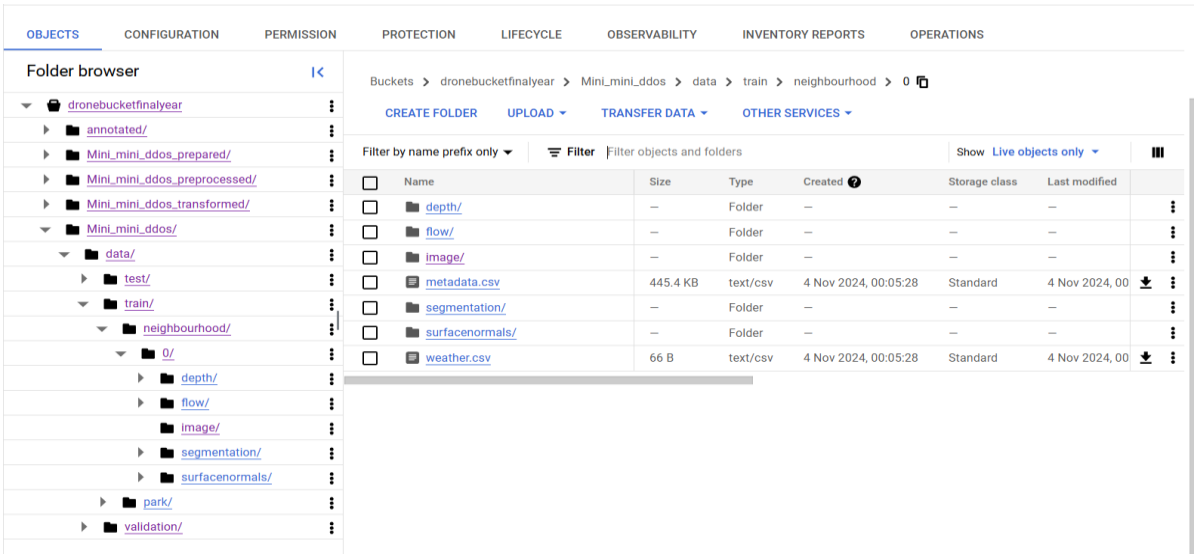
DDOS Data stored in Google cloud storage



Note. The above image shows the data loaded in Google Cloud Storage.

Figure 122

The Metadata file and weather data file for each flight are stored in Google Cloud Storage



Note. The above image shows the data loaded in Google Cloud Storage.

Figure 123

Contents of the metadata file

PREVIEW

	123 drone_orie	123 drone_orie	123 drone_orie	123 camera_p	123 camera_p	123 camera_p	123 linear_vel	123 linear_vel	123 linear_vel	123 linear_acc	123 linear_acc	123 linear_acc	123 angular_v	123 angular_v	123 angular_
8	-0.2061250061	-0.00808046013	-0.8838824034	-133.4462585	153.9097748	-11.41559219	-1.140495777	-0.3200290501	-0.9031839371	-2.785474777	-1.547499895	2.651776791	-2.542375088	2.399998665	1.54978
7	-0.1759496629	0.06826238334	-0.8791401386	-133.5008087	153.8679504	-11.51187801	-1.295861959	-0.3661121428	-0.7659516931	-2.913666725	-0.1982389987	2.448573589	-1.455849648	2.438046455	1.20888
6	-0.1712698638	0.07612224668	-0.8784626722	-133.5084839	153.8635254	-11.52370834	-1.322035193	-0.3669912219	-0.7440600991	-2.900928259	0.00192009471	2.416027546	-1.257298827	2.448592424	1.15121
3	-0.02713691629	-0.10013932 73	-0.9648286104	-43.30193329	138.2822266	-17.46762085	-8.796298027	-2.963681221	0.8268330693	0.4613839388	-1.554380894	1.286223412	-0.3980216682	0.0748809427	-0.31581
8	0.02402984537	0.00060248654	0.9975328445	69.88542175	88.61496735	-10.12945843	-7.532250404	1.036805034	-0.0505541563	-0.0966770649	-0.03020442836	0.04881191254	-0.00804047100	0.01453807112	0.01499
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.01248
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.01248
7	0.02378719114	0.00048291322	0.9975534678	69.63677216	88.64897156	-10.13132	-7.535080433	1.035927057	-0.04911794141	-0.0920150578	-0.02834260091	0.0469493866	-0.00766887608	0.01384531613	0.01248
8	0.02326312661	0.00022538355	0.9975891113	69.04869843	88.72938538	-10.13547897	-7.541815758	1.033891559	-0.04563389719	-0.0809134841	-0.02395156398	0.04246520996	-0.00684405490	0.01219328027	0.00592
1	0.02313286811	0.00016182300	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.00424
1	0.02313286811	0.00016182300	0.997595489	68.89028931	88.75102997	-10.13654232	-7.543485641	1.03339994	-0.04475391284	-0.07815942168	-0.02287727967	0.04134178162	-0.00665154913	0.01178166457	0.00424
4	0.02307836711	0.00013522407	0.9975979328	68.8223877	88.76030731	-10.13699245	-7.544184208	1.033196092	-0.04438394308	-0.07700735331	-0.02242858522	0.04087543488	-0.00656224275	0.0116086863	0.00354
8	0.[illegible]75165915	-0.00002436281	0.997607708	68.39221191	88.81912994	-10.13974762	-7.548792362	1.031877518	-0.04190891236	-0.06940442324	-0.01949266903	0.03774547577	-0.00600289739	0.01047458872	-0.0009
4	0.[illegible]4474163	-0.00012548558	0.9976100326	68.09777832	88.85942078	-10.1415472	-7.551022053	1.031257629	-0.04068887979	-0.06572329998	-0.01808215864	0.03620529175	-0.00573356496	0.00992615334	-0.0029884
8	0.02249869332	-0.00014771253	0.9976100922	68.0298233	88.86872101	-10.14195251	-7.551608562	1.031096578	-0.04036485776	-0.06475353241	-0.01771507785	0.03580093384	-0.00562459603	0.00978063885	-0.0034925

Note. The above image shows the contents of the metadata file.

Figure 124

Contents of weather file

123 rain	123 roadwetne	123 snow	123 roadsnow	123 mapleleaf	123 dust	123 fog
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Note. The above image shows the contents of the weather file.

Data Augmentation

For every one of these images, there was a 50% chance of horizontal flipping-a trick used to introduce mirrored points of view. Angular diversities were varied with the datasets in random rotation within ±45°. Light changes were simulated by changes to brightness and contrast: bright variabilities of ±30%, and contrasting randomness of ±0.5, ensured illumination diversity comparable to that in the real world within this data sample. Next,

motion blur was simulated using Gaussian blur with a kernel size of (3, 3) and variance from 0.1 to 0.5 to simulate real-world flying scenarios of drones. Lastly, sensor inaccuracies and environmental disruptions were simulated by adding Gaussian noise with a standard deviation in the range of 0.01 to 0.05. Taken together, all these augmentation techniques not only diversified the dataset but ensured a standard resolution at 224x224 pixels to enable good generalization of the model over varying conditions, as seen in Figure 125, which shows the image after performing all the augmentations on the image (Wan, D. et al. 2023). Figure 126 shows that the augmented image is stored back in GCS.

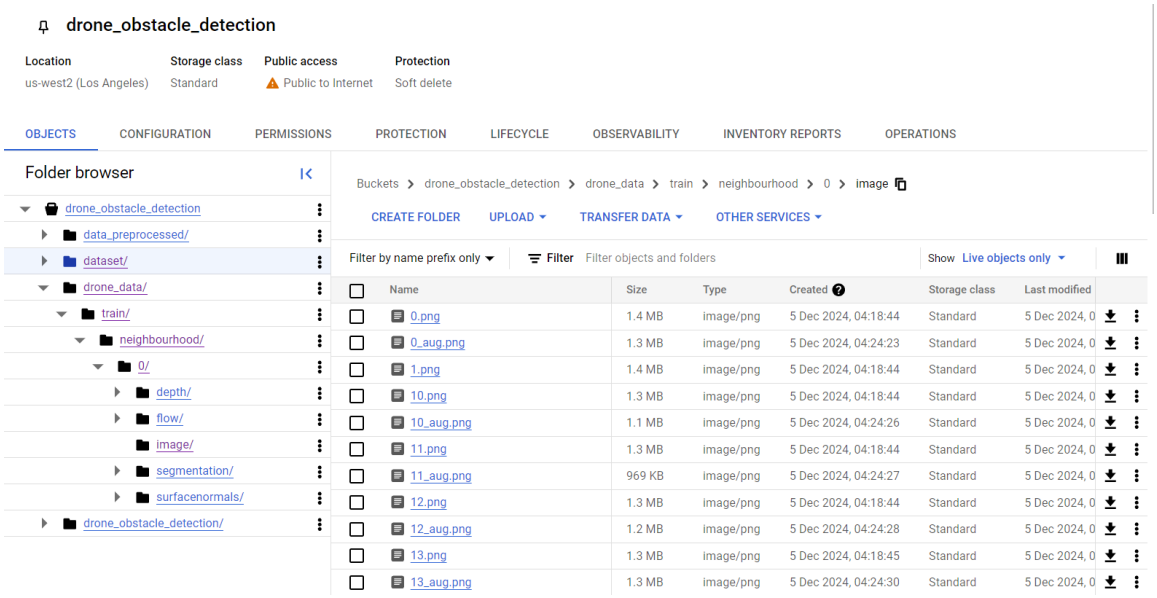
Figure 125

Data Augmentation

Note. The above image shows the image after doing data augmentations.

Figure 126

Augmentated Image stored in Google Cloud Storage



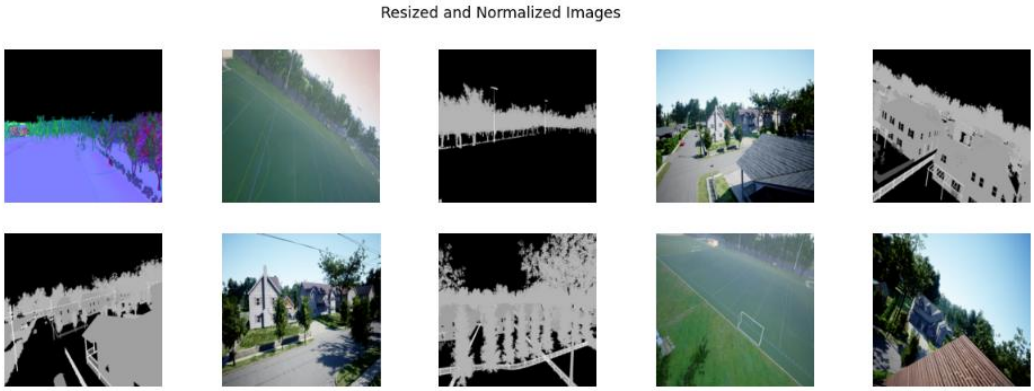
Note. The above image shows the storage of annotated images in Google Cloud Storage.

Data Preprocessing and Transformation

This stage of preprocessing and transformation was quite critical to cleanse and standardize the data. Preprocessing includes resizing all images to a fixed dimension, normalizing the pixel values as shown in Figure 127, and making color adjustments with enhancements in contrast to improve image quality as shown in Figure 128. Normalization of depth maps is performed, as shown in Figure 129, and refinement in segmentation masks helps to draw objects clearly, as shown in Figure 130. Figure 131 shows the images obtained after performing histogram equalization. Finally, the images were standardized as shown in Figure 132. Preprocessing metadata included combining metadata and weather files into one consolidated CSV, handling duplicates, cleaning missing values, and performing timestamp object conversions for time-based analysis, as shown in Figure 133. Each of these steps in preprocessing was followed by the storage of cleaned and transformed data back into GCS for an organized representation, as represented by Figure 134, the comprehensive dataset for analysis.

Figure 127

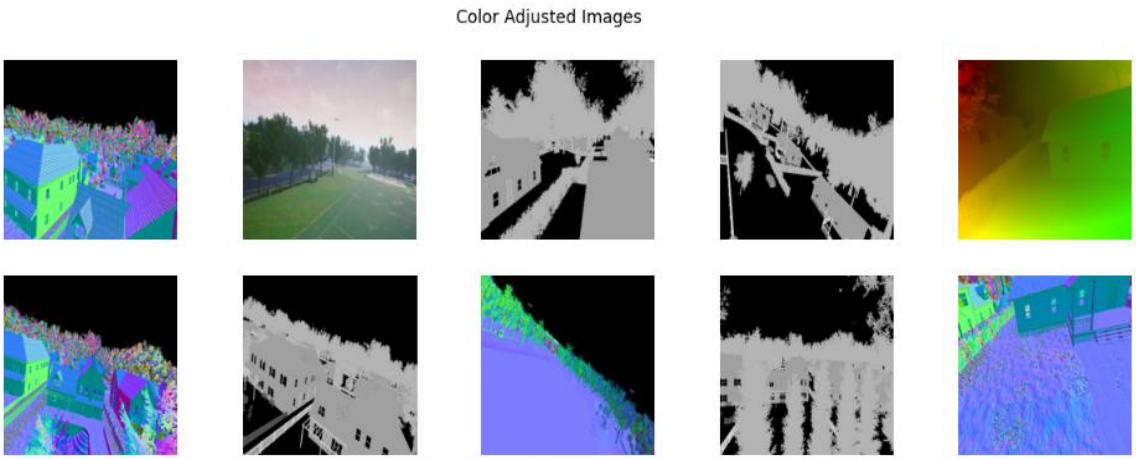
Images after resizing and normalizing images



Note. The above image shows the resized and normalized images.

Figure 128

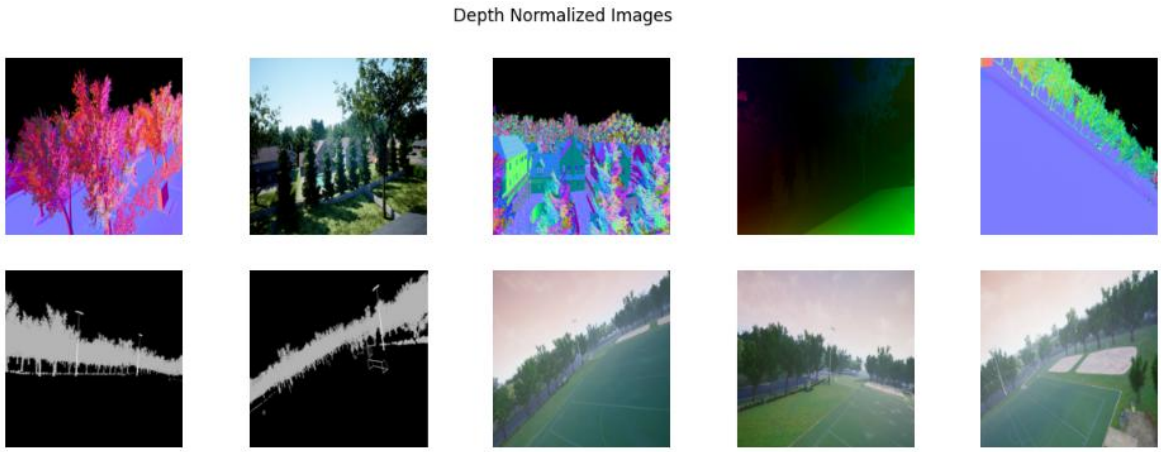
Images after color adjustment



Note. The above image shows the color adjusted images.

Figure 129

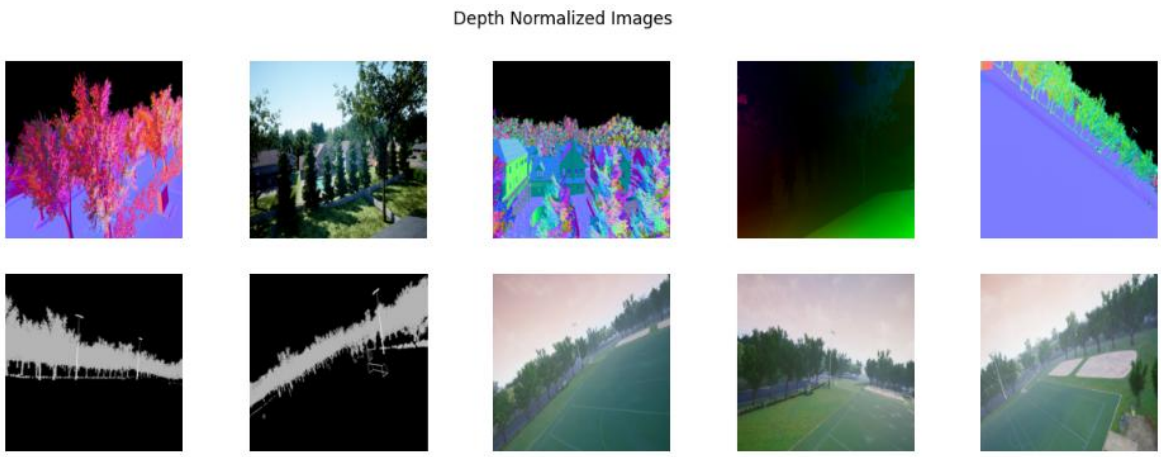
Images after performing depth normalisation



Note. The above image shows the depth normalized images.

Figure 130

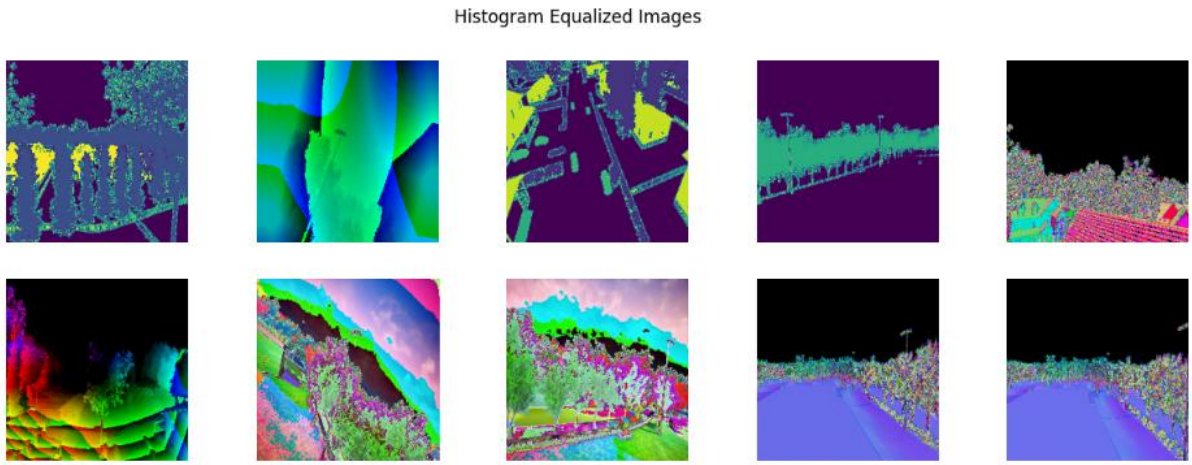
Images after segmentation masks



Note. The above image shows the segmentation mask processed images.

Figure 131

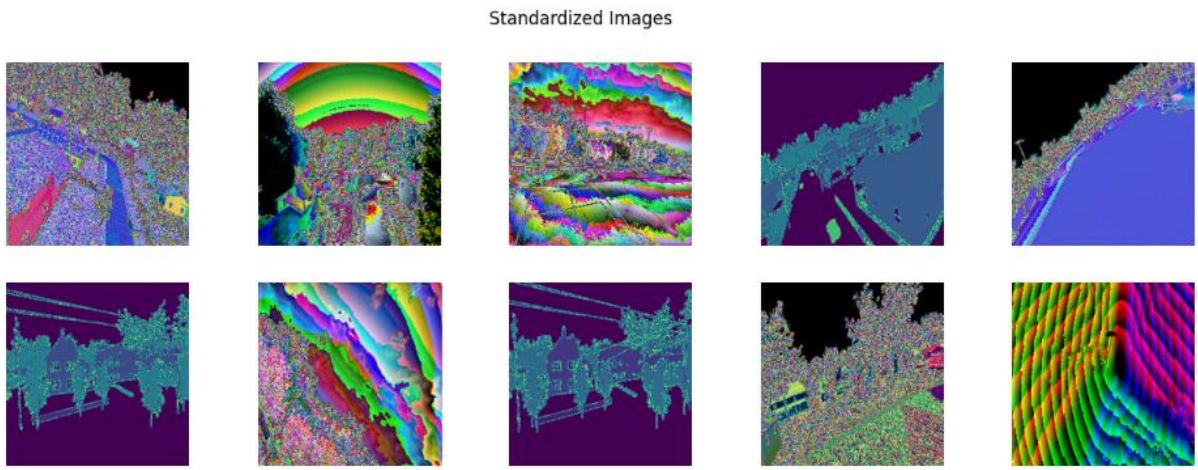
Images after performing the histogram equalisation



Note. The above image shows the histogram equalised images.

Figure 132

Images after performing Standardisation



Note. The above image shows the standardised images.

Figure 133

Master metadata file

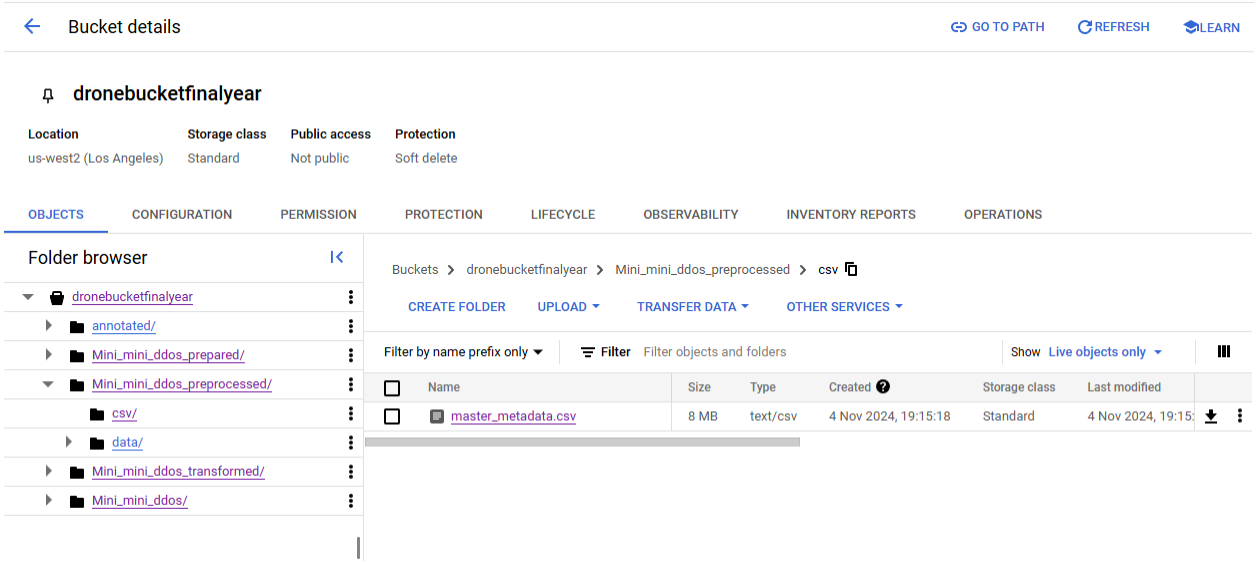
PREVIEW

flight_id	dataset	location	timestamp	image_file	image_pat	depth_pat	segmentat	flow_path	surface_n	weather_c	timestamp	img_index	drone_pos	drone_pos
0	train	neighbourhood	1.70828E+18	12.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	13.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	14.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	15.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	16.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	17.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	18.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	19.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	2.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	20.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	21.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	22.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	23.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	24.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895
0	train	neighbourhood	1.70828E+18	25.png	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_	gs://data_298a_dataset/DATA 29		1.70828E+18	0	42.30175018	126.73895

Note. The above image shows the master metadata file.

Figure 134

Master metadata file stored in GCS



Note. The above image shows the folder in which master metadata file is stored in GCS.

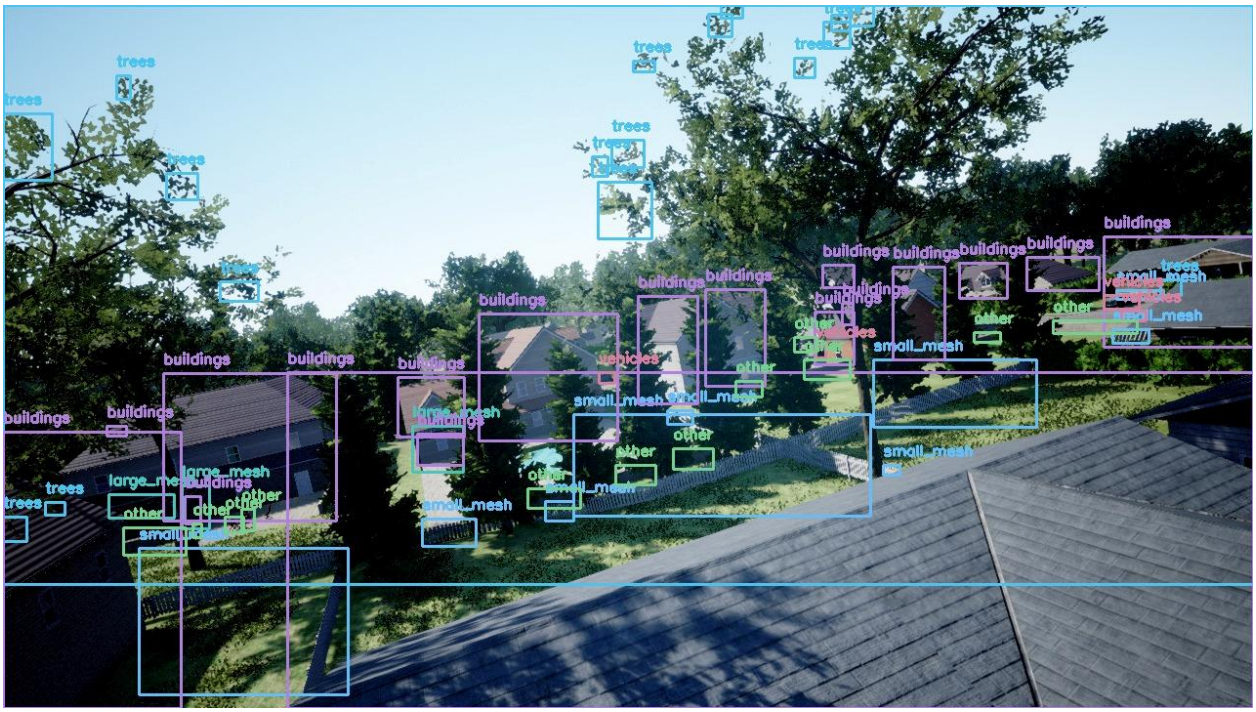
Data Annotations

For each flight folder, the content downloads images and their comparing division covers from GCS into a brief neighborhood catalog on the Google Process Motor occasion.

Using OpenCV, the content then iterates through each partition veil to recognize forms for every obstacle class. For each unique form over a specific area threshold -say 50 pixels to avoid noise-, we decide that the bouncing box facilitates and attracts a square shape on the first image of the doled-out variety. At this stage, each bouncing box with its class name focuses on the visual interpretation of the groups. The moment an annotated picture has been created as shown in Figure 135 it gets saved for a brief instance, and then uploaded to GCS.

Figure 135

Image after performing annotation using OpenCV



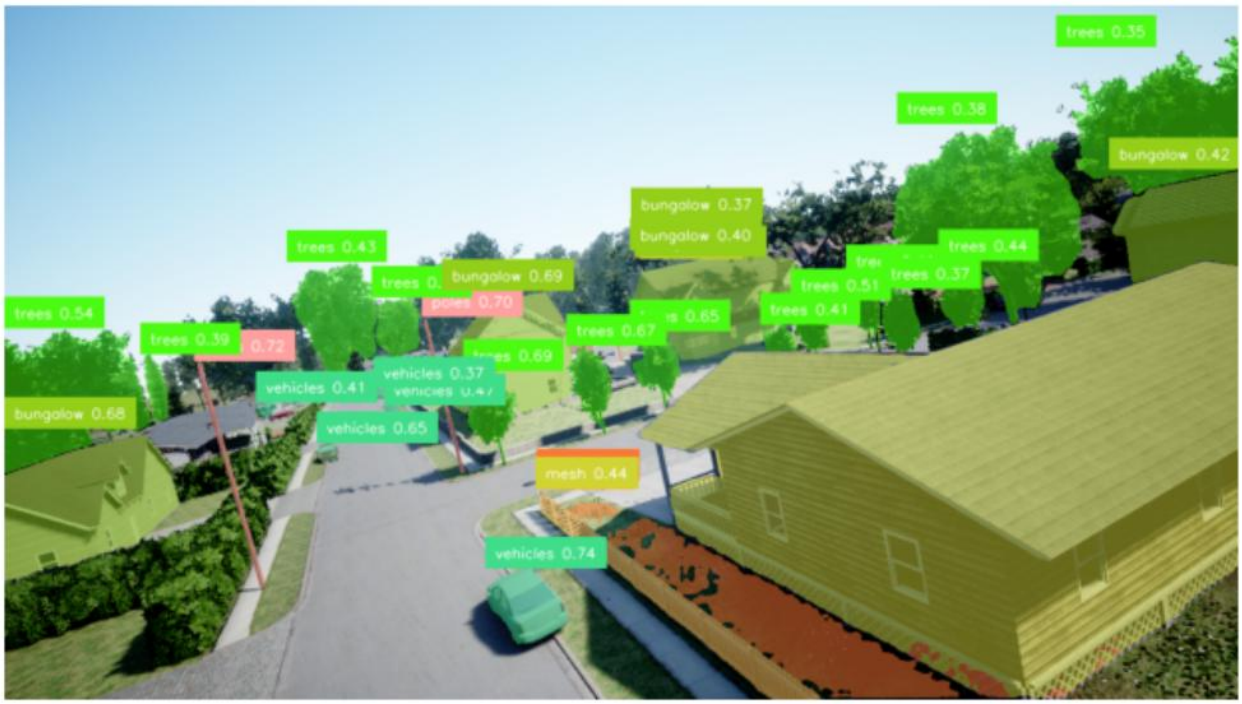
Note. The above image shows a sample image after performing the annotations using OpenCV.

This project's annotation was done using the state-of-the-art auto-model for object detection and segmentation, known as the Grounded Segment Anything Model. It uses ontology-based classification, hence allowing for quick identification and labeling of objects in images with consistent quality. This greatly reduces manual effort in the annotation

process, yet it is as precise and accurate on a diverse dataset. These provided the necessary semantic representation mapping from classes of objects, including ultra-thin, fences, vehicles, and trees, so annotation could be fast and more straightforward. For the detection of such objects from images, Grounded SAM would identify them through a lookout for their semantic counterparts while segregating them based on their set categories as seen in Figure 136.

Figure 136

Image after performing annotation using GroundedSam



Note. The above image shows a sample after performing the annotations using GroundedSam.

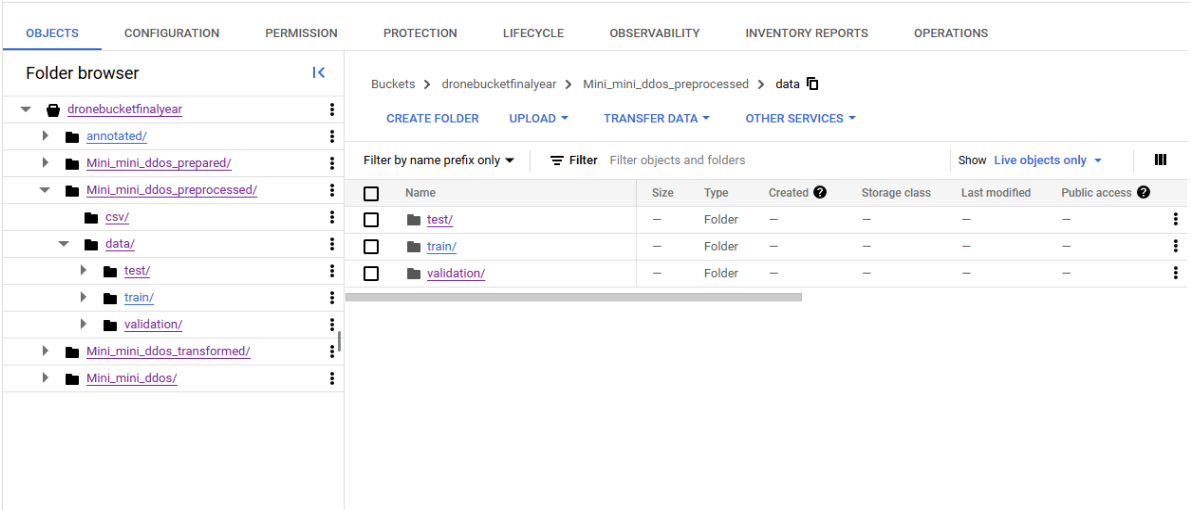
Data Preparation and Splitting

Following preprocessing, the dataset was organized for model training. The data was separated into training, validation, and test sets in a balanced manner based on classes. Figure

137 shows how the data obtained after preprocessing, transformation, and annotation was divided into 80% training, 10% validation, and 10% testing. Furthermore, photos and information were organized into folders corresponding to each of these sets to facilitate loading and model training. This allowed for strong model development without sacrificing data integrity during splits.

Figure 137

Splitting data into train, test, and validation test



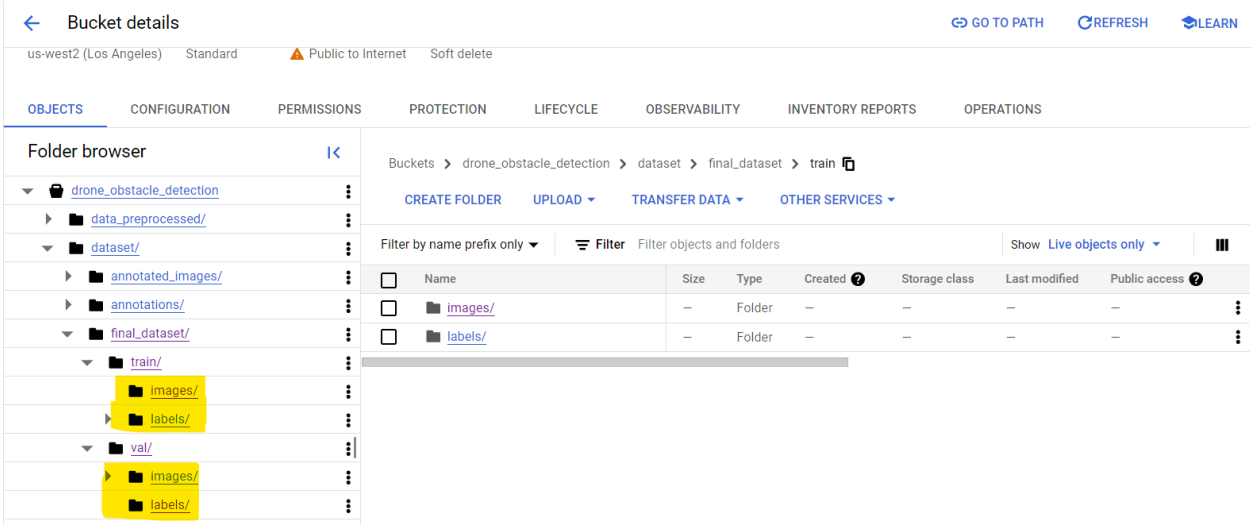
Note. The above image shows the folder where train, test, and validation split are stored in GCS.

Apart from annotation, a dataset was prepared in a format that was suitable to be used by the model for training. It was be divided into three splits which are training, validation, and testing. Each split would comprise images and corresponding annotation files, each in separate folders on GCS as seen in Figure 138. In such a case, the data is well organized and hence directly accessible by the model without any pre-processing or preparation at the time of training. The YAML configuration file defining the structure of the dataset had to be created thereafter: the path to training, validation, and test data; class ID mapping to labels as

seen in Figure 139. It was also uploaded on GCS for easy integration into the model training pipeline. Figure 140 shows all the annotations are stored in a single folder after annotations whereas Figure 141 shows all the images stored in the folder after performing all the annotations. This pipeline for annotation and preparation of the dataset ensured that the data was comprehensively labeled, well-organized, and fully compatible with the modeling and training framework.

Figure 138

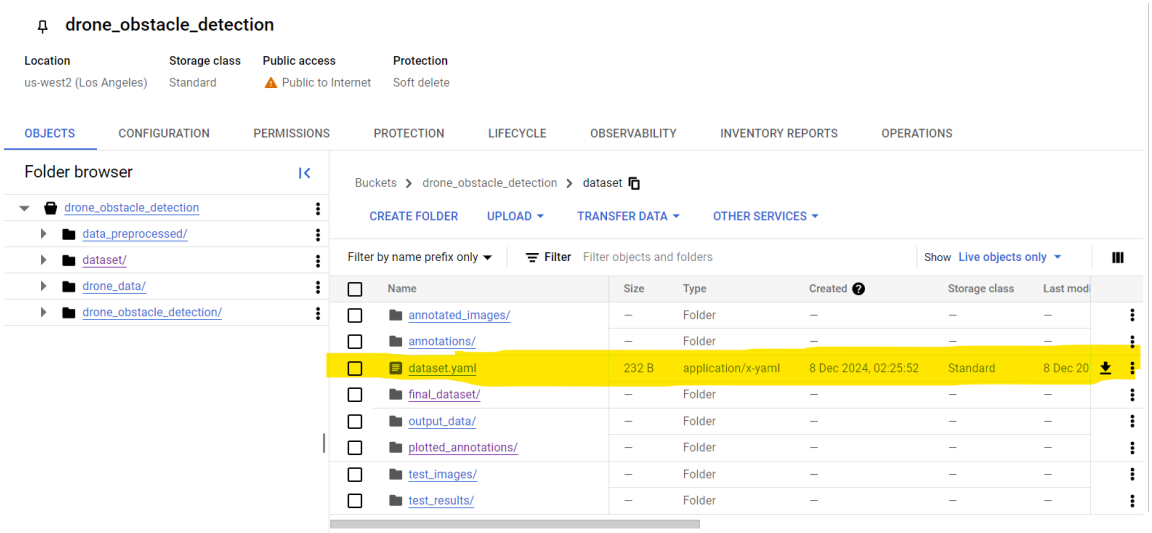
Images and annotations stored in a structure suitable for modeling in GCS



Note. The above image shows the GCS folder in which images and annotations are stored in GCS.

Figure 139

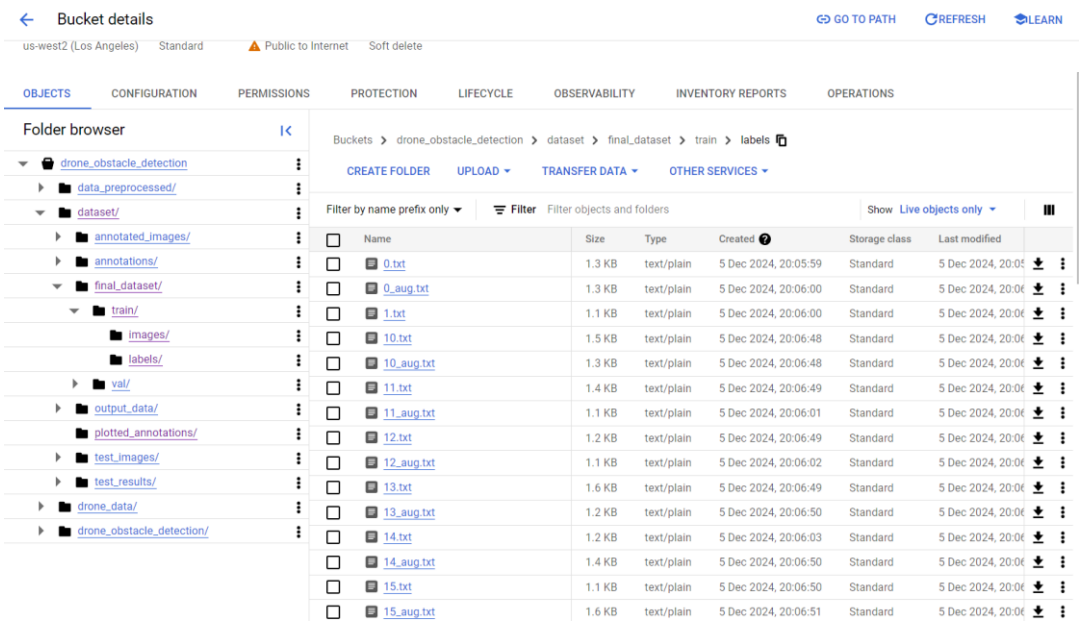
Yaml file created and stored on GCS



Note. The above image shows the Yaml file created and stored in GCS.

Figure 140

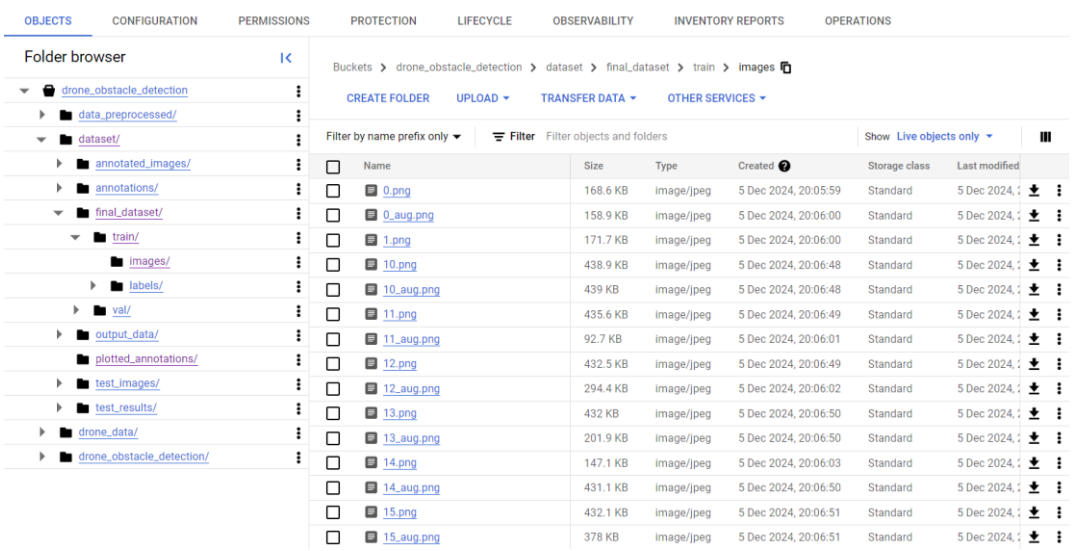
Annotations are created and stored on GCS in compatible folders



Note. The above image shows the annotations created and stored in GCS.

Figure 141

Images created and stored on GCS after annotations in compatible folders



Note. The above image shows the images created and stored in GCS after annotations.

Exploratory Data Analysis (EDA)

EDA was performed to understand the data that would be used and to develop informed models. We used Jupyter Notebook and Google Colab for visualization of class distributions, checking for any class imbalance, then analyzed different feature correlations. The EDA consisted of statistical summaries and visualizations that could precisely bring out outliers or trends in affecting model performance. This would, therefore, lay a very strong basis for model configuration optimization and the improvement of detection accuracy. Thereafter, results from EDA were documented and stored on GCS for future references (Gupta, S., & Verma, R., 2017).

Data Warehouse and Management

Considering the volume of preprocessed data, we used Google BigQuery for data warehousing. This would efficiently store and query large volumes of data without having

any jagged edges in integrating image metadata and transformations. The scalability inherent to BigQuery provided efficiency during large volumes of queries without loss of performance, hence making the data processing efficient and reliable. Advanced analytics supported by data warehoused data allowed the evaluation of a model and retraining if needed.

Data Pipeline Monitoring and Management

This data pipeline will constantly be at a latter part of monitoring and maintaining to ensure a reliable system. Monitoring of the flow of data, the time it takes to process that data, and its performance is provided by Google Cloud monitoring tools. Automated alerts would be set up in cases of failures or slowdowns to thus enable rapid interventions. Performance evaluations shall be done routinely to ensure the effectiveness and scalability of the pipeline, sustaining an infrastructure robust enough to bear increased volume and complexity.

This structured approach ensured that every stage of processing the data, from ingestion to analysis, was seamless and efficient to guarantee the success of the project. We stored each processed and transformed dataset back in GCS to ensure we have a cohesive and well-managed data environment supportive of real-time needs created by the drone-based obstacle detection system.

Appendices

Appendix A - System Testing

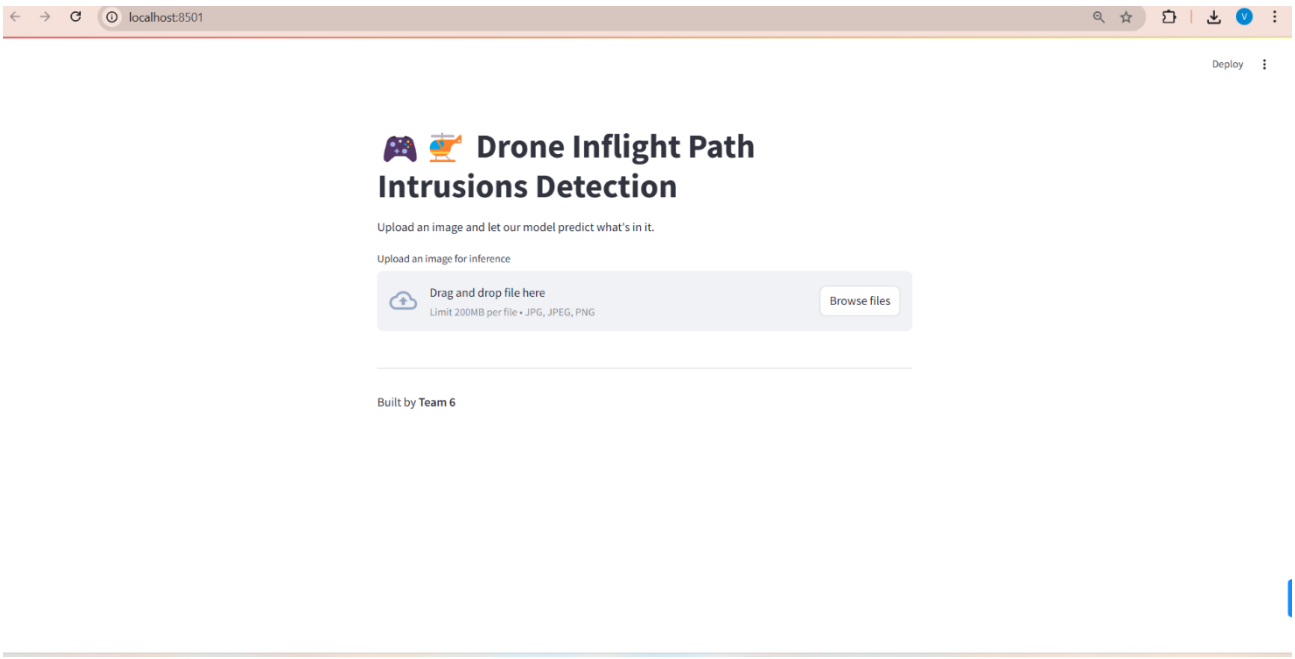
In this project, Streamlit serves as a convenient framework for creating a web-based user interface that handles user interactions and displays results. The workflow is as follo

Use Case

Landing Screen. The user is presented with a screen for uploading custom images or a folder containing all the images. Figure 142 shows the Home Page of UI.

Figure 142

Home Page of UI

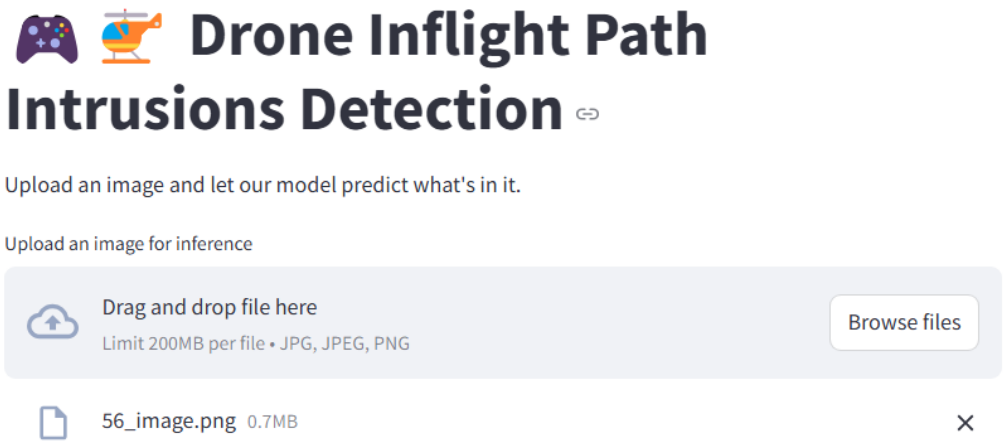


Note. The above image shows the home page of the UI.

Upload or select. The user can opt to upload an image or a folder containing many images, or they can select and upload images directly. The user clicks the “Upload” button to select an image. Streamlit temporarily stores these uploaded files in memory. Figure 143 shows the upload and select options for input, and it can be seen that a file was uploaded and stored in the memory.

Figure 143

Showing Upload and Select Options

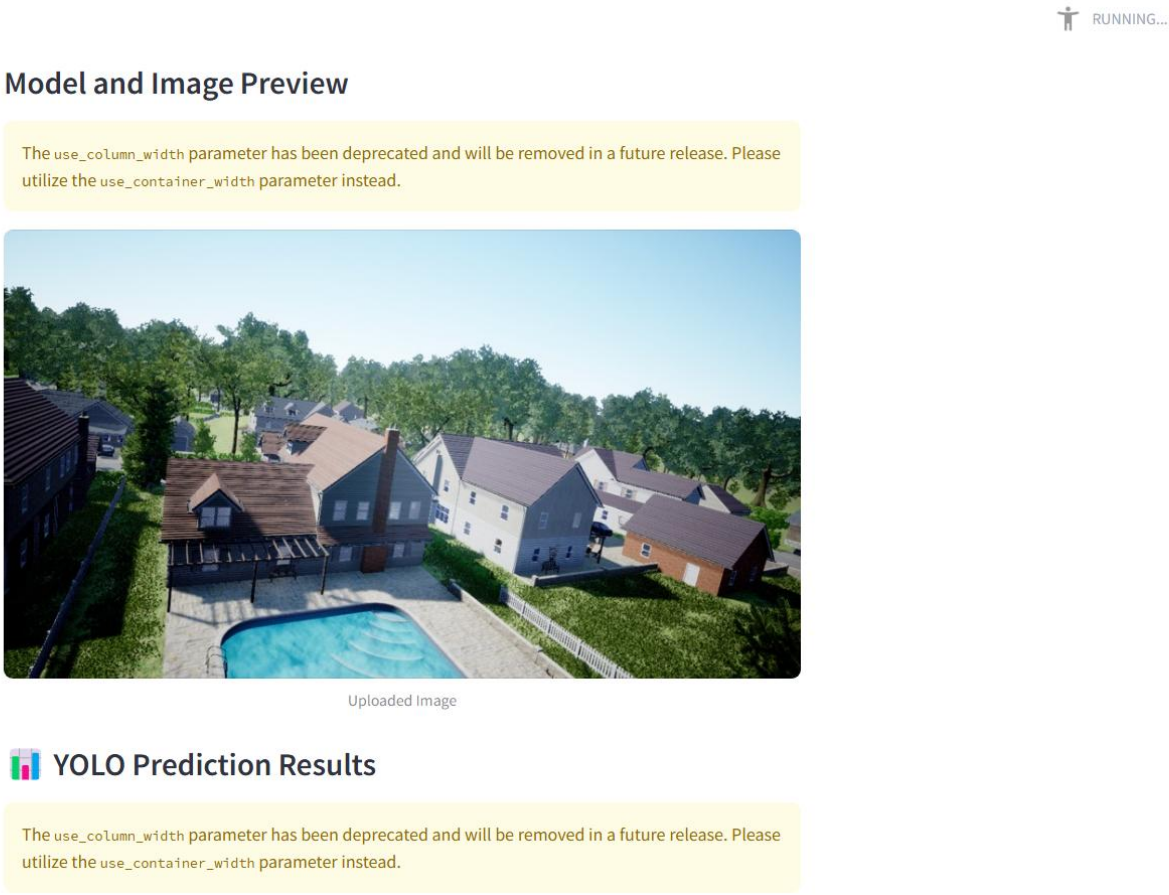


Note. The above image shows the uploading of files in the UI.

Saving Temporary Files. To facilitate processing by the model, the app writes the in-memory file to a temporary file on the system. This allows the model, which expects an actual file path, to access the image directly from disk. The uploaded image then can be previewed easily so that we can see the original image as seen in Figure 144.

Figure 144

Original Image Preview



Note. The above image shows the original image preview.

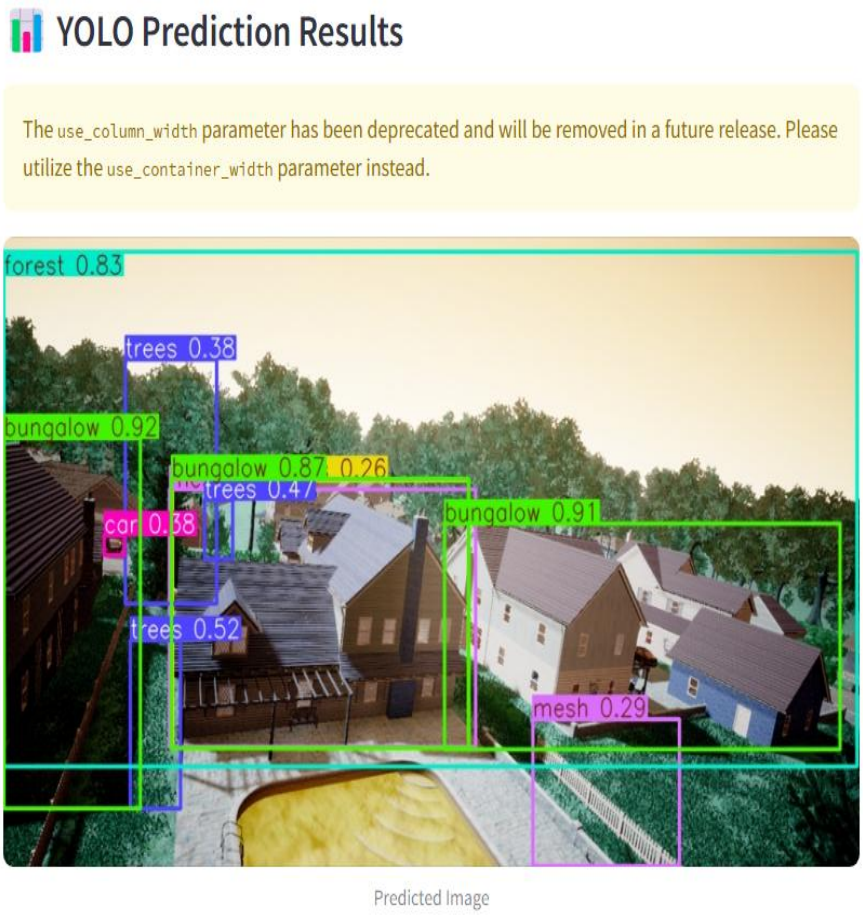
Loading and Running YOLO. A pre-trained best-performing model, which is the YOLO model (in a .pt file) is loaded from disk. YOLO then performs object detection on the uploaded image, generating bounding boxes and confidence scores for any objects it identifies.

Displaying Results and GPT Summary. Streamlit displays the annotated images showing all detected objects with bounding boxes in the same browser interface where the user uploaded the file.

Simultaneously, the detection info (object names and confidence scores) is passed to OpenAI's GPT model, which returns a concise textual summary. Streamlit then displays this summary in real time, giving users a quick interpretation of the model's performance and findings. Figure 145 depicts the prediction results on the images, and Figure 146 shows the textual summary about the images' predictions and the model's performance and findings.

Figure 145

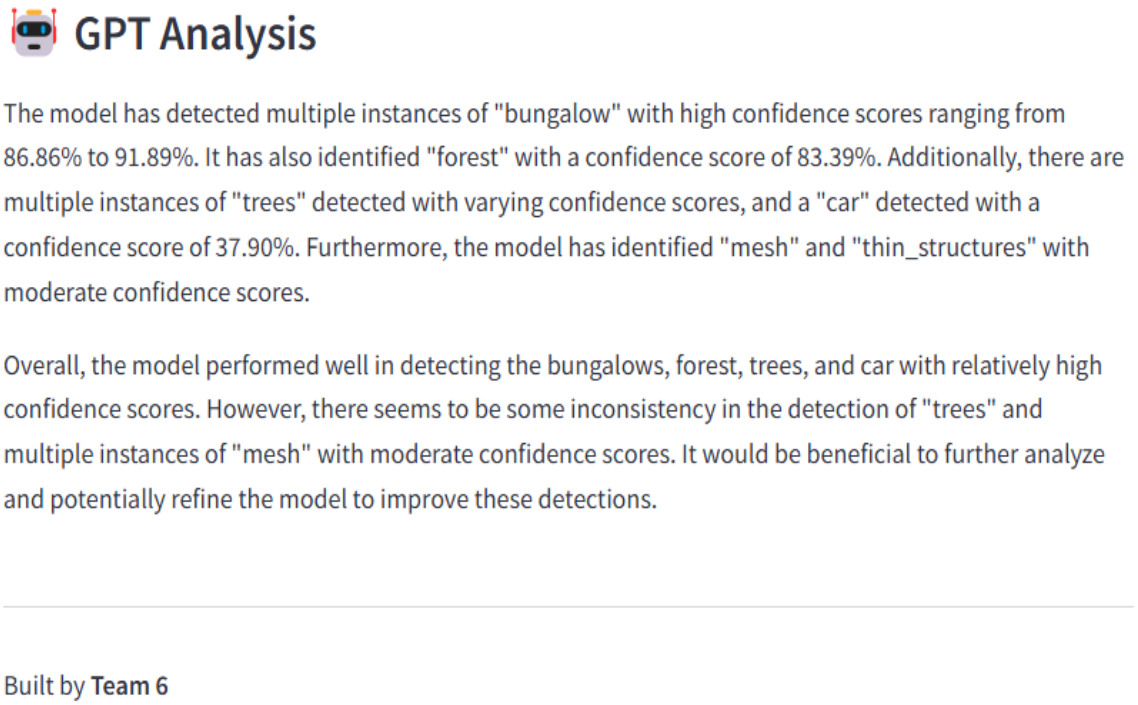
YOLOv11 Predictions on Images



Note. The above image shows the YOLOv11 Predictions on Images.

Figure 146

Textual summary of the images' predictions and the model's performance and findings



Note. The above image shows the textual summary of the images' predictions and the model's performance and findings.

Temporary File Cleanup. Once the workflow is complete, the temporary file is removed to maintain a clean environment.

Appendix B - Project Data Source and Management Source

Datasets (Raw and Processed Data)

1. Raw Data:

   https://drive.google.com/drive/folders/1FjsU8AHgylrpFM_iPya5Lu9zZXR3U1MT

2. Processed Data:

   https://drive.google.com/drive/folders/1BXFNOpDCRR2whh9Pgi4uRCigijqf-244

Models

1. YOLOv11:

   https://drive.google.com/drive/folders/1_YV74OsoUi93qEsYSkOTGjKkkmUFs810

   Hugging Face Deployed model: https://huggingface.co/vsham001/Yolo298B

2. Faster R-CNN using Detectron2:

   https://drive.google.com/drive/folders/1gMUJPv6DnbTsA0rCobMkn3HN-jcmGul0

3. RetinaNet using Detectron2:

   https://drive.google.com/drive/folders/1Ppc89g1z2KYLoPsvVJ7Oh6JRw_Js6gbR

4. RT-DETR:

   https://drive.google.com/drive/folders/1YqU6qnS1BWtrAn79CzbdkRypDH68B7zo

Appendix C - Project source code, Presentation and Demonstration

Source code and fine-tuning:

https://drive.google.com/drive/folders/1jgq4JxLP12WtUOSLhoURBadDDpLhFnEg

PowerPoint presentation: https://docs.google.com/presentation/d/1qCc8486Fcdjf0tNF-12PksAYBf9VskLt_COih12Tn_I/edit?slide=id.g35723523c1b_2_54#slide=id.g35723523c1b_2_54

Demonstration:

https://drive.google.com/drive/folders/1rwyVBFTPKGX5QZw92N-3-cIOAMJZKjdP