UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MONIELLY GOMES MONTEIRO

UAV-based Building Facade Cleaning Using Autonomous Windows Detection

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Edison Pignaton de Freitas

Porto Alegre
February 2026

CIP — CATALOGING-IN-PUBLICATION

Monteiro, Monielly Gomes

UAV-based Building Facade Cleaning Using Autonomous Windows Detection / Monielly Gomes Monteiro. – Porto Alegre: PPGC da UFRGS, 2026.

108 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2026. Advisor: Edison Pignaton de Freitas.

1. Building Facades. 2. Unmanned Aerial Vehicle. 3. Artificial Intelligence. I. Freitas, Edison Pignaton de. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof[a]. Marcia Barbosa
Vice-Reitor: Prof. Pedro Costa
Pró-Reitora Adjunta de Pós-Graduação: Prof[a]. Claudia Wasserman
Diretor do Instituto de Informática: Prof. Luciano Paschoal Gaspary
Coordenador do PPGC: Prof. Gabriel Luca Nazar
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

“ Balance is the foundation of the Work. If you yourself do not have a stable foundation, upon what will you stand to direct the forces of Nature? Be therefore swift and active like the Sylphs, yet avoid frivolity and extravagance; be energetic and strong like the Salamanders, but avoid irritability and ferocity; be flexible and attentive to images like the Undines, yet avoid indolence and fickleness; be diligent and patient like the Gnomes, but avoid coarseness and greed. In this way you will gradually develop the powers of your soul and become able to command the Spirits of the elements.”

— LIBER LIBRAE

ACKNOWLEDGMENTS

I would like to express, first of all, my gratitude to God, the great and powerful Yahweh, who granted me the breath of life and, through His mysteries, has sustained me until this moment. I am deeply grateful to my parents, Raimundo Monteiro de Souza and Marinalva Gomes, for all their support, guidance, love, and care. To my siblings, Gabriel Gomes Monteiro and Mariany Gomes Monteiro, I am immensely thankful for brightening my days and making my path lighter. I also thank my friends and advisors Venicio Silva Araújo and Heinstein Frederich Leal dos Santos for his constant companionship and friendship throughout these years. I extend my sincere appreciation to Professor Edison Pignaton de Freitas for his support, guidance, and dedication, and for showing me how enjoyable it can be to work with something one truly loves. Equally important, I thank Professor Manuel Menezes de Oliveira Neto for his support and the knowledge he shared, especially in the computer vision courses, which I had the pleasure of taking. And also Professor Alexandre dos Santos Roque for his valuable support and guidance during the preparation of the articles. I also thank the Skydrones team for their support, assistance, and partnership in the implementation and execution of the system presented here. To the Marcos Vizzoto for the knowledge shared in the area of drones. And also to the Scientific Initiation student, Guilherme Kohl, for the support and assistance. And finally, I express my gratitude to the community of the Rio Branco Settlement, in the rural area of the municipality of Nova Olímpia, Mato Grosso, the place where I grew up and which instilled in me values that I carry for life.

ABSTRACT

This research investigates the transformation of conventional vehicles into autonomous systems through the integration of artificial intelligence techniques. Leveraging synthetic data generated by Latent Diffusion Models (LDMs), the study addresses the scarcity of annotated data for glass facades, enabling efficient and precise object detection in contemporary architectural environments. A comparative evaluation of ten YOLO algorithm variants revealed that YOLOv12 achieved the highest mean Average Precision (mAP) of 80% for the "glass window" class on a combined real and synthetic dataset. However, YOLOv4 demonstrated practical performance advantages due to its native C implementation. Simulation tests in Gazebo and ROS2 validated the successful integration of perception, decision-making, and control modules, with the drone completing tasks autonomously and accurately identifying windows. Field experiments, both real-world and simulated, further confirmed the system's operational robustness, adaptability, and scalability across different drone platforms, highlighting its potential for real-world autonomous applications in facade inspection and cleaning.

Keywords: Building Facades. Unmanned Aerial Vehicle. Artificial Intelligence.

UAV-based Building Facade Cleaning Using Autonomous Windows Detection

RESUMO

Esta pesquisa investiga a transformação de veículos convencionais em sistemas autônomos por meio da integração de técnicas de inteligência artificial. Utilizando dados sintéticos gerados por Modelos de Difusão Latente (LDMs), o estudo aborda a escassez de dados anotados para fachadas de vidro, permitindo uma detecção de objetos eficiente e precisa em ambientes arquitetônicos contemporâneos. Uma avaliação comparativa de dez variantes do algoritmo YOLO revelou que o YOLOv12 alcançou a maior média de Precisão (mAP) de 80% para a classe "vidraça" em um conjunto de dados combinado (real + sintético). No entanto, o YOLOv4 demonstrou vantagens práticas de desempenho devido à sua implementação nativa em C. Testes de simulação no Gazebo e ROS2 validaram a integração bem-sucedida dos módulos de percepção, tomada de decisão e controle, com o drone executando tarefas de forma autônoma e identificando janelas com precisão. Experimentos em campo tanto no mundo real quanto no simulado, confirmaram ainda a robustez operacional do sistema, sua adaptabilidade e escalabilidade em diferentes plataformas de drones, destacando seu potencial para aplicações autônomas em inspeção e limpeza de fachadas no mundo real.

Palavras-chave: Fachada de Edifício, Veículo Aéreo Não Tripulado, Inteligência Artificial.

LIST OF ABBREVIATIONS AND ACRONYMS

AGPL3	GNU Affero General Public License version 3
AI	Artificial Intelligence
BFE	Boundary Feature Extraction
BiFPN	Bidirectional Feature Pyramid Network
CBS	Conv-BN-SiLU
CIoU	Complete Intersection over Union
CMP FACADE	Computational Morphology Project – Facade Dataset
CSP	Cross Stage Partial
CSPNet	Cross Stage Partial Network
CSPPAN	Cross Stage Partial Path Aggregation Network
CUDA	Compute Unified Device Architecture
CPU	Central Processing Unit
DCFF	Dual-Channel Feature Fusion
DFPN	Dense Feature Pyramid Network
DIP	Digital Image Processing
DJI	Da-Jiang Innovations
DSD MatchingNet	Dual-Scale Decoder Matching Network
E-IoU	Efficient Intersection over Union
ELAN	Efficient Layer Aggregation Network
ECP Paris	École Centrale Project Paris
FRF	Faster RCNN Facades
FPN	Feature Pyramid Network
GDNet-B	Gated Dilated Network – Base
GDD	Glass Detection Dataset

GFSD	Glass Facade Segmentation Dataset
G-ELAN	Generalized Efficient Layer Aggregation Networks
GIoU	Generalized Intersection over Union
GHz	Gigahertz
GPU	Graphics Processing Unit
IB-HSI Dataset	Indian Buildings – Hyperspectral Imaging Dataset
IOU	Intersection over Union
LCFI	Local Context Feature Integration
LED	Light Emitting Diode
LiDAR	Light Detection And Ranging
LDM	Latent Diffusion Models
LTS	Long-Term Support
LVM	Logicol Volume Manager
MLP	Multilayer Perceptron
mAP	mean Average Precision
NMS	Non Maximum Suppression
OpenCV	Open Source Computer Vision Library
OpenVINO	Open Visual Inference and Neural Network Optimization
PAN	Path Aggregation Network
PGI	Progressive Gradient Information
PNG	Portable Network Graphics
PSA	Polarized Self-Attention
PSPNet	Pyramid Scene Parsing Network
R-CNN	Region-based Convolutional Neural Network
RAM	Random Access Memory
RD	Real Data

RSD	Real and Synthetic Data
RDNet	Residual-Dense Network
ResNet	Residual Network
R-ELAN	Re-parameterized Efficient Layer Aggregation Network
RepNC-SPELAN	Re-parameterized No-Cross Split-Path Efficient Layer Aggregation
RepVGG	Re-parameterized VGG Network
RGB	Red, Green and Blue
RANSAC	Random Sample Consensus
RTSP	Real Time Streaming Protocol
RTP	Real-time Transport Protocol
SAA	Spatial Attention Attention
SIFT	Scale-Invariant Feature Transform
SPP	Spatial Pyramid Pooling
TXT	Text
UAV	Unmanned Aerial Vehicle
UFRGS	Federal University of Rio Grande do Sul
VAE	Variational Autoencoders
VGG	Visual Geometry Goup Network
WIoU	Wise Intersection over Union loss
YOLO	You Only Look Once

LIST OF FIGURES

Figure 1.1 An example of drone cleaning facade application. 15

Figure 2.1 Artificial intelligence, machine learning, and deep learning 17
Figure 2.2 Classical programming vs Machine Learning 18
Figure 2.3 Forms of data representation 18
Figure 2.4 Diagram of the Latent Diffusion Model (LDM) architecture. 19
Figure 2.5 Stable Diffusion Architecture. 21
Figure 2.6 The network architecture of YOLOv1 22
Figure 2.7 The network architecture of YOLOv2 22
Figure 2.8 The network architecture of YOLOv3 23
Figure 2.9 The network architecture of YOLOv4 23
Figure 2.10 The network architecture of YOLOv7 24
Figure 2.11 The network architecture of YOLOv8 25
Figure 2.12 The network architecture of YOLOv9 25
Figure 2.13 The network architecture of YOLOv10 26
Figure 2.14 The network architecture of YOLOv11 26
Figure 2.15 The network architecture of YOLOv12 27

Figure 4.1 Comparison between an image collected from the web and an image generated with stable diffusion. 38
Figure 4.2 Solution Overview 39
Figure 4.3 Workflow of the specialized dataset development. 41
Figure 4.4 Annotation Validation 45
Figure 4.5 Component diagram of the embedded system used. 49
Figure 4.6 Tinker board and Camera Intel Real Sense 50
Figure 4.7 Path followed by the drone 51
Figure 4.8 SkyClean schematic drawing 52
Figure 4.9 Communication between the different software components 54
Figure 4.10 Diagram Class 62

Figure 5.1 Performance of YOLO for class Glass Window 68
Figure 5.2 Precision and Recall for class Glass Window 68
Figure 5.3 Results YOLOv12 76
Figure 5.4 F1-Confidence YOLOv12 77
Figure 5.5 Precision-Confidence YOLOv12 77
Figure 5.6 Precision-Recall YOLOv12 78
Figure 5.7 Recall-Confidence YOLOv12 78
Figure 5.8 Urban environment drone simulation 79
Figure 5.9 UAV point-of-view simulation 80
Figure 5.11 UAV identifying windows 87
Figure 5.10 Detection of a window in a digital image 88
Figure 5.12 Detection identified by the UAV 89
Figure 5.13 Skyclean performing a detection mission 90
Figure 5.14 Frames Detection 91
Figure 5.15 Drone altitude relative to the ground 91
Figure 5.16 Drone temperature and its components 92
Figure 5.17 Test of the Drone Using the Pump 93

LIST OF TABLES

Table 4.1 SkyClean Dimensions 53

Table 5.1 YOLOv2 RD 64
Table 5.2 YOLOv2 RSD 64
Table 5.3 YOLOv3 RD 64
Table 5.4 YOLOv3 RSD 64
Table 5.5 YOLOv4 RD 64
Table 5.6 YOLOv4 RSD 64
Table 5.7 YOLOv5 RD 65
Table 5.8 YOLOv5 RSD 65
Table 5.9 YOLOv7 RD 65
Table 5.10 YOLOv7 RSD 65
Table 5.11 YOLOv8 RD 65
Table 5.12 YOLOv8 RSD 65
Table 5.13 YOLOv9 RD 66
Table 5.14 YOLOv9 RSD 66
Table 5.15 YOLOv10 RD 66
Table 5.16 YOLOv10 RSD 66
Table 5.17 YOLOv11 RD 66
Table 5.18 YOLOv11 RSD 66
Table 5.19 YOLOv12 RD 67
Table 5.20 YOLOv12 RSD 67

CONTENTS

1 INTRODUCTION....14
2 THEORETICAL FOUNDATION....17
2.1 Artificial Intelligence Support....17
2.2 Latent Diffusion Models (LDMs)....19
2.3 Stable Diffusion....20
2.4 Object Detection with YOLO....22
3 RELATED WORK....28
3.1 Datasets....28
3.2 Object detection with CNNs....30
3.3 Detection implementation in UAVs....33
4 PROPOSED SOLUTION....37
4.1 Creating the dataset....41
4.1.1 Stage I. Preparation of the Data Acquisition Procedure....42
4.1.2 Stage II. Data Processing....43
4.1.3 Stage III. Reliability and Training....44
4.2 Training of computer vision models....46
4.3 Design of the Embedded System....48
4.4 Skyclean design....52
4.4.1 Advantages....52
4.4.2 Equipment Purpose....52
4.4.3 SkyClean Technical Data....53
4.5 Equipment Dimensions....53
4.6 Simulation level Implementation....54
4.7 Implementation on a UAV....55
4.7.1 Image Acquisition....55
4.7.2 Mathematical foundation....56
4.7.2.1 Bounding Box Generation by YOLO....56
4.7.2.2 Class Selection....57
4.7.2.3 Non Maximum Suppression (NMS)....57
4.7.2.4 Visual Display of Detections....58
4.8 System Architecture....59
4.8.1 Hardware Architecture for Training....59
4.8.2 Hardware Architecture of the Embedded System....59
4.8.3 Software Architecture....60
5 RESULTS AND EVALUATION....63
5.1 Training dataset with YOLO versions....63
5.1.1 Results Analysis for YOLOv2....67
5.1.2 Results Analysis for YOLOv3....69
5.1.3 Results Analysis for YOLOv4....70
5.1.4 Results Analysis for YOLOv5....70
5.1.5 Results Analysis for YOLOv7....71
5.1.6 Results Analysis for YOLOv8....72
5.1.7 Results Analysis for YOLOv9....72
5.1.8 Results Analysis for YOLOv10....73
5.1.9 Results Analysis for YOLOv11....73
5.1.10 Results Analysis for YOLOv12....74
5.1.11 YOLOv12 performance....75
5.2 Validation by Simulation....79

5.3 Field Experiments with real UAVs....86
5.3.1 Experiments with the 3DR Iris+ drone....87
5.3.2 Experiments with the Skyclean drone....89
6 DISCUSSIONS AND LESSONS LEARNED....94
7 CONCLUSION....97
7.1 Future Research Directions....98
7.2 Publications and Submissions....98
REFERENCES....104

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs), also called drones, are autonomous or semi-autonomous aerial vehicles with high maneuverability, good stealth, and adaptability (WANG et al., 2023) have been traditionally used in military operations for decades. With the development of related technologies (YANG et al., 2018) they have been successfully applied in many civilian fields, including power line inspection and rescue operations assistance (FLOREANO; J.WOOD, 2015), security and law enforcement (MORAES; FREITAS, 2020), crop row detection and surveillance (BASSO; FREITAS, 2019; VANEGAS et al., 2018), crop yield assessment and plant protection (FENG et al., 2020; TETILA et al., 2020), among others (PASANDIDEH et al., 2022).

Given its ability to carry different systems and accept compatibility with various sensors, the drone emerges as an alternative for tasks in hard to reach areas or those that pose risks to human presence. In other words, the drone is representative of “synesthetic media technologies” (JABLONOWSKI, 2017) that combine a plethora of sensing modes to produce multisensory knowledge (KAUFMANN, 2016) in human and nonhuman terms.

A paradigmatic example is the maintenance of building facades, which in this work are understood as "facades" referring to glazed curtain wall systems, typical of multi story commercial or residential buildings where the external facade is predominantly composed of glass panels. This is an activity that involves complex technical challenges and high operational risks. The cleaning, inspection, and repair of external surfaces, especially at great heights, require solutions capable of overcoming limitations such as restricted access, adverse weather conditions, and the need for regular interventions to preserve the structural integrity and aesthetic appearance of the buildings.

To achieve this objective without endangering human life, Figure 1.1 presents an example of such usage provided by the Skyclean facade cleaning service offered by the Brazilian company Skydrones[1]. This system is equipped with a water pump capable of generating pressure up to 130 bar, ensuring deep and effective cleaning. It offers versatility with options for hot and cold water, which can be combined with specific chemicals to maximize the efficiency of the cleaning process, adapting to the needs of different types of surfaces and dirt.

In the literature, facade analysis is often formulated as a semantic segmentation

[1] https://www.skydrones.com.br/drone-skyclean/

Figure 1.1: An example of drone cleaning facade application.



Source: Skydrones

problem and still remains quite challenging (LIU et al., 2020). It is important to emphasize that facades in this context do not refer to structures with logos, but rather the main set of windows, doors, and balconies (LIU et al., 2020).

Although buildings are highly organized architectures, their analysis faces several challenges: 1) buildings are often in a complex environment where changes in lighting, visual perspective, and occlusion can make analysis difficult; 2) the shape and precise position of facade components (e.g., windows, balconies) can be very difficult to predict compared to rough estimation due to the diversity of texture, style, and other factors; 3) it is difficult to obtain a large database to train models for automatic identification of the different features(LIU et al., 2020). In addition to these problems, the cleaning task itself can become sub optimal, meaning that certain corners may be left uncleaned because they are not “visible" due to these specific characteristics.

Therefore, within the presented context, this work proposes the development of a solution aimed at granting greater autonomy to the Skyclean facade cleaning system a drone developed by Skydrones and illustrated in Figure 1.1 through the automatic detection of windows (SKYDRONES, 2025). To this end, the construction of a dataset comprising both real images collected from public repositories and synthetic images generated by generative models is proposed. Based on this dataset, convolutional neural networks of the YOLO family, optimized for object detection tasks and specifically tailored for identifying windows on building facades, was trained.

The proposed solution integrates a computer vision subsystem based on artificial intelligence responsible for the precise identification of glass windows with an embedded control subsystem onboard the drone. This control subsystem leverages the local-

ization and bounding box information of the detected windows provided by the detection model to autonomously command the activation of high pressure water jets and execute the cleaning procedure, thereby optimizing operational efficiency and reducing the need for human intervention.

The contributions of this work provide the foundation for the development of the proposed system, aligning with the general objective, which guides the structuring and integration of the solution as a whole. They are also aligned with the specific objectives, which organize the stages, functionalities, and validations of the research.

General Objective:

- To develop a complete system for detecting glass windows on building facades using autonomous drones.

Specific Objectives:

- To create a training dataset using selected web images and images generated by Generative Models.
- To train computer vision models using images of building facades, with the objective of performing specific detection and segmentation of glazed windows.
- To implement the trained weights into an embedded detection algorithm for integration in a UAV system.
- To implement the system at the simulation level using ROS, Gazebo, and PX4 Autopilot.
- Deploy the implemented system in a physical UAV.

This research is organized as follows: Chapter 1 presents the context, motivation, approach, proposal, and main contributions of the work. Chapter 2 describes the technical and theoretical concepts used. Chapter 3 analyzes previous studies and research relevant to the context of the work. Chapter 4 details the methodology and development of the system, including the architecture and main components. Chapter 5 presents the results obtained and the evaluation of the performance of the system. Chapter 6 discusses the limitations, observations, and lessons learned during the development of the work. Finally, Chapter 7 summarizes the main conclusions of the work and suggests directions for future research. "

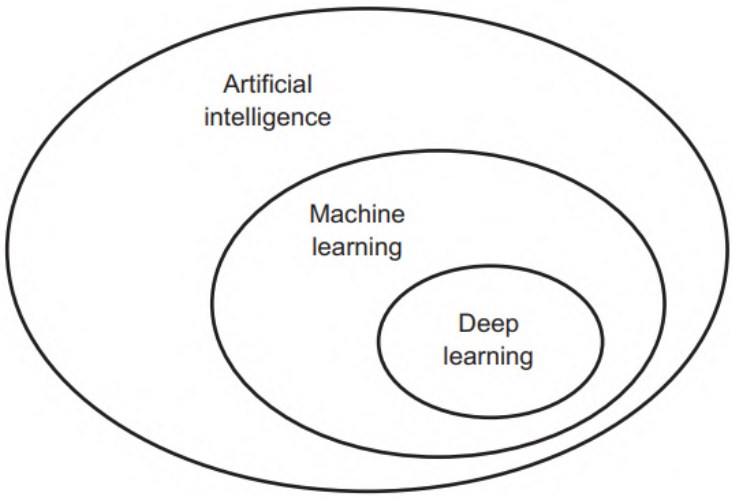
2 THEORETICAL FOUNDATION

Recent advances in Artificial Intelligence have enabled significant progress in visual perception systems, particularly through the integration of deep learning and computer vision. Learning based models can automatically extract meaningful representations from images, supporting complex tasks such as image generation and object detection. Generative approaches, including diffusion models, and real-time detectors such as YOLO have become key tools in modern computer vision pipelines. This chapter presents the theoretical foundations that support these methods.

2.1 Artificial Intelligence Support

Artificial Intelligence can be described as the effort to automate intellectual tasks typically performed by humans. As such, as shown in the Figure 2.1 AI is a general field that encompasses machine learning and deep learning, but also includes many other approaches that may not involve any learning (CHOLLET, 2021).

Figure 2.1: Artificial intelligence, machine learning, and deep learning



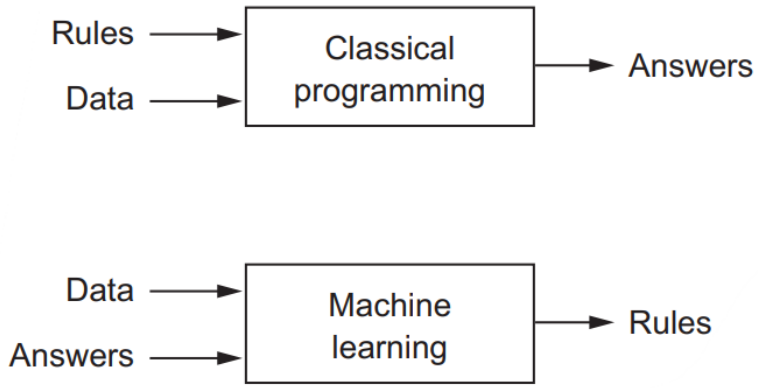
Source: (CHOLLET, 2021)

Computer vision, a subfield of AI, focuses on enabling machines to interpret and make decisions based on visual data. The integration of deep learning techniques into computer vision has significantly improved the accuracy and efficiency of image and video analysis, leading to advances in various applications, such as autonomous vehicles, medical imaging, and facial recognition systems. The goal of computer vision is to

extract useful information from images (PRINCE, 2012).

Machine learning focuses on developing algorithms and models that allow computers to learn from data. Instead of following explicitly programmed instructions, machine learning systems identify patterns and make predictions or decisions based on input data. As shown in the figure 2.2 machine learning system is trained rather than explicitly programmed (CHOLLET, 2021).

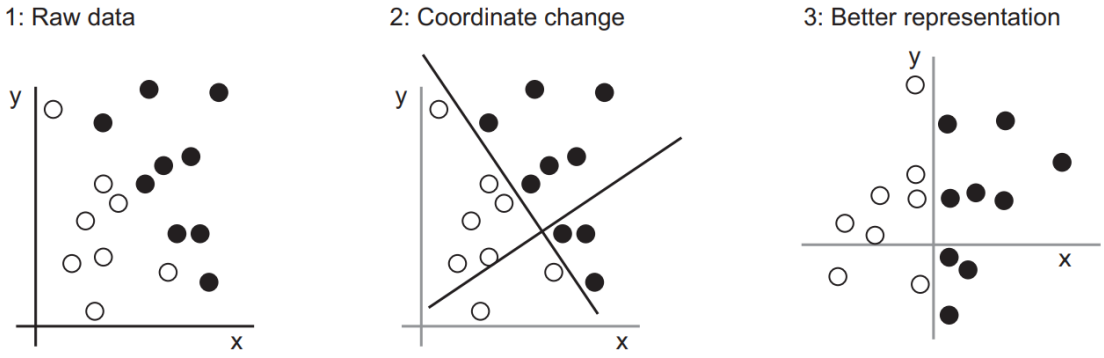
Figure 2.2: Classical programming vs Machine Learning



Source: (CHOLLET, 2021)

Therefore, the central problem in machine learning and deep learning is to meaningfully transform data: in other words, to learn useful representations of the input data at hand representations that get us closer to the expected output (CHOLLET, 2021), as shown in the Figure 2.3.

Figure 2.3: Forms of data representation



Source: (CHOLLET, 2021)

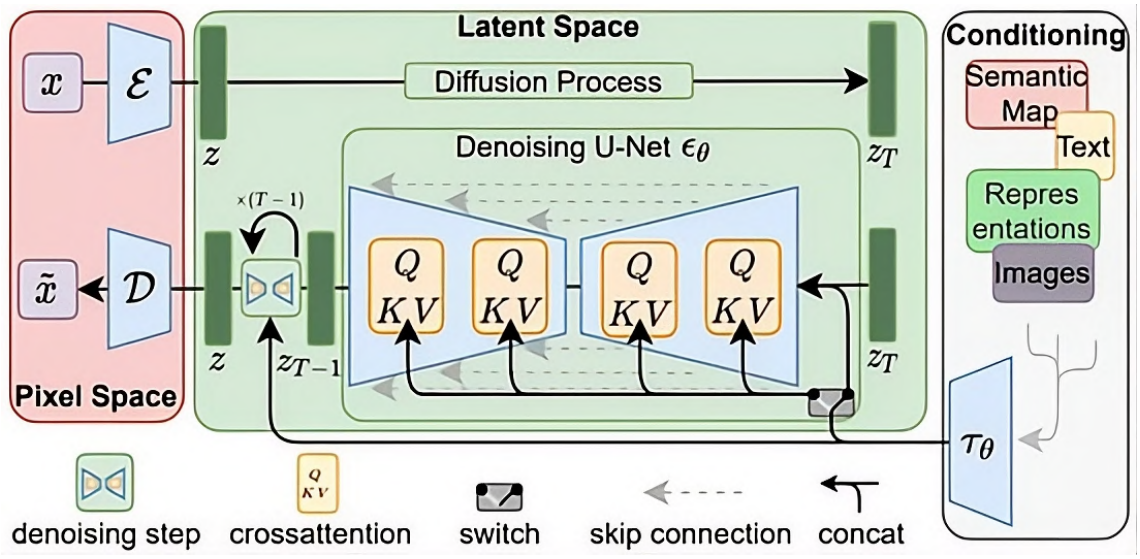
By extracting complex and subtle features from images, these techniques enable the precise identification of relevant structural and architectural elements. The use of deep learning not only enhances the robustness of the system but also accelerates the analysis process, making it more scalable and suitable for a wide range of scenarios.

2.2 Latent Diffusion Models (LDMs)

Diffusion Models dominate the generative image landscape, often operating in latent spaces defined by VAEs (Variational Autoencoders) for greater efficiency. Although GANs (Generative Adversarial Networks) remain relevant in niche tasks such as upscaling and style transfer, Diffusion Models now serve as the foundation for most text to image and image to image systems (SORDO et al., 2025).

They are probabilistic models designed to learn a data distribution p(x) by gradually denoising a normally distributed variable, which corresponds to learning the reverse process of a fixed Markov Chain of length T (ROMBACH et al., 2022). The authors suggest using an encoder network to transform the input into a latent representation, where the forward diffusion process is applied. Then, the reverse process follows the same principle as a standard diffusion model, employing a U-Net to generate new data, which are finally reconstructed back into the original space by a decoder network. (see Figure 2.4).

Figure 2.4: Diagram of the Latent Diffusion Model (LDM) architecture.



Source: (ROMBACH et al., 2022)

These models can be interpreted as an equally weighted sequence of denoising autoencoders $\varepsilon_\theta(x_t, t)$; $t = 1, \ldots, T$, which are trained to predict a denoised variant of their input x_t, where x_t is a noisy version of the input x with t uniformly sampled from $\{1, \ldots, T\}$ (ROMBACH et al., 2022). Therefore, given a pre-trained VAE encoder $\mathscr{E}$, which maps an image $\mathbf{x}$ to a latent representation $\mathbf{z} = \mathscr{E}(\mathbf{x})$, the diffusion process is applied in the latent space. The training objective for the Latent Diffusion Model (LDM) is defined as:

$$\mathscr{L}_{\text{DM}} = E_{x,\varepsilon\sim\mathscr{N}(0,1),t}\left[\|\varepsilon - \varepsilon_\theta(x_t,t)\|_2^2\right], \tag{2.1}$$

where the noisy latent $\mathbf{z}_t$ is generated via the forward diffusion process:

$$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{z} + \sqrt{1-\bar{\alpha}_t}\,\varepsilon, \tag{2.2}$$

and where:

- $\mathbf{z} = \mathscr{E}(\mathbf{x})$ is the latent code of the input image $\mathbf{x}$;
- $\varepsilon \sim \mathscr{N}(0,\mathbf{I})$ is standard Gaussian noise;
- t is a timestep sampled uniformly from $\{1,\ldots,T\}$;
- θ is the learnable parameter;
- $\varepsilon_\theta(\mathbf{z}_t,t)$ is the model's prediction of the noise.

Latent Diffusion Models (LDMs) are yet another innovative extension of Diffusion Models (ROMBACH et al., 2022). Instead of applying the diffusion on a high dimensional input (namely pixel or image space), is project the input image into a smaller latent space and apply diffusion with the obtained latents as inputs (SORDO et al., 2025).

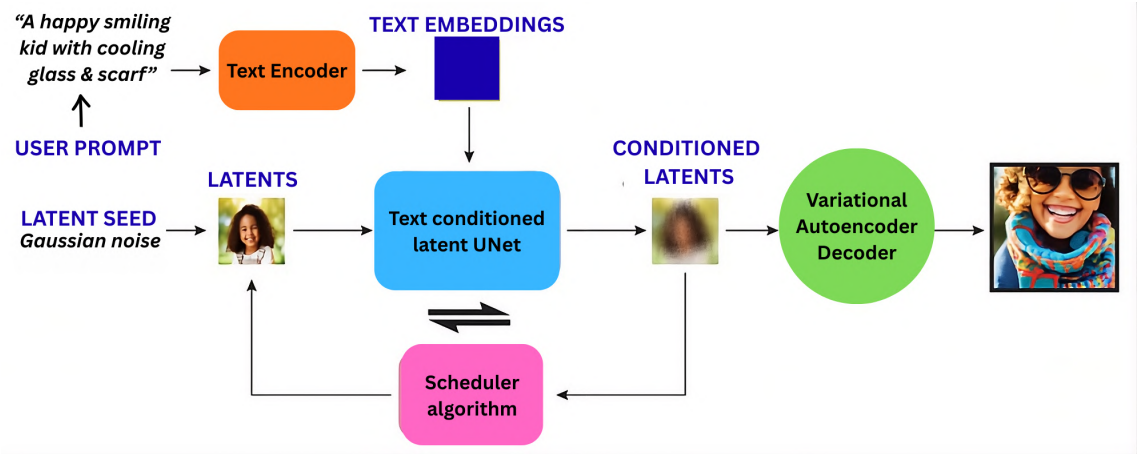
At a simpler level, a diffusion model gradually adds noise to data and then learns how to reverse that noise to recover the original data. The training teaches the model to "denoise" a random input step by step until a clear image appears. This is similar to unmixing paint colors layer by layer to retrieve the original scene. And combine the advantages of diffusion models known for generating realistic outputs, and autoencoders used for compressing data. Operating in a latent space makes them much more efficient and faster than regular diffusion models.

2.3 Stable Diffusion

Although Generative Adversarial Networks (GANs) suffer from mode collapse and unstable adversarial training, and Variational Autoencoder (VAEs) tend to produce blurry or low sharpness images, Stable Diffusion overcomes these limitations by generating high resolution images full of details and with great variety from prompts, making it an excellent alternative for data generation. The Figure 2.5 presents the Stable Diffusion architecture, showing how a user-provided text is transformed into an image. The process begins with the prompt being converted into embeddings by a text encoder, while, in par-

allel, an initial Gaussian noise is generated in the latent space. This noise is iteratively refined by a text-conditioned U-Net, guided by a scheduler algorithm that controls the progressive removal of noise. At the end of the iterations, structured latent representations aligned with the prompt's meaning are obtained and then decoded by a Variational Autoencoder (VAE), resulting in a final image consistent with the given description.

Figure 2.5: Stable Diffusion Architecture.



Source: (VIVEKANANTHAN, 2024)

Accordingly (VIVEKANANTHAN, 2024) Stable Diffusion provides several notable advantages for image generation and editing:

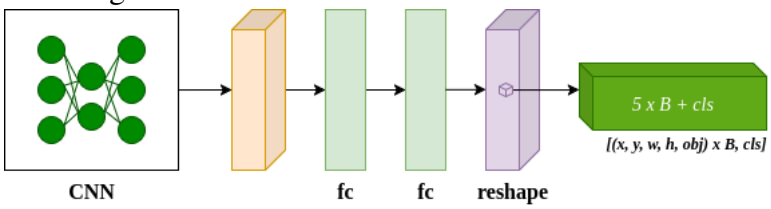
- Superior Visual Quality: It produces images with fine details and lifelike textures through its iterative enhancement process.
- Consistent Semantic Integrity: The model preserves the original meaning of the input, generating images that remain coherent and aligned with the intended content.
- Broad Feature Diversity: It effectively mitigates mode collapse, offering a wide range of features and variations in the generated images, thereby avoiding repetitive outputs.
- Broad Applicability: It performs well on both static and dynamic images and can handle different noise levels, providing users with greater control over the final result.

Therefore, Stable Diffusion overcomes the limitations of previous models by generating detailed, varied, and semantically coherent images in the latent space. Its efficiency and quality make it an excellent choice for creating synthetic data, especially in applications that require large volumes of realistic images for training computer vision models, as in the case of this work.

2.4 Object Detection with YOLO

The YOLO algorithm, introduced in 2016 by Joseph Redmon and collaborators, is a series of object detection models that revolutionized computer vision with its efficiency and speed. The initial model, described in (REDMON et al., 2016a), uses a single neural network to predict bounding boxes and class probabilities from an image in one pass. Subsequent versions have improved accuracy and performance. This study provides an overview of the YOLO models used for training.

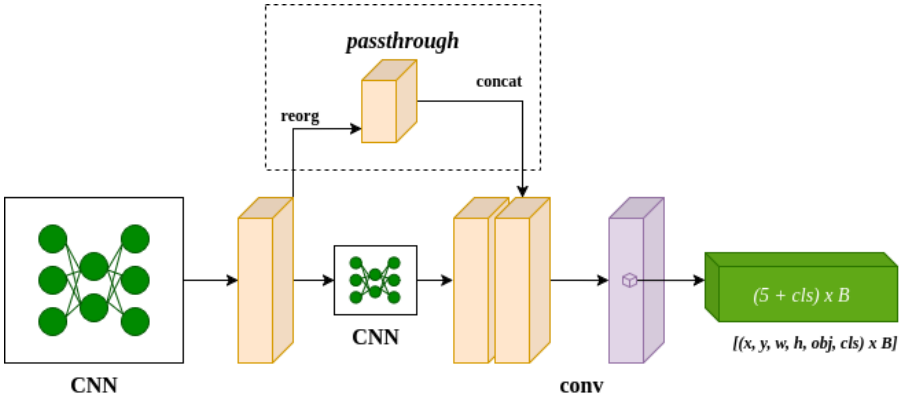
Figure 2.6: The network architecture of YOLOv1



Source: The Author

The Figure 2.6 illustrates the initial architecture of the YOLO network, a neural network designed to segment the input image into a grid of cells, where each cell is responsible for predicting a set of bounding boxes. Initially, the input image is processed by a convolutional neural network (CNN) for feature extraction. Subsequently, these features are passed through two fully connected layers, which integrate the information to capture global image characteristics (WANG et al., 2023).

Figure 2.7: The network architecture of YOLOv2

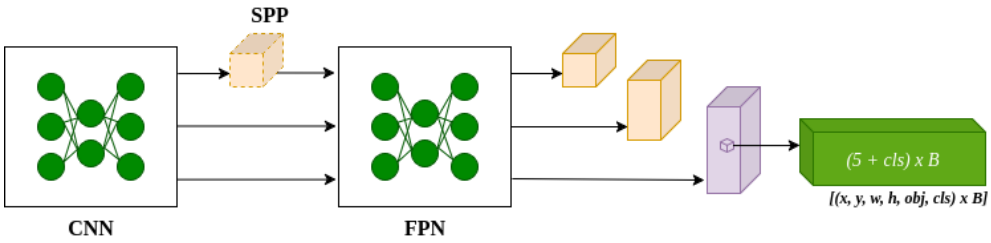


Source: The Author

YOLOv2 (REDMON; FARHADI, 2016), proposed by Redmon and Farhadi, combines new and existing techniques to create a fast and accurate object detector, based on a fully convolutional network and anchor based predictions. In Figure 2.7, it is possible to see that they converted the entire object detection architecture into a fully convolutional

network (WANG; LIAO, 2024). Its main innovations include the use of k-means with IoU (Intersection over Union) to generate more stable anchors (dimension cluster), direct prediction of the object's position, a passthrough layer to improve small object detection, resolution calibration to align pre-training and detection, and joint training with the WordTree hierarchy, enabling the detection of up to 9000 categories.

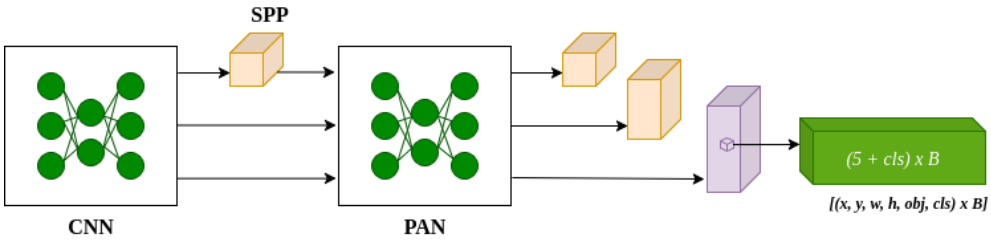
Figure 2.8: The network architecture of YOLOv3



Source: The Author

YOLOv3 (REDMON; FARHADI, 2018), proposed by Redmon and Farhadi in 2018, enhanced one-stage detectors by integrating advanced techniques and optimizations. As shown in Figure 2.8, its architecture combines FPN (Feature Pyramid Network) for multi-scale predictions, incorporates residual networks, and introduces DarkNet53. In addition, it introduced changes to label assignment: each ground truth is associated with only one anchor, and objectness classification shifted from soft labels to hard labels based on IoU. The use of SPP (Spatial Pyramid Pooling) made it possible to combine features at different resolutions, from local to global, significantly improving accuracy.

Figure 2.9: The network architecture of YOLOv4



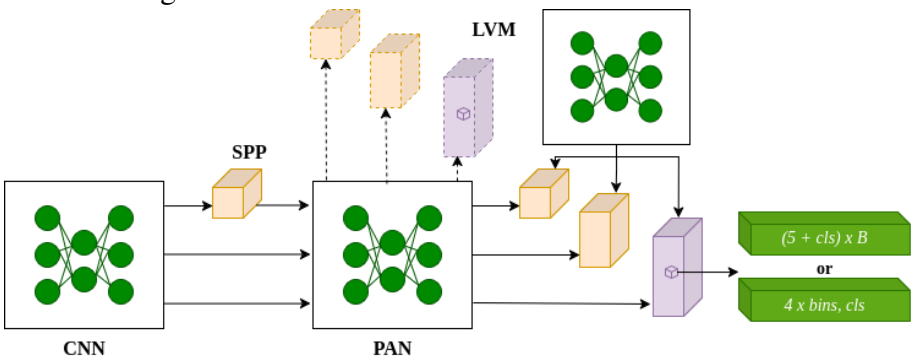
Source: The Author

YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020), released in April 2020 by Alexey Bochkovskiy, enhances real-time object detection by incorporating recent advances in computer vision. It employs CSPNet as the backbone and replaces FPN with PAN (Path Aggregation Network), as shown in Figure 2.9, serving as the foundation for subsequent YOLO architectures. Trainable on a single GPU, it achieves inference speeds up to three times faster than multi-GPU models. Its innovations include Bag of Freebies

and Bag of Specials to improve accuracy without compromising inference, Grid Sensitive Decoder for better edge prediction, Self Adversarial Training for increased robustness, and Training with Memory Sharing to optimize memory usage and increase batch size.

YOLOv5 (ZHU et al., 2021), based on PyTorch YOLOv3, refines its architecture while adopting EfficientDet's model scaling to offer various model sizes. Initially released in June 2020 as YOLOv5 r1.0, it integrated CSPNet and CSPPAN from YOLOv4, with subsequent optimizations in speed and accuracy trade-offs, activation functions, and YOLOR based hyperparameters. The latest version, YOLOv5 r7.0, which shares the same architecture as shown in Figure 2.9, was released in November 2022. Licensed under AGPL3, YOLOv5 is widely used due to continuous updates. Key features include a Power based Decoder for stable training by limiting anchor scaling, improving recall, and Neighborhood Positive Samples to enhance prediction accuracy by including neighboring grids and adjusting the sigmoid scaling for center point decoding.

Figure 2.10: The network architecture of YOLOv7

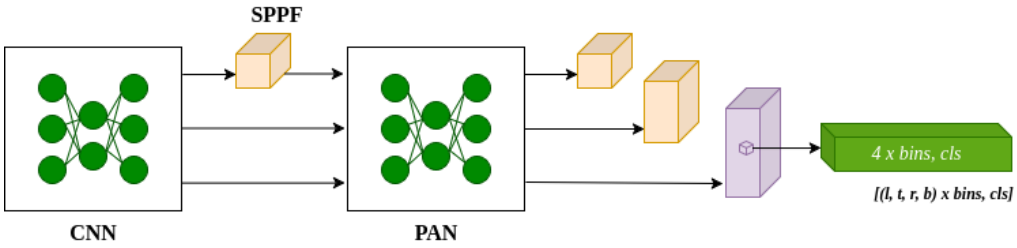


Source: The Author

YOLOv7 (WANG; BOCHKOVSKIY; LIAO, 2023a) introduces trainable auxiliary architectures, integrable or removable during inference, incorporating elements from YOLOR, RepVGG, and auxiliary losses. It replaces CSPNet with ELAN and proposes E-ELAN for large scale models, supporting anchor based and anchor free designs across various computer vision tasks like object detection, instance segmentation, and pose estimation, achieving real-time state of the art performance. Key innovations include: Planned RepConv to extend RepVGG reparameterization to residual and concatenation based architectures; Consistent Label Assignment to ensure alignment between branches using dynamic methods; Coarse to Fine Label Assignment for direct refinement of predictions in feature space via auxiliary losses; Partial Auxiliary Loss to selectively update features while prioritizing main tasks; and versatile models optimized for multiple vision applications. As shown in Figure 2.10, the CNN extracts image features, the SPP organizes

these features across multiple scales to handle objects of varying sizes, the PAN combines information from different layers to refine detection, and the final layer (LVM) generates predictions for object localization and classification.

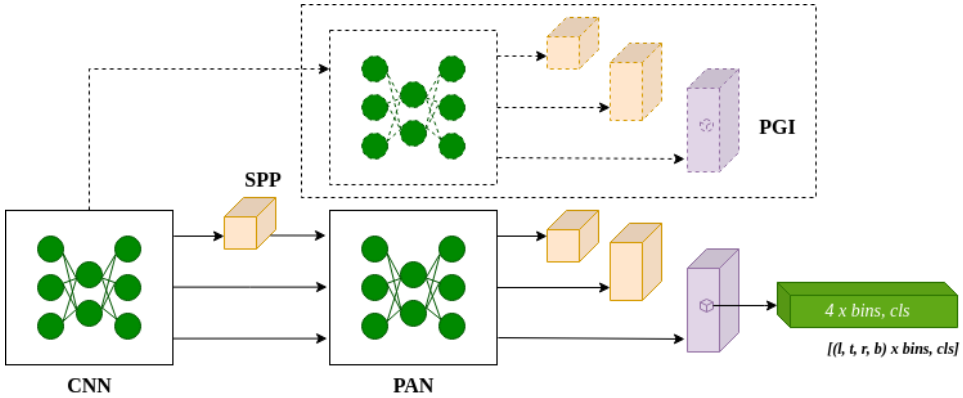
Figure 2.11: The network architecture of YOLOv8



Source: The Author

YOLOv8 (REIS et al., 2024), a reformulation of YOLOv5, enhances the API interface and implements significant optimizations in the underlying code. As shown in Figure 2.11, architecturally, it refines the ELAN from YOLOv7 while retaining the decoder from YOLOv6 2.0, functioning as a technological integration platform rather than a standalone new version. Its latest version, Glenn, incorporates advancements such as YOLOv9 and YOLO World. Key features include code optimization, improving training performance by approximately 30%, and a simplified API that facilitates the integration of the detection model with advanced tasks such as segmentation, pose estimation, and multi-object tracking.

Figure 2.12: The network architecture of YOLOv9

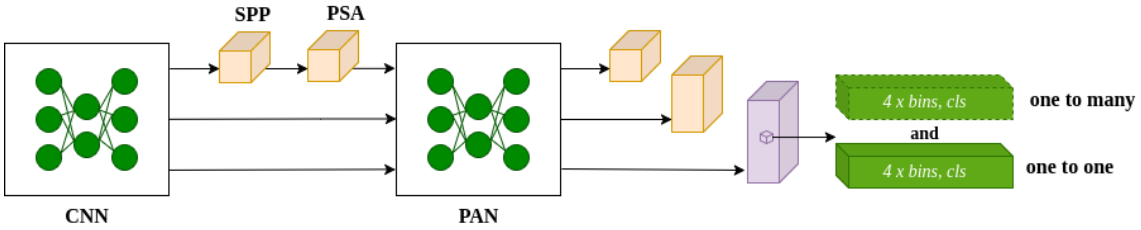


Source: The Author

YOLOv9 introduces Programmable Gradient Information (PGI), shown in Figure 2.12, enhancing model interpretability, robustness, and versatility using reversible architecture and multi-level information retention (WANG; LIAO, 2024). It extends ELAN to G-ELAN for high accuracy and speed with fewer parameters. Key features include: Auxiliary Reversible Branch to address information bottlenecks; Multi-level Auxiliary

Information to retain task critical data; Generalization to Down stream Tasks for robust performance in small datasets and multi-task learning; and Generalization to Various Architectures for compatibility with diverse models.

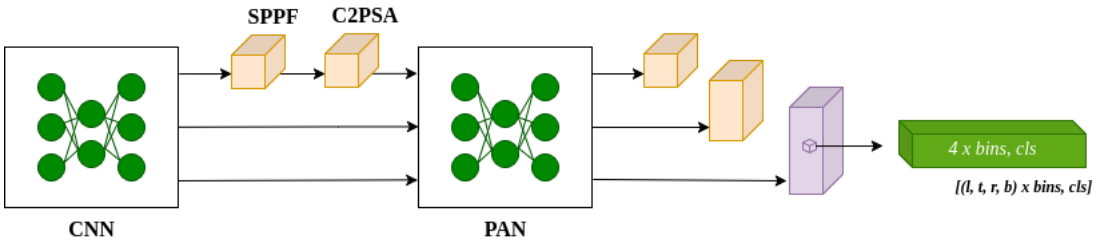
Figure 2.13: The network architecture of YOLOv10



Source: The Author

As shown in Figure 2.13, YOLOv10 has an architecture similar to YOLOv6 3.0 but adds a Transformer module for improved global feature extraction and uses one to many and one to one matching, enabling end to end detection without NMS. Its innovations include Dual Label Assignment, Rank guided Block Design to determine convolution types, and Partial Self attention combining CSPNet and Transformer.

Figure 2.14: The network architecture of YOLOv11

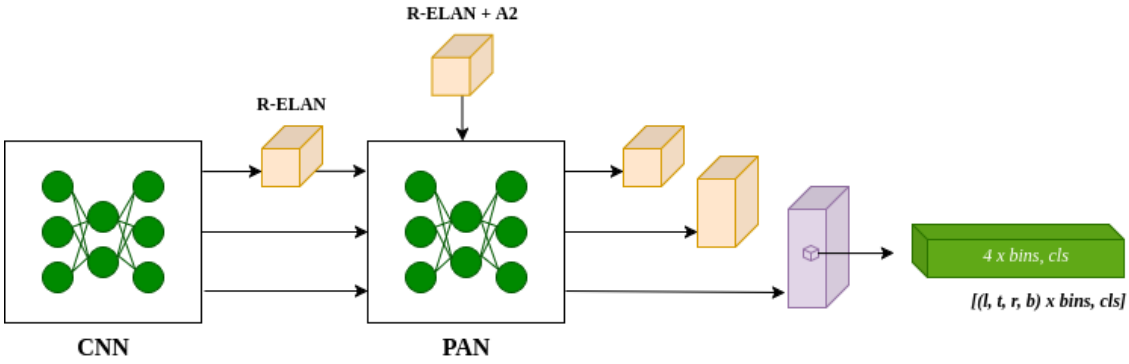


Source: The Author

YOLO11 keeps the backbone–neck–head design but replaces C2f blocks with lighter, faster C3k2 CSP bottlenecks to speed up processing. It retains the SPPF and adds the C2PSA module for spatial attention, improving detection of small or occluded objects. The neck still combines multi-scale features but now with C3k2 for greater efficiency. In the head, multiple C3k2/C3k blocks refine the features, followed by CBS (Conv-BN-SiLU) layers and final convolutions that produce bounding box coordinates, objectness scores and class predictions, all consolidated in the Detect layer (KHANAM; HUSSAIN, 2024). In Figure 2.14, the overall architecture of YOLO11 is illustrated, highlighting the backbone, neck, and head components and their respective modules.

YOLOv12 improves the YOLO architecture by incorporating efficient attention through the area attention module, which reduces computational cost while maintaining a large receptive field. It introduces R-ELAN, with a residual shortcut and bottleneck

Figure 2.15: The network architecture of YOLOv12



Source: The Author

design, to enhance feature integration, reduce memory usage, and speed up convergence. The backbone preserves the hierarchy of previous versions but with fewer blocks in the final stage (TIAN; YE; DOERMANN, 2025). Other optimizations include the A2 Module, R-ELAN, and enhancements to the overall model structure, including FlashAttention and reduced computational overhead in the multi-layer perceptron (MLP), balancing performance, speed, and stability for real-time applications (KHANAM; HUSSAIN, 2025). In Figure 2.15, the architectural modifications of YOLOv12 are presented, emphasizing the integration of area attention, R-ELAN blocks, and the optimized backbone structure.

3 RELATED WORK

This section presents and critically analyzes the main works directly related to the scope of this research. The review is structured to contextualize the proposed contribution within three complementary dimensions: (i) the availability and limitations of datasets for facade analysis, particularly those focused on modern architecture with a predominance of glass; (ii) advances in object detection and segmentation based on Convolutional Neural Networks (CNNs), with emphasis on applications involving Unmanned Aerial Vehicles (UAVs); and (iii) practical implementations of deep learning models embedded in UAVs for facade inspection, defect detection, and autonomous navigation.

3.1 Datasets

The increasing application of image guided Unmanned Aerial Vehicles (UAVs)/drones in infrastructure inspection and maintenance is a current research challenge, given the scarcity of suitable datasets for facades glass windows classification in modern architectural buildings.

The identification of building facades through computer vision has become a field of interest, driven by applications ranging from architecture and urban planning to automation and urban monitoring. Their semantic information is important for widespread applications such as sunlight analysis, light pollution assessment, energy conservation, emergency response, and so on (MAO et al., 2022). The development of algorithms capable of classifying facades with different styles, textures, and shapes requires a significant volume of data for training neural networks.

Considering that modern building is the most common style in cities and is growing much faster than the buildings of other architectural styles, suitable samples are urgently needed to train the classifiers that segment the free facade accurately (WEI et al., 2024). These datasets should include images under different lighting conditions and from various angles, ensuring that deep learning models are trained to handle real-world challenges.

Thus, the lack of previously available datasets in the literature poses a significant challenge for the development and validation of UAV-based applications based on computer vision models, affecting researchers and developers in areas as facade cleaning and building inspection. This context justifies the collection and curation of a new image

dataset, ensuring full control over the collected, selected, labeled, and annotated data.

The creation of a dataset for building facade identification using computer vision faces challenges at various stages, impacting data quality and the performance of deep learning models. Image acquisition, curation, and accurate labeling of facade components are essential for improving model accuracy and reliability. Techniques such as data augmentation, learning rate adjustments, normalization, and regularization can enhance algorithm performance, reducing the need for extensive manual annotations.

Compared to other existing datasets, such as CMP Facade and ECP Paris, which focus on neoclassical style buildings, the dataset proposed in this work offers a large set of labeled data on contemporary architecture characterized by the use of glass facades. As stated in (WEI et al., 2024): “the precise ground truth still must be manually annotated by humans".

Facing the challenge of facade segmentation, (MAO et al., 2022) introduced the Glass Facade Segmentation Dataset (GFSD) based on an ensemble method to identify glass facades in oblique aerial images, including 356 oblique aerial images and 44 ground view images of glass facades. The study demonstrates that the segmentation of glass facades in oblique aerial images can be significantly improved through an ensemble model and a new annotation strategy. Although the proposed method has proven effective, it still faces challenges, such as the segmentation of small facades or those in areas close to the ground, where distortion is higher.

(WEI et al., 2024) provided a study of a pixel level annotated dataset of low regularity facades, named Irregular Facades (IRFs). The IRFs were developed based on 1057 pixel level annotated facade images with six classes: Background, Plant, Wall, Window, Door, and Fence. Their study shows that models trained on traditional facades struggle with modern ones and emphasizes that sample quality and material diversity matter more than quantity for generalization. Furthermore, the research confirms that classifiers trained with classic facade images are not able to effectively segment modern facades.

The work reported in (MEI et al., 2022) discusses the glass detection problem. First, it constructs a large scale glass detection dataset (GDD), which consists of 3,916 images and their respective masks. Second, it proposes a new glass detection network (GDNet-B), which includes multiple well designed large field contextual feature integration (LCFI) modules and two boundary feature enhancement (BFE) modules. Although the results are promising, the model still faces challenges, such as detecting glass with low reflection or in environments with unfavorable lighting. In the context of dataset construc-

tion, (HABILI et al., 2022) introduces the LIB-HSI dataset, featuring hyperspectral and RGB images of light industrial building facades to classify materials such as wood, brick, and concrete using deep learning based semantic segmentation. Results show improved material classification and segmentation. While not focused on glass facades, extending analysis to glass could reveal optical properties, transparency, coatings, and environmental impacts.

While datasets such as CMP Facade (TYLECEK, 2012), eTrims (KORČ; FÖRSTNER, 2009), and ECP Paris are widely recognized in facade analysis, they provide limited coverage of glass facades, which are the primary focus of this study. This gap constrains the advancement of deep learning models for the detection and segmentation of such structures, highlighting the need for an up-to-date dataset.

Considering the scarcity of studies focused on this subject within the existing literature, this work presents a significant contribution to the field by addressing limitations found in existing studies and datasets. While most publicly available datasets include elements, and other non-relevant components, this study focuses mainly on images of glass windows, both in residential houses and large commercial buildings.

3.2 Object detection with CNNs

Diverse deep learning based approaches have been proposed to enable drones to identify features on building facades and safely navigate along them. These approaches range from semantic segmentation to specialized object detection techniques. A summary review of the ones most related to this current research is presented in the following.

To provide the drone with environmental perception and the ability to avoid obstacles while ensuring navigation safety and accuracy, (WANG et al., 2020) proposes the use of an onboard RealSense D435 camera attached to the drone. This camera detects obstacles while simultaneously capturing color and depth images of the flight scene. Using the YOLO V7 network architecture (WANG; BOCHKOVSKIY; LIAO, 2023b), they detect the category and bounding box of the obstacles.

For obstacle detection, (LEE; HO; ZHOU, 2021) proposes the Faster RCNN method because it combines two modules: 1) a fully convolutional deep network that proposes regions and 2) a Fast RCNN detector that utilizes the proposed regions. According to this article, the use of a fully convolutional deep network significantly improves the computational time for region proposal and enables real-time object detection. To achieve this,

Faster RCNN networks are built from scratch and from pre-trained models.

To address contextual information in facade analysis, where segmentation accuracy is relatively low, (ZHANG; PAN; ZHANG, 2022) suggests using the Pyramid Scene Parsing Network (PSPNet) to extend global and local features through context aggregation based on different regions. In this article, they develop a hybrid deep learning approach that integrates image semantic segmentation and object detection methods to improve detection accuracy. The model developed by this work is programmed using Pytorch and trained on four NVIDIA GeForce 2080Ti GPUs with 11 GB of memory.

According to (LIU et al., 2020), windows, doors, and balconies are the most important structures in facade analysis. Since most of these structures have a square shape, it is possible to use this prior information to significantly improve detection. Object detection generates bounding boxes that show the location of these objects; if the boxes can correctly locate the objects, the results are significantly refined. To perform these procedures, this article also uses the Faster RCNN architecture to detect the objects and generate the bounding boxes.

However, (NORDMARK; AYENEW, 2021) suggests the Mask RCNN model, which is a two-stage framework. The first stage analyzes the image and generates responses, while the second stage classifies the proposals and generates masks and bounding boxes. Mask RCNN is an intuitive extension of the Faster RCNN architecture and generates predictive segmentation masks for each region of interest (RoI). In Mask RCNN, the mask represents pixel level segmentation and the alignment of each object, aiding in extracting each object instance separately without background. For this study both the training and inference was done on Google Colab with the free Tesla K80 GPU of about 12GB.

To achieve more precise navigation capability through images, (WANG et al., 2023) proposes the fusion of target detection algorithms based on CNNs like YOLO with the Deep Q-Network method for obstacle detection. They used both the Faster RCNN and YOLO architectures, where YOLO demonstrated lower accuracy and a higher missing detection rate for smaller objects in the image. They optimized the output of the Faster RCNN model according to specific scenario characteristics and UAV kinematic properties, thus reducing the training difficulty of the recognition network.

For solve the problem of labeling building facades he article (BORTOLOTI et al., 2022) employs two approaches to train its networks: a primary and a secondary approach. In the primary approach, they perform training via transfer learning on Google

Street View imagery, which they refer to as Faster RCNN Facades (FRF). The secondary approach involves using a pre-trained CNN model on the Places365 database and remapping the outputs to land use classes. According to their mAP results, when using pre-trained weights (transfer learning), the models perform better than when trained from scratch.

To perform salient object detection, the article (SRIVASTAVA; SRIVASTAVA, 2020) use the YOLOv2 network to identify the bounding boxes of objects. Next, they apply boundary correction to the predicted boxes. After this step, the image is segmented using Gabor filters, and the corresponding segment is selected to apply a second boundary correction. Finally, the method offers the user the option to choose between first or second level correction.

Still on salient object detection (SOD), Yuyan (LIU et al., 2025) presents an innovative approach using weakly supervised learning with scribble based annotations. Its main contribution is the dual flow framework, which separates edge feature extraction from salient object feature extraction, allowing the model to learn how to identify precise edges and structural details of objects. The main challenge of scribble based methods is the lack of precise structural information, which makes it difficult to identify clear edges and fine details of salient objects. However, fully supervised methods require pixel level annotations, which are expensive and time consuming to produce. Experimental results show that the proposed method outperforms existing weakly supervised methods and is competitive with some fully supervised approaches, especially in complex scenarios with multiple objects and complicated backgrounds.

To perform small object detection in aerial images captured by drones (UAVs), (PAN; YANG, 2024) proposes an improved and lightweight version of the YOLOv8 algorithm. Detecting small and densely distributed objects in aerial images is a significant challenge due to the capture distance, the reduced size of objects, and the hardware limitations of drones. The main contributions include the introduction of lightweight modules (ESEMB and Faster), the improvement of the Neck structure to capture semantic information from small objects, the redefinition of the detection head, and the use of an inner MPDIoU loss function to enhance bounding box regression. The significant reduction in model size and the number of parameters, combined with increased accuracy, makes the proposed model suitable for drone applications, where computational efficiency and precision are critical.

In the same context of small object detection, (ZHANG; PENG; LIU, 2025) uti-

lizes an improved version of YOLOv9 to enhance target detection in aerial images captured by drones. This approach incorporates BiFormer attention mechanisms, RepNCSPELAN4 Ghost modules that replace traditional convolutions with "ghost" convolutions (GhostConv) to reduce computational complexity, a Bidirectional Feature Pyramid Network (BiFPN) to facilitate fusion and information exchange between different network layers, and the Focal WIOU loss function to optimize bounding box regression.

To handle changes in lighting, different viewpoints, and seasonal variations, (ZHAO et al., 2022) proposes the DSD MatchingNet, an architecture designed for matching local features in images. The authors adopt a deformable feature extraction module to extract multi-level feature maps, in which deformable convolutional networks are applied to extract the final feature map. In addition, a fully convolutional structure is adopted as the backbone of the deformable feature extraction module, while convolutions with a stride of two are used to perform downsampling of the feature map.

Considering precision in a fully digital scenario, (WANG et al., 2024) presents the application of deep learning in a digital watermarking process with three main stages: interest point extraction, watermark insertion, and watermark extraction. The Superpoint network detects robust points in the image, selecting areas less vulnerable to attacks such as noise and geometric distortions. The watermark is inserted using a U-Net like network, and the extraction is performed through the ConvNext network, employing a statistical rule to ensure high accuracy in data recovery. The loss function considers both visual quality and precision, ensuring robustness without compromising the appearance of the image.

3.3 Detection implementation in UAVs

In recent years, the use of Unmanned Aerial Vehicles (UAVs) combined with computer vision techniques has driven the development of intelligent systems for the inspection and maintenance of glass facades an essential step for planning automated cleaning operations. Accurate detection of defects and irregularities enables safe and efficient adjustment of cleaning parameters, and convolutional neural network based models, such as YOLO, have stood out for their speed and precision in real-time detection.

Numerous studies explore the use of drones combined with deep learning algorithms to capture aerial images, detect damage, and reconstruct 3D models of surfaces. These works strengthen the advancement of automated facade inspection and provide a

technological foundation for developing autonomous building cleaning systems, integrating visual recognition, operational safety, and energy efficiency.

The following presents the main related works that address the use of drones and neural networks for defect detection and facade mapping, highlighting their technical contributions, limitations, and potential for integration with future applications focused on automated building cleaning.

The paper (PANDEY; BINGI; IBRAHIM, 2025) employs UAVs to capture high resolution images from multiple angles, creating an automated imaging and classification system specifically designed for monitoring the condition of glass facades. To detect and classify defects in real time with high accuracy, several YOLO models (v8, v11, and v12) are implemented. The authors report that their best performance was achieved using YOLOv12, which reached 70% accuracy on the test set. Additionally, SIFT (Scale-Invariant Feature Transform) and RANSAC (Random Sample Consensus) techniques are incorporated to stitch panoramic facade images together, enabling a more precise and continuous reconstruction of the analyzed surface.

Following the same line of research aimed at efficiency and accuracy in embedded systems, the paper (CHEN et al., 2022) proposes a real-time object detection system for drones that combines energy efficiency and high accuracy through the fusion of visible and infrared images. Based on an optimized version of YOLOv2, the model was adapted to run on low-power embedded devices such as the NVIDIA Jetson TX2, balancing CPU and GPU usage to ensure performance and low energy consumption. Decision level fusion integrates the advantages of both image modalities structural detail from visible images and thermal sensitivity from infrared improving detection under low-light conditions, occlusion, and visual clutter. Tested with over 28,000 images collected by a DJI M300RTK drone equipped with a Zenmuse H20T camera, the study concludes that the integration of neural network optimizations and multisensor fusion makes the embedded system more robust, accurate, and applicable to scenarios such as environmental monitoring, surveillance, and search and rescue.

Complementing approaches involving fusion and YOLO architecture optimization, the study by (ZHOU et al., 2025) specifically tackles the challenge of imprecise damage localization and detection difficulties on glass facades. It proposes a damage detection algorithm based on the YOLOv10 architecture, implementing improvements in the backbone, neck, detection layers, and loss functions to optimize model performance. The backbone was replaced with VanillaNet, and the neck incorporated BiFPN, which em-

ploys attention mechanisms and residual connections. Additionally, the detection layer was expanded with a new 160×160 head. Finally, the loss function was enhanced with Inner CIoU, enabling more precise bounding box adjustments and improved damage localization.

Exploring another aspect of automated inspection, the paper (HUANG et al., 2025) introduces CladdingNet, an embedded system that uses a drone with a binocular camera and onboard GPU to perform real-time automatic detection of cracks in stone cladding (e.g., granite). Based on an improved U-Net with transfer learning and Dense Blocks, the model identifies and measures cracks in UAV captured images, classifying damage levels into four states. The system integrates onboard GPU processing, Wi-Fi data transmission, and 3D mapping of damaged panels, achieving high accuracy and outperforming networks such as U-Net, ResNet, and DeepLab-v3. Thus, CladdingNet provides an efficient, safe, and automated solution for structural facade inspection, replacing time consuming and risky manual methods.

Similarly, the paper (SOUSA; SOUSA; MAUéS, 2022) reinforces the potential of UAVs for structural diagnosis by describing the use of Digital Image Processing (DIP) techniques applied to aerial images to detect areas of ceramic cladding detachment on building facades. The detection algorithm employed is YOLOv2, implemented in a study divided into two phases: the first focused on creating an image dataset, and the second subdivided into preprocessing and model application to identify damaged areas. This approach demonstrates the viability of object detection algorithms in building inspections, especially in complex urban environments.

While some works focus on facade inspection, the paper (ALPARSLAN; CETIN, 2021) expands the application field of UAVs by proposing a YOLOv4 based method to assist indoor navigation. The system uses a Convolutional Neural Network (CNN) to detect and classify objects such as doors, windows, and stairs, while data from a 2D LiDAR sensor are used to validate object positions. The fusion of visual and depth information enables identification of passage points, classification as open or closed, and calculation of their width to verify safe drone passage. This proposal, implemented without additional sensors, is particularly suited for operations in disaster affected indoor areas, broadening drone use in search and rescue scenarios.

Continuing architectural improvements to YOLO, the paper (LIU, 2025) proposes a crack detection method based on YOLOv8, optimized for optical inputs, using UAVs to inspect and photograph the exterior walls of 55 residential buildings, yielding 3,208 valid

images for manual annotation. The model incorporates a lightweight multi-scale attention mechanism module (LMSAM) in the backbone's feature extraction layer and an enhanced dual channel feature fusion structure (DCFF) to strengthen integration between deep and shallow features. Additionally, it includes a shallow detection head to improve identification of thin, low contrast cracks and replaces the GIoU loss with E-IoU, significantly enhancing edge localization accuracy.

The paper (MA et al., 2022) focuses on a more general aerial object detection problem and proposes an improved version of YOLOv3 called AVS-YOLO, designed for detecting objects in drone captured aerial images. The work addresses challenges such as small object detection, scale variation, background interference, and class imbalance, introducing several structural improvements: (1) a densely connected feature pyramid network (DFPN) enabling bidirectional fusion between shallow and deep features; (2) a scale aware attention module (SAA) that enhances the network's ability to detect objects of varying sizes; (3) dense residual blocks (RDNet) to prevent network degradation and improve feature extraction; and (4) a median frequency balancing mechanism to handle class imbalance.

Finally, the paper (YOON et al., 2024) presents an automated method for building facade inspection using UAVs and convolutional neural networks (CNNs). The proposal combines automatic defect detection (e.g., cracks) with three dimensional facade reconstruction. The system uses the YOLOv5 model to identify structural elements and defects in drone captured images, whose coordinates are converted into geographic points using the coordinate system and the Haversine formula. These data are then transformed into spatial vectors that serve as the basis for creating 3D models in SketchUp software, with defects represented in their real positions. Experimental results showed that the generated 3D models are highly faithful to actual buildings, with an average total size error of only 1.47%. This method significantly reduces inspection time and cost, increases inspector safety, and provides a more objective and detailed analysis of structural conditions, although opportunities for improvement remain in future studies.

Thus, the reviewed works form a solid foundation for future research aiming to integrate visual recognition, autonomous control, and safe operation of drones in urban environments, contributing to the evolution of technologies focused on the intelligent maintenance of modern buildings.

4 PROPOSED SOLUTION

This research proposes the development of a solution designed to enhance the autonomy of the Skyclean facade cleaning system, a drone developed by Skydrones and illustrated in Figure 1.1, by enabling the automatic detection of windows (SKYDRONES, 2025). To achieve this goal, the work is organized into five main components:

1. Create a training dataset using selected web images and images generated by Generative Models.
2. Training of computer vision models using images of building facades, with the objective of performing specific detection and segmentation of glazed windows.
3. Implementation of the trained weights into an embedded detection algorithm for integration with an unmanned aerial vehicle (UAV) system.
4. Simulation level implementation via ROS, Gazebo, and PX4.
5. Development of a complete system for detecting glass windows on building facades using autonomous drones.

For dataset selection, initially a manual screening was conducted to remove irrelevant or low quality images, where each image was annotated using the LabelImg tool to identify objects such as glass windows, doors, persons, walls, pillars, cars, and generate files compatible with the format required for YOLO network training. After downloading, the images were organized into a directory structure and sequentially renamed to facilitate management. Subsequently, they underwent preprocessing, including resizing and pixel normalization, followed by a thorough validation of the annotations. Finally, all images were stored with backup in a secure system to ensure their preservation.

The inclusion of the *cars* and *persons* classes in the model training is directly related to operational safety requirements. These objects represent dynamic and unpredictable obstacles that the drone must detect and avoid, thereby reducing the risk of collisions during mission execution. Similarly, classes such as *door*, *pillar*, and *wall* correspond to structural elements of the building that also constitute potential obstacles during close-range navigation along the façade. Although they are part of the operational environment, their detection is necessary to allow trajectory adjustments and ensure that the cleaning process occurs without unintended impacts.

Therefore, training these classes is not primarily intended for inspection purposes, but rather to enhance the system's situational awareness. By recognizing both the target

class *glass window* and other relevant environmental components, the drone can make safer and more autonomous decisions during façade cleaning, balancing operational performance with risk mitigation.

To enhance the dataset, a synthetic dataset was added, generated through the Stable Diffusion generative network, accessed via the stable-diffusion-webui interface. The Figure 4.1 on the right shows an image generated using stable-diffusion-webui, while the figure on the left represents an image obtained from the web. The visual differences between them are minimal, highlighting the realism achieved by the generative model.

Figure 4.1: Comparison between an image collected from the web and an image generated with stable diffusion.



Source: Google



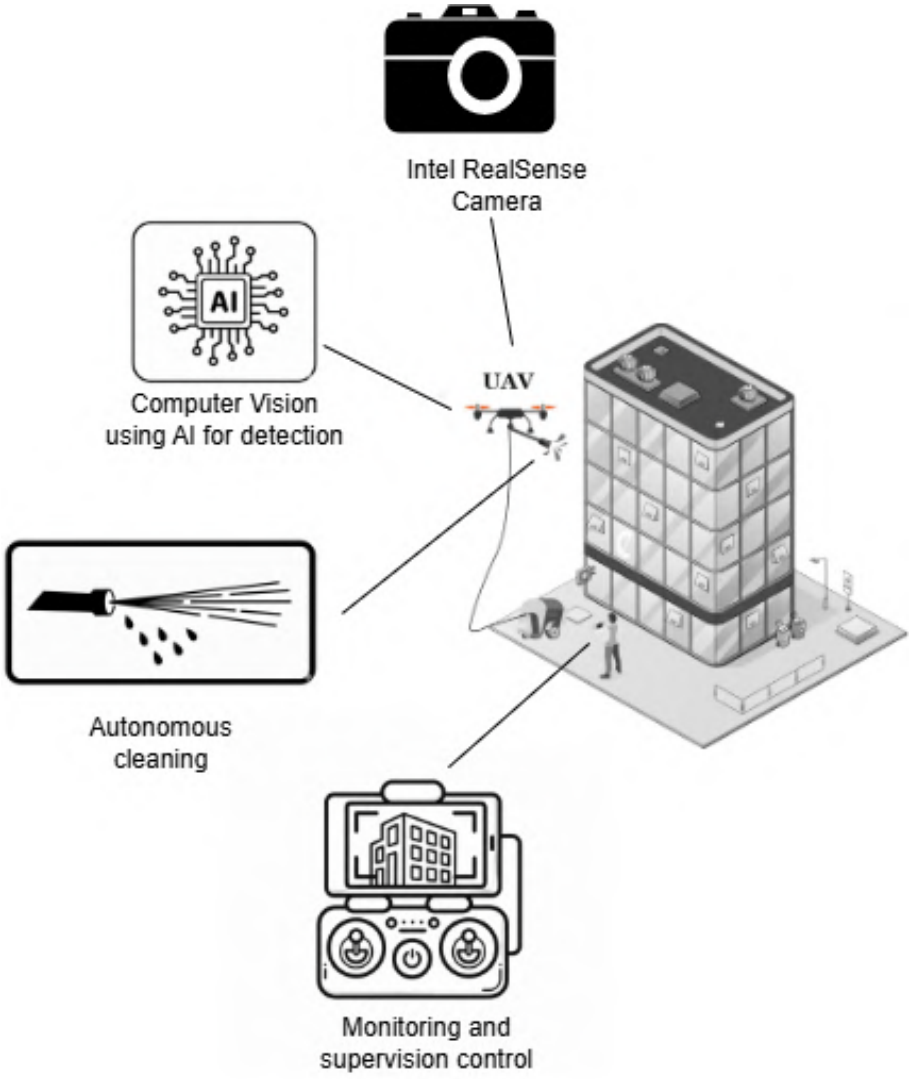
Source: Stable Diffusion

Subsequently, a comprehensive comparative analysis was conducted among ten versions of the YOLO architecture (YOLOv2, YOLOv3, YOLOv4, YOLOv5 YOLOv7, YOLOv8, YOLOv9, YOLOv10, YOLOv11, and YOLOv12), with the aim of identifying the version that would deliver the best performance for the specific application proposed in this study. The YOLOv1 version was deliberately excluded, as it represents the oldest iteration and is currently the least competitive in terms of performance, whereas YOLOv2 was retained in the evaluation due to its well established suitability for devices with limited computational resources.

To develop a solution that provides greater autonomy to the facade cleaning system, a computer vision subsystem based on artificial intelligence was created for the detection of glass windows, integrated with an automated control subsystem embedded in the drone. This control subsystem uses the information from the detection process to activate high pressure jets and perform the cleaning process automatically.

Figure 4.2 illustrates the architecture of the proposed solution, highlighting its main components. Initially, the Intel RealSense camera performs continuous streaming

Figure 4.2: Solution Overview



Source: The Author

and real-time image capture. Next, the pre-trained computer vision model, loaded with the corresponding weights, executes the detection of targets of interest, providing the necessary signals to trigger the autonomous cleaning system. Finally, the human operator monitors the entire process through the monitoring and supervision module, ensuring the human presence and intervention required by current legislation, even in an autonomous context (ANAC, 2025).

4.1 Creating the dataset

To create this dataset, a workflow was developed. The Figure 4.3 illustrates the image preparation stages for training an AI model, as well as its potential applications, both for image capture during the training phase and for using the trained model in detection tasks. These tasks are crucial to ensure that the files are properly structured for further processing. Each stage is composed of subtasks specifically organized to provide a step-by-step sequence for further study application, and reprodutibility.

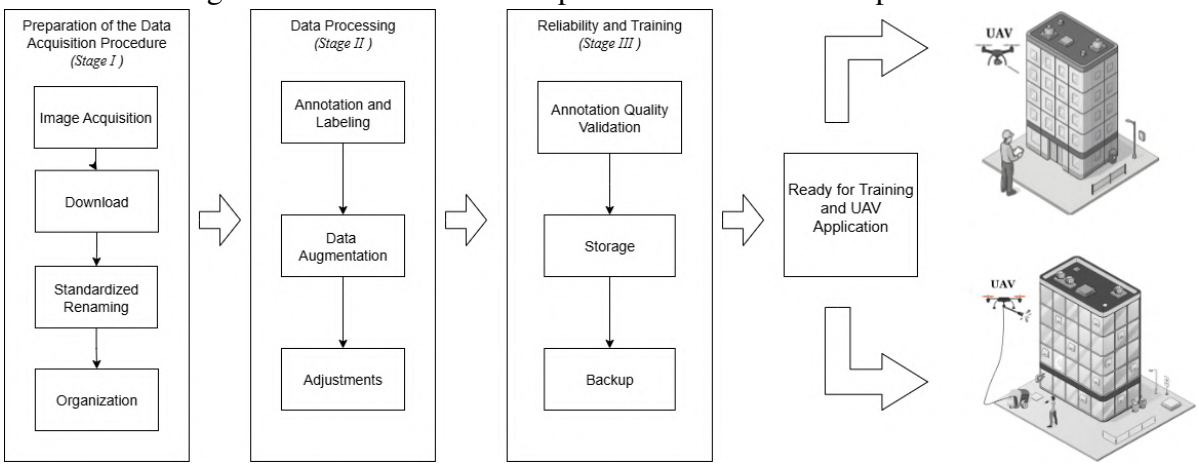
The development of this work consists of a workflow of 11 steps divided into three stages, which are detailed in the following.

- Stage I. Preparation of the Data Acquisition Procedure: The objectives, requirements, and planning for the image collection were defined. The images were manually collected through internet searches. After collection, the files were organized and renamed to facilitate processing.

- Stage II. Data Processing: In this stage, the dataset was carefully filtered to remove irrelevant images. The images were annotated to identify elements such as glass windows, doors, people, walls, pillars, and cars, using specialized tools. Before training, the images underwent preprocessing and annotation validation to ensure greater accuracy.

- Stage III. Reliability and Training: The descriptions were reviewed and documented in detail. The dataset was stored with backups in a secure system and prepared for use in machine learning models.

Figure 4.3: Workflow of the specialized dataset development.



Source: The Author

4.1.1 Stage I. Preparation of the Data Acquisition Procedure

Providing large datasets to train and test models is necessary to achieve success in AI learning (MARCINóW et al., 2024). The initial datasets should have controlled variation, as excessive diversity may hinder pattern identification due to the limited number of examples for each type. This could cause the model to interpret important features as “noise". However, it is crucial to ensure some variation in the dataset to prevent the model from becoming too specific and struggling to generalize to new examples.

1. Image Acquisition

In this step, the images were manually collected through specific searches on Google, DuckDuckGo, and other relevant websites such as ArchDaily and Freepik, playing a key role in the data selection. A total of 1,040 images were gathered and underwent an organization process, including renaming and annotation. Assuring data diversity and representativeness, a variety of keywords related to the topic of interest were used, such as “glass facades", “buildings with glass facades", and “glass windows".

2. Download

After the acquisition phase, the images were manually downloaded and organized into a dedicated project directory. During this process, each file was carefully examined to ensure it met the essential requirements for training, such as the presence of glass windows. Subsequently, the images were converted to a single format to optimize and standardize the subsequent processing stages.

3. Standardized Renaming

For the subsequent training of images in the neural network, it will be necessary to create a “*txt*” file containing the path of each image to be used in the training process. To optimize this procedure, the step of standardizing the image names was adopted, applying a uniform nomenclature by using a standard name followed by a sequential number, or just a sequential number, thus providing data consistency and organization.

4. Organization

By default, the neural network requires images to be organized in a structured manner for the training process. A widely adopted model for image classification involves segmenting the dataset into two main directories: “train” and “test”. The “train” directory contains the images used for training the model, enabling the extraction and learning of characteristic patterns for each class. The “test” directory consists of images intended for model validation, allowing performance evaluation and hyperparameter adjustment.

4.1.2 Stage II. Data Processing

In this stage, image annotation and labeling are performed, along with the necessary adjustments to ensure the quality and consistency of the dataset. Preprocessing is applied, including format standardization, resizing, and image normalization, ensuring they are properly prepared to maximize the model's performance during training.

5. Annotation and Labeling

Image annotation refers to the process of marking or highlighting specific regions within an image to provide detailed information about the objects present. This process is important in tasks such as object detection and segmentation, where it is necessary to precisely identify areas of interest. Examples of annotation include creating bounding boxes (rectangles around objects), polygons (shapes that outline objects more precisely), segmentation masks (where each pixel is labeled), and keypoints (such as joints in pose recognition).

On the other hand, image labeling involves assigning a category or class to the entire image. This process is widely used in classification tasks, where each image is associated with a specific category. Examples include binary classification, determining whether the image belongs to a category or not, multiclass classification, where the image is labeled in one of several possible classes, and multilabel classification, where multiple labels can be assigned to a single image, such as "window" and "door" at the same time).

The main difference between these tasks is that annotation marks specific parts of an image, while labeling assigns a class to the entire image. Both are crucial for providing neural network models with accurate and structured information to enhance learning and generalization during training. This process represents the most time consuming step in the creation of the dataset, as the annotations and labels are manually applied, image by image, requiring a high level of precision and attention to detail. For this dataset, each image was carefully annotated and labeled to identify glass windows, doors, people, walls, pillars, and cars. The annotation was carried out using specialized tools, such as LabelImg, generating annotation files compatible with the required format, such as TXT for YOLO.

6. Data Augmentation

Data augmentation is a technique widely used in machine learning tasks with the goal of artificially increasing the quantity and diversity of the available data. This approach aims to enhance the robustness of the model by improving its ability to handle variations

that occur in real-world scenarios, such as changes in lighting, position, scale, and the presence of noise, among others. In this work, rotation, translation, brightness shifts, and blur techniques were applied to enrich the dataset used for training to improve the robustness and generalization.

7. Adjustments

Some fundamental adjustments for neural network training include image resizing and normalization. Resizing ensures that all images have a consistent size, allowing the neural network to process them efficiently. This adjustment can be configured according to the network's specifications, with the input size being adapted to meet the model's requirements.

Normalization, in turn, adjusts the pixel values to a standard range, usually between 0 and 1 or -1 and 1. This process helps stabilize training, enabling the network to learn more quickly and efficiently. Although resizing and normalization can be automatically configured by the network, the user has the flexibility to adjust these parameters as needed to optimize the model's performance. This adjustment can be done considering the specific characteristics of the data and the project's requirements.

4.1.3 Stage III. Reliability and Training

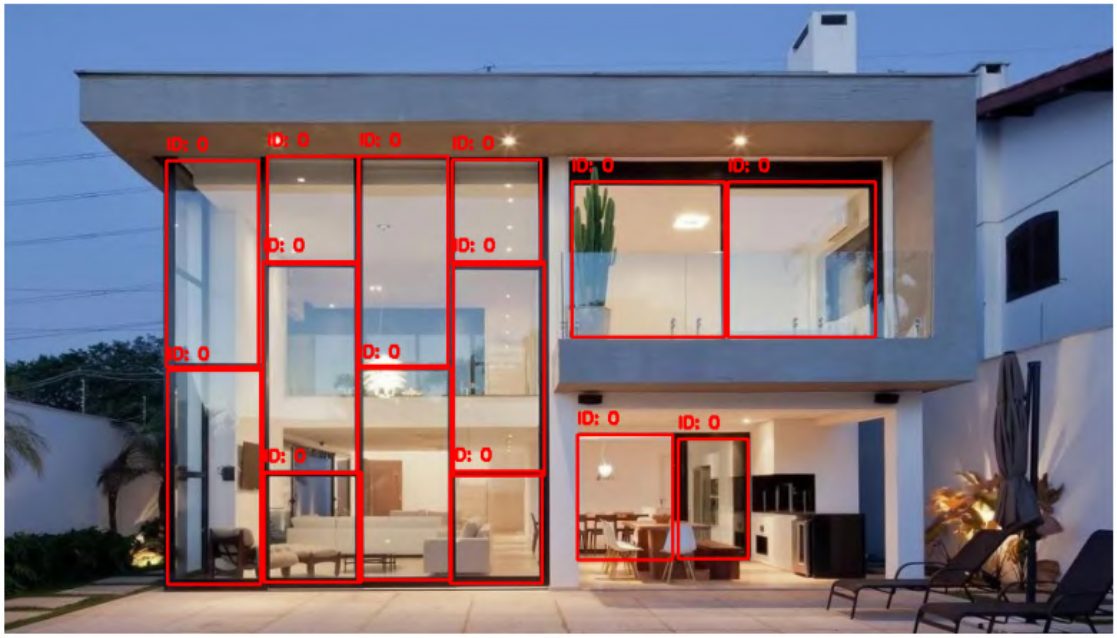
Inconsistent or inaccurate data can compromise the performance of the neural network, leading to less reliable predictions. To mitigate these risks, it is important to validate annotations, maintain organized storage, and ensure that the backup folder for weights is accessible. Additionally, reviewing the model's configurations, including hyperparameters and data partitioning, contributes to more efficient training and better generalization capability.

8. Annotation Quality Validation

To verify and validate the quality of the annotations, a Python algorithm was developed to process images and their corresponding label files in the YOLO format, enabling the visualization of bounding boxes over the images. The code begins by importing essential libraries, such as *cv2* for image processing and *matplotlib.pyplot* for graphical display.

The function plot bounding boxes takes as input the image paths and the label file. The image is loaded using *cv2.imread* and converted to the RGB format. Then, the label file is read, where each line contains information about the annotated objects, the class, and the normalized bounding box coordinates (center, width, and height).

Figure 4.4: Annotation Validation



Source: The Author

These coordinates are converted into absolute values based on the image dimensions, allowing the precise calculation of the bounding box vertices' positions. Each bounding box is drawn on the image using *cv2.rectangle*, while the class identifier is displayed next to it with *cv2.putText*. Finally, the annotated image is displayed using matplotlib, without axes, to facilitate the visual analysis of the annotations.

Figure 4.4 presents the validation of the labels using the developed algorithm, enabling a visual assessment of the annotation accuracy and the identification of potential adjustments to enhance the quality of the dataset.

9. Storage

The storage of labeled and annotated images should follow an organized structure to ensure data integrity and facilitate access during model training. Images are stored in structured directories, often divided into subsets such as training, validation, and testing. The annotation files, which contain information about bounding boxes and associated classes, are saved in the same directory as the images using standardized formats such as "txt" (YOLO).

To preserve data quality, it is recommended to use lossless image formats, such as PNG, or compressed formats, such as high quality JPEG. The file naming convention should be consistent and standardized, ensuring an exact match between each image and its corresponding annotation file. Storage can be managed locally or through cloud services, depending on project requirements. For large datasets, compression and indexing

techniques can be applied to optimize space usage and access time.

10. Backup

To ensure the integrity of the training process, it is important to create a dedicated backup folder where the model's weights will be regularly saved. This backup folder plays a key role in preserving training progress, allowing the model to be restored from the most recent training point in case of system failure or unexpected interruption. The backup folder also facilitates the analysis of the model's performance at different training stages. By saving the weights at various points, it becomes possible to compare the model's performance throughout the training process and adjust hyperparameters as needed.

11. Training

After preparing the dataset, which includes annotating and labeling the images, as well as configuring the essential files for training, such as the .data file (which contains the paths to the train, test, and backup folders), the ".cfg" file (which defines the network architecture), and the label files "train.txt" and "test.txt" (which list the training and validation images, respectively), the training process is initiated using the Darknet framework.

From the terminal, a command is executed, calling the ".data", ".cfg", and initial ".weights" files. During training, the model monitored in runtime, and parameters such as learning rate and batch size can be adjusted. The model weights are saved every 1000 iterations, ensuring continuous progress tracking. After this, the model is evaluated using the validation set, with mAP (mean Average Precision) a widely used metric to measure detection accuracy. The model can be used to perform inference on new images or videos, enabling efficient and accurate object detection.

Contributing to future research and supporting the scientific community, this work presents a dataset consisting of 3,805 labeled and annotated images, accompanied by a detailed analysis of the stages involved in its creation, combining real and synthetic data with techniques such as data augmentation. The dataset is publicly available and can be accessed at the following URL: <https://github.com/monymon/dataset_glasswindow>

4.2 Training of computer vision models

Commonly, the architecture of a CNN includes an input layer, convolutional layers, pooling layers, fully connected layers, and an output layer. These networks are used to extract sequences of feature vectors at multiple levels from the surface, capturing textures, edges, and structural information from cracks through an image processing approach.

Before the emergence of the YOLO family of models, object detection was predominantly performed using two stage approaches, such as R-CNN and Fast R-CNN, which represented fundamental milestones in the evolution of computer vision. These methods employed region proposal mechanisms to first generate candidate bounding boxes in an image, followed by the application of a classifier on those proposed regions (REDMON et al., 2016b).

In the specific case of R-CNN, this two stage paradigm was structured into three main modules: the first dedicated to generating category independent region proposals; the second consisting of a convolutional neural network responsible for extracting a fixed dimension feature vector for each region; and the third comprising a set of class specific support vector machines (SVMs) (GIRSHICK et al., 2014).

Although effective in terms of accuracy, such methods exhibited significant limitations regarding computational cost and processing speed, particularly in applications requiring real-time responses, such as mobile robotics, intelligent surveillance, and embedded systems. The need to operate distinct modules for region proposal and classification made the pipeline more complex and less efficient. In this context, the development of single stage detectors, exemplified by the YOLO family, emerged as a solution that combines architectural simplicity, high inference speed, and satisfactory performance, bringing about a substantial transformation in the field of object detection.

To implement the YOLO network with Darknet framework (YOLOv2, YOLOv3, YOLOv4 e YOLOv7), it was necessary to configure a training file that defines the essential parameters for optimizing processing and object detection. The input image dimensions are set as width=640 and height=640, providing an efficient balance between resolution and computational performance. The parameter channels=3 is configured to reflect the standard RGB image format.

The learning rate is specified as *learning rate=0.00261* and the number of images processed simultaneously per iteration is set as *batch size=128*, which influences the stability and efficiency of training. Additionally, subdivisions=16 determines the number of subdivisions of the batch, helping to adjust memory usage during training.

$$F = (N_c + 5) \cdot 3 \tag{4.1}$$

$$M_b = N_c \cdot 2000 \tag{4.2}$$

$$S = 80\%\ M_b\ ,\ 90\%\ M_b \tag{4.3}$$

Where M_b is the parameter for max batches, N_c is the number of classes, F is the number of filters, and S is the number of steps corresponding to 80% and 90% of the max batches. The number of filters in the convolutional layers is configured based on the number of classes. Parameters such as *max-batches* and steps are also defined based on the number of labeled classes. The value of *max-batches* represents the maximum number of training iterations, while steps define the learning rate adjustment points throughout the training process. According to the author (WANG; BOCHKOVSKIY; LIAO, 2023c), these parameters are fundamental and indispensable for training a new class in the network.

For implementation using the PyTorch framework (YOLOv5, YOLOv8, YOLOv9, YOLOv10, YOLOv11, YOLOv12), it was necessary to configure the data file, select the model to be used, and define the desired number of epochs for training. The remaining structure, such as the organization of the training, validation, and backup folders, is similar to that used in Darknet.

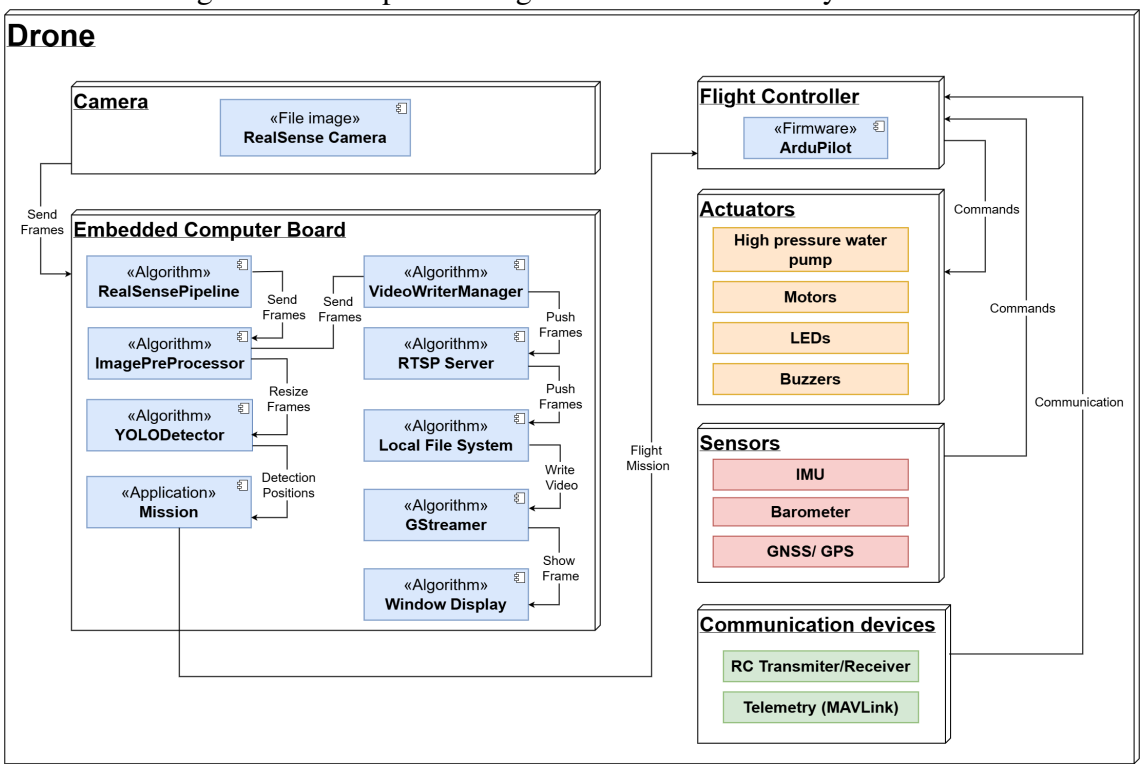
All labels are created using the LabelImg program, which allows for object selection and the creation of bounding boxes around them. Each bounding box is saved with the name of the corresponding class, forming the dataset necessary for training. Correct configuration of these parameters is significant for training an efficient and accurate model capable of performing object detection with high precision.

4.3 Design of the Embedded System

For drones to perform increasingly complex tasks in civil, industrial, and even military applications, it became necessary to equip them with greater perception, decision making capability, and autonomy. This is where embedded systems come into play, transforming the drone into an intelligent platform. These systems form the computational core of the aircraft, responsible for processing sensor data, executing navigation and control algorithms, and ensuring autonomous or semi-autonomous operation throughout the mission.

Figure 4.5 presents the component diagram of the embedded system integrated

Figure 4.5: Component diagram of the embedded system used.



Source: The Author

into the drone, in which the camera and the embedded computational board are integrated into the drone system, endowing it with intelligent capabilities, while the flight controller, actuators, sensors, and communication devices constitute native components of the unmanned aerial platform. The architecture is composed of an Intel RealSense camera, an ASUS Tinker Board 2S (ASUS, 2025), shown in the Figure 4.6, the CubeOrange flight controller, and an additional set of sensors responsible for the aircraft's navigation, detection, and approach.

The Skyclean drone employs an ASUS Tinker Board as its onboard processing unit. This board receives initialization commands and coordinates the entire autonomous operational cycle. Embedded within the system is an object detection model based on the YOLO architecture, pre-trained with optimized weights specifically tailored for identifying windows on building facades. The cleaning procedure follows the sequence described below:

1. After initial positioning in front of the facade, the algorithm performs real-time detection of windows within the field of view of the Intel RealSense camera mounted on the drone.
2. For each window identified and classified with confidence above the established

Figure 4.6: Tinker board and Camera Intel Real Sense



Source: The Author



Source: The Author

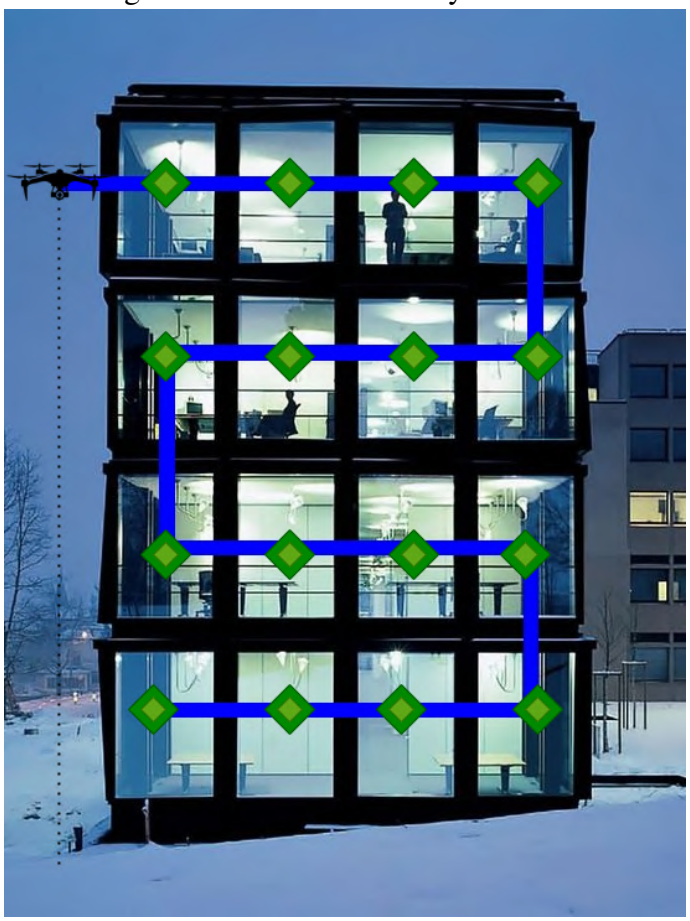
threshold, the drone executes the cleaning routine using a water jet pressurized at 200 bar.

3. Upon completion of the window cleaning, the drone performs a controlled lateral displacement to the right, exposing new regions of the facade to the camera's field of view.
4. If no additional windows are detected in the current frame, the drone descends vertically to the next facade level and repeats the scanning motion in a zigzag pattern, as shown in the figure 4.7, alternating the horizontal direction at each level.

Despite the high degree of autonomy enabled by onboard processing and neural network based detection, the system mandates continuous human supervision. This requirement arises from three primary factors: (i) the substantial cost of the equipment (approximately 100 to 150 thousand Brazilian reais, depending on the specific type of use); (ii) the safety risks posed to people and property in the event of system failure (approximately 15 kg, including water tank, high pressure pump, and reinforced structure); and (iii) aeronautical regulations governing the operation unmanned aircraft in urban environments (ANAC, 2025).

To address this need for supervision, the drone is equipped with a real-time telemetry and video transmission link. This channel enables the ground operator to continuously monitor the visual perspective captured by the RealSense camera, assess the quality of the

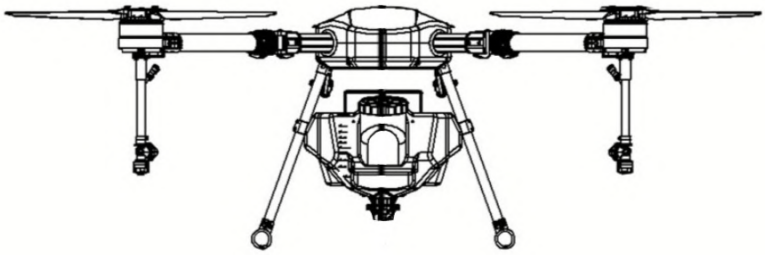
Figure 4.7: Path followed by the drone



Source: The Author

cleaning performed, and, when necessary, assume manual control of the vehicle in contingency situations.

Figure 4.8: SkyClean schematic drawing



Source: (SKYDRONES, 2025)

4.4 Skyclean design

This section is part of the SkyClean drone manual, provided by the company itself (SKYDRONES, 2025). The SkyClean is a proprietary drone, consisting of a remotely piloted aerial system equipped with four motors. Its onboard electronics, sensors, and control system stabilize the aircraft, enabling quick learning even for inexperienced pilots. The equipment was developed as part of the SkyDrones Solutions line, focusing on performing high pressure cleaning operations at height. Figure 4.8 shows the schematic drawing of the model.

4.4.1 Advantages

- Automatic stabilization and altitude hold;
- Robust equipment with high payload capacity;
- Compact and foldable structure, built with carbon fiber and electric propulsion;
- High customization capability, allowing adaptation to different types of operation.

4.4.2 Equipment Purpose

The SkyClean is used for high pressure cleaning operations at height. Attached to the drone body is a 120 cm lance with a special nozzle at the tip, which may be a Turbo model, fan type, or standard high pressure washer pattern. A 1/4" hose, 50 meters long, is also attached to the drone and connected to a high pressure pump located on the ground.

4.4.3 SkyClean Technical Data

The **SkyClean** solution is composed of two parts: the SkyClean drone and its components, and the ground solution with the pump and its peripherals.

SkyClean Drone

- 4 electric motors;
- Composite chassis (carbon fiber and aluminum);
- MEMS gyroscopes — 3 axes;
- Accelerometers — 3 axes;
- Magnetic sensor — 3 axes;
- 1 differential GNSS receiver (4 constellations);
- SkyDrones drawer system;
- Onboard 1080p camera with real-time transmission;
- Radio controller with built in screen and operating system running the SkyClean control application.

4.5 Equipment Dimensions

Table 4.1: SkyClean Dimensions

Component	**Specification**
Dimensions (arms open, without propellers)	140 cm × 140 cm × 60 cm
Dimensions (arms folded)	66 cm × 66 cm × 60 cm
Motor diameter	80 mm
Motor-to-motor distance	1393 mm
Weight (without battery)	11 kg

4.6 Simulation level Implementation

To pre-test the system implementation, a simulated environment was set up using various software and frameworks dedicated to drone simulation and programming. In this environment, Gazebo serves as a three dimensional physics simulator, enabling realistic reproduction of the scenario, sensors, and dynamic behavior of the unmanned aerial vehicle. Flight control is managed by PX4, a widely used open source autopilot firmware for unmanned aerial vehicles, which can operate in Software-in-the-Loop (SITL) mode to interact directly with Gazebo's virtual environment.

Communication between PX4 and the ROS 2 ecosystem is established through Micro XRCE-DDS, a lightweight implementation of the Data Distribution Service (DDS) protocol designed for embedded devices and distributed applications. ROS 2 Humble, in turn, provides the necessary middleware infrastructure for developing high level algorithms such as trajectory planning, computer vision, and autonomous control. The integration of these tools creates a robust and flexible platform for experimentation and validation of intelligent systems, allowing the drone's performance to be evaluated under controlled conditions before deployment on real hardware.

- Gazebo Garden: software for realistic simulation of vehicle control.
- PX4 v1.14: flight control software. This system is used both in simulations and on real drones.
- Micro-XRCE-DDS: topic translator between ROS and PX4.
- ROS 2 Humble: framework for programming flight commands.

Figure 4.9 illustrates the communication flow and integration between the different components of this architecture, highlighting the interaction among the simulator, the control system, and the perception modules based on deep learning.

Figure 4.9: Communication between the different software components



Source: The Author

This simulation stack is widely adopted in research and development projects in

the field of aerial robotics because it enables the validation of complex systems in a controlled and realistic environment before their deployment on physical vehicles. In this context, the simulated drone corresponds to the x500 model, equipped with an RGB camera both of which are provided as native resources by the simulator. This configuration provides a consistent foundation for evaluating system behavior, from data acquisition to the execution of control commands.

4.7 Implementation on a UAV

This section presents the pipeline that enables real-time visual detection performed by the drone, covering all stages from image acquisition to the generation of the final bounding boxes displayed on the operator's interface. Initially, the mathematical principles that characterize a digital image are explained, along with the structuring of the frames captured by the Intel RealSense camera in the spatial domain. Next, the post processing of the YOLO model output is detailed, including the conversion of predictions into real-world coordinates, the selection of the most likely class, the application of Non Maximum Suppression (NMS) to remove redundant detections, and finally, the visual rendering of the detections overlaid on the original frame.

4.7.1 Image Acquisition

To perform window detection, a video is generated in real time. This detection video is created from an ordered sequence of frames captured using an Intel RealSense camera. When these frames are played back at a specific rate (frames per second), they produce a continuous motion sequence, resulting in an animated representation of the acquired scene. Each frame consists of three channels in the visible light spectrum: red, green, and blue. The sensor captures the image by obtaining the intensity of the light for the respective spectrum channels at the same time t (BASSO; FREITAS, 2020).

An image may be defined as a two dimensional function, $f(x,y)$, where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x,y) is called the intensity or gray level of the image at that point (GONZALEZ; WOODS, 1992).

It can be represented computationally by an $M \times N$ matrix $A_{M\times N}$, where each

element $a_{i,j} \in N$ (or a subset of the natural numbers) represents the intensity or shade of a color (GONZALEZ; WOODS, 1992). In mathematical notation:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix}$$

The section of the real plane spanned by the coordinates of an image is called the spatial domain, with x and y being referred to as spatial variables or spatial coordinates (GONZALEZ; WOODS, 1992). The frames used for detection are all in the spatial domain.

4.7.2 Mathematical foundation

The post processing pipeline of the YOLO (You Only Look Once) model outputs consists of a sequence of mathematical operations that transform the tensors produced by the convolutional neural network into final object detections expressed in the two dimensional coordinate space of the input image. The YOLO paradigm relies on direct regression of bounding boxes and class probabilities from a regular grid of cells, eliminating the explicit region proposal stage characteristic of two stage approaches.

Below is presented a formal and mathematically rigorous description of the main stages of the post processing pipeline, covering everything from the interpretation of the network's raw predictions to the application of Non Maximum Suppression (NMS) and the obtention of the final detections.

4.7.2.1 Bounding Box Generation by YOLO

During training, YOLO divides the image into a spatial grid of cells. For each cell, the network predicts B bounding boxes, each defined by the vector:

$$(\hat{x}, \hat{y}, \hat{w}, \hat{h}, \hat{p}, c_1, c_2, \ldots, c_K),$$

where $(\hat{x}, \hat{y})$ are the coordinates of the bounding box center, $\hat{w}$ and $\hat{h}$ are the width and height, $\hat{p}$ is the confidence score (objectness), and c_k is the probability of class k. The predicted values are normalized with respect to the image dimensions. Mathematically,

the conversion from the network's raw output to real (denormalized) bounding box coordinates is given by:

$$x = \hat{x}W, \qquad y = \hat{y}H,$$

$$w = \hat{w}W, \qquad h = \hat{h}H,$$

where (W,H) are the width and height of the original image. Since the model uses the *center-width-height* format, but graphical display requires the *top-left-width-height* (x-min, y-min, width, height) format, the following conversion is performed:

$$x_{\text{left}} = x - \frac{w}{2}, \qquad y_{\text{top}} = y - \frac{h}{2}.$$

Thus, the final bounding box is represented by:

$$\text{bbox} = (x_{\text{left}}, y_{\text{top}}, w, h).$$

4.7.2.2 Class Selection

The class probability vector is

$$\mathbf{c} = (c_1, c_2, \ldots, c_K).$$

The class with the highest value is selected:

$$c_{\max} = \max_k c_k, \qquad k = \arg\max_k c_k.$$

The detection is kept if:

$$\hat{p} > \tau_1 \quad \text{e} \quad c_{\max} > \tau_2,$$

where τ_1 and τ_2 are confidence thresholds (in this case, 0.7).

4.7.2.3 Non Maximum Suppression (NMS)

After all valid boxes are generated, the model may produce multiple boxes for the same object. The Non Maximum Suppression (NMS) procedure removes duplicates while keeping only the best one.

Given the initial set of bounding boxes $\{b_1, b_2, \ldots, b_N\}$ with their respective con-

fidence scores $\{s_1, s_2, \ldots, s_N\}$, where:

$$s_i = \text{final score of the box } b_i$$

This final score is typically:

$$s_i = \hat{p} \cdot c_{\max}$$

the algorithm works as follows:

1. The boxes are sorted in descending order of confidence:

$$s_{i_1} \geq s_{i_2} \geq \cdots \geq s_{i_N}.$$

2. The box with the highest confidence, b_{i_1}, is selected and added to the final set of detections.
3. For each remaining box b_j, the Intersection over Union (IoU) is computed:

$$\text{IoU}(b_{i_1}, b_j) = \frac{\text{Área}(b_{i_1} \cap b_j)}{\text{Área}(b_{i_1}) + \text{Área}(b_j) - \text{Área}(b_{i_1} \cap b_j)}.$$

4. All boxes b_j such that:

$$\text{IoU}(b_{i_1}, b_j) > \tau_{\text{NMS}},$$

where τ_{NMS} is the suppression threshold (in this work, 0.4 is used).

5. The process is repeated with the next non removed box.

In the end, the resulting set contains only the most representative boxes, eliminating unnecessary overlaps.

4.7.2.4 Visual Display of Detections

After NMS, each detection is represented by:

$$(b, k, s) = \left((x_{\text{left}}, y_{\text{top}}, w, h),\ k,\ s\right),$$

where k is the final class and s is the confidence score.

The visual display consists of drawing, on the original image, a rectangle corre-

sponding to the bounding box:

$$\text{Rectangle}(x_{\text{left}}, y_{\text{top}}, w, h).$$

Additionally, a label containing the class name and the confidence score is displayed:

$$\text{label} = \text{Classe}(k^s).$$

In OpenCV, these operations are performed using functions such as `rectangle()` and `putText()`, but mathematically they are equivalent to overlaying a geometric shape defined by the edges: $(x_{\text{left}}, y_{\text{top}})$, $(x_{\text{left}} + w, y_{\text{top}})$, $(x_{\text{left}}, y_{\text{top}} + h)$, $(x_{\text{left}} + w, y_{\text{top}} + h)$. The final result is an annotated image containing all the filtered and labeled bounding boxes, representing the visual prediction of the YOLO model.

4.8 System Architecture

The system architecture considered in this work is organized into three parts: (i) the hardware architecture used for training the YOLO model, (ii) the hardware architecture of the embedded system, and (iii) the software architecture responsible for the deployment of the proposed solution on the embedded system.

4.8.1 Hardware Architecture for Training

The training experiments were conducted on a system equipped with an NVIDIA GV100 (TITAN V) GPU 12GB RAM and an Intel® Core™ i5-9400 CPU (2.90 GHz, 6 cores), running Ubuntu 22.04.4 LTS. This hardware configuration provided sufficient computational performance to support efficient and stable model training.

4.8.2 Hardware Architecture of the Embedded System

For the implementation of the proposed embedded system, the ASUS Tinker Board 2S single board computer was employed as the main processing platform, responsible for executing the control algorithm and vision processing tasks. The unmanned aerial vehicle used was the SkyClean drone, equipped with the CubeOrange flight controller

(CubePilot standard). Image acquisition and depth data were obtained through an Intel RealSense camera, configured to operate in stereo mode.

4.8.3 Software Architecture

Structural clarity and scalability are fundamental principles for building robust systems. Scalability has been regarded as one of the most important properties of a software design, significantly impacting many external quality attributes such as reusability, maintainability, and understandability (XIANG et al., 2019).

The software architecture, represented in a UML class diagram in Figure 4.10, follows a compositional pattern centered on the MainApp class, which acts as the application coordinator. This class orchestrates the entire application flow, composing and coordinating four modules.

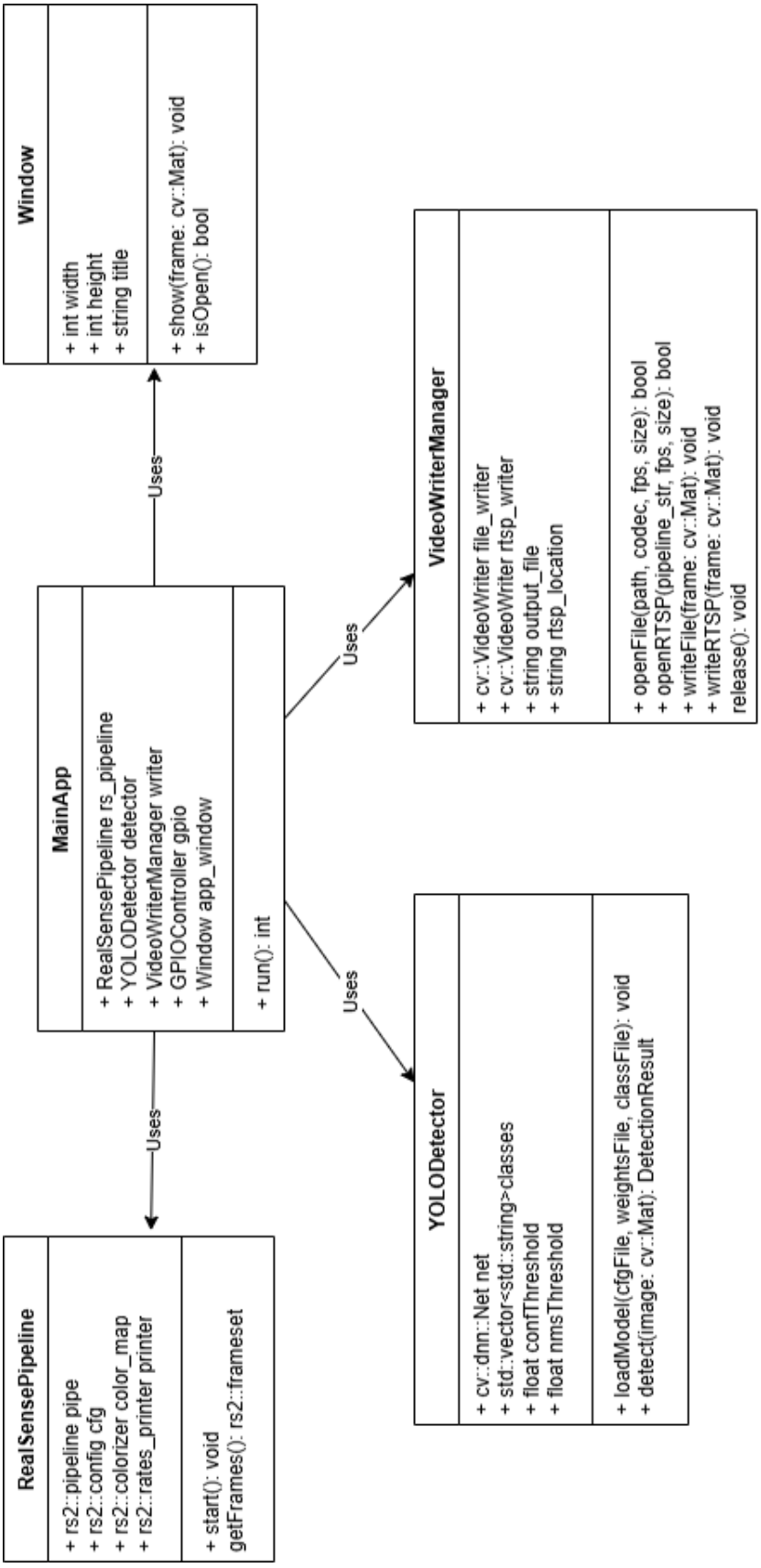
1. **RealSensePipeline**
   Responsible for the complete abstraction of the Intel RealSense device. Encapsulates the initialization of the librealsense pipeline, configuration of streams (aligned color and depth), the depth map colorizer, and frame rate monitoring utilities for performance diagnostics. Exposes a minimalist interface consisting of the methods $start()$ and $get_frames()$, which return synchronized pairs of color and depth frames in OpenCV format (cv::Mat).
2. **YOLODetector**
   Module dedicated to object detection based on the YOLO (You Only Look Once) family. Manages the loading of the model (.cfg, .weights, and .names files), configuration of the cv::dnn::Net network with CUDA or OpenVINO backend when available, and inference hyperparameters (confidence threshold and non maximum suppression threshold). Provides the $detect()$ method, which receives an RGB frame and returns a structured vector containing bounding boxes, class indices, and confidence scores.
3. **VideoWriterManager**
   Responsible for video persistence and dissemination. Maintains two independent destinations: (i) local file recording (via cv::VideoWriter with H.264/H.265 codec) and (ii) RTSP/RTP streaming using GStreamer or ffmpeg as optional backends. The public interface allows opening/closing destinations asynchronously, queuing

frames for recording, and safely releasing resources upon application termination.

4. **Window**

   Real-time visualization layer based on OpenCV HighGUI. Encapsulates window title, dimensions, and flags, providing the $show()$ method for rendering annotated frames and $is-open()$ to check the graphical interface state.

Figure 4.10: Diagram Class



Source: The Author

5 RESULTS AND EVALUATION

In the context of Machine Learning, the main objective is to use previously collected data to estimate or predict an outcome. When this outcome is associated with distinct categories, the problem is characterized as a classification task (GRANDINI; BAGLI; VISANI, 2020) (MORTAZ, 2020).

5.1 Training dataset with YOLO versions

Aiming to provide a system solution with improved performance, this study compared the performance of different versions of the YOLO applied to a dataset specifically developed for this research, focusing on the detection of glass windows in modern buildings' facades. Two experiments were conducted: the first using exclusively real images and the second combining real and synthetic images generated with Stable Diffusion. The research sought to identify the most suitable YOLO version for this task and to evaluate the contribution of synthetic data, given the limited availability of datasets targeting this specific type of window architecture in the literature.

Ten of the twelve existing YOLO versions were evaluated, using the Darknet framework (for YOLOv2, YOLOv3, YOLOv4, and YOLOv7) and the PyTorch framework (for the remaining versions). The YOLOv1 and YOLOv6 versions were excluded due to the obsolescence of the former and software incompatibilities related to dependency installation and training execution for the latter. Only implementations with simplified and standardized training processes were utilized. Experimental configurations were kept constant across all training processes, including the same number of images for training and testing, a resolution of 640×640 pixels, a batch size of 16, and a learning rate of 0.00261. All training processes were conducted via Command Line Interface (CLI).

Table 5.1: YOLOv2 RD

Class	Precision	Recall	mAP50
Glass Window	0.301	0.4	40.01
Door	0.309	0.4	43.75
Person	0.083	0.0	5.32
Wall	0.273	0.2	13.21
Pillar	0.075	0.0	33.63
Car	0.318	0.4	37.76

Source: The Author

Table 5.2: YOLOv2 RSD

Class	Precision	Recall	mAP50
Glass Window	0.473	0.6	41.70
Door	0.263	0.2	10.71
Person	0.118	0.1	6.18
Wall	0.111	0.0	3.94
Pillar	0.174	0.3	33.17
Car	0.429	0.5	26.58

Source: The Author

Table 5.3: YOLOv3 RD

Class	Precision	Recall	mAP50 (%)
Glass Window	0.626	0.4	57.82
Door	0.741	0.4	57.31
Person	0.858	0.2	44.98
Wall	0.375	0.2	16.40
Pillar	0.308	-	37.71
Car	0.800	0.8	81.71

Source: The Author

Table 5.4: YOLOv3 RSD

Class	Precision	Recall	mAP50
Glass Window	0.736	0.7	69.06
Door	0.667	0.2	24.64
Person	0.833	0.3	30.45
Wall	-	-	6.11
Pillar	0.500	0.1	7.88
Car	0.875	0.3	85.41

Source: The Author

Table 5.5: YOLOv4 RD

Class	Precision	Recall	mAP50
Glass Window	0.570	0.4	64.44
Door	0.525	0.5	58.15
Person	0.833	0.2	54.46
Wall	0.250	0.2	17.77
Pillar	0.273	0.1	61.00
Car	0.800	0.1	96.94

Source: The Author

Table 5.6: YOLOv4 RSD

Class	Precision	Recall	mAP50
Glass Window	0.756	0.8	72.83
Door	0.385	0.2	21.14
Person	0.563	0.6	41.63
Wall	-	-	3.21
Pillar	-	-	1.92
Car	0.895	0.2	71.63

Source: The Author

Table 5.7: YOLOv5 RD

Class	Precision	Recall	mAP50
Glass Window	0.685	0.609	64.80
Door	0.764	0.553	64.10
Person	0.907	0.667	70.90
Wall	0.532	0.238	21.80
Pillar	0.796	0.556	56.00
Car	0.871	0.882	93.20

Source: The Author

Table 5.8: YOLOv5 RSD

Class	Precision	Recall	mAP50
Glass Window	0.844	0.700	76.10
Door	0.563	0.250	29.00
Person	0.692	0.500	50.30
Wall	0.701	0.296	23.90
Pillar	0.882	0.250	28.80
Car	0.764	0.719	78.60

Source: The Author

Table 5.9: YOLOv7 RD

Class	Precision	Recall	mAP50
Glass Window	0.463	0.3	47.50
Door	0.531	0.4	52.30
Person	0.667	0.1	29.70
Wall	0.333	0.3	20.60
Pillar	-	-	04.20
Car	0.778	0.9	80.90

Source: The Author

Table 5.10: YOLOv7 RSD

Class	Precision	Recall	mAP50
Glass Window	0.672	0.7	64.30
Door	0.429	0.2	14.60
Person	-	-	01.00
Wall	-	-	05.20
Pillar	0.250	0.1	08.60
Car	0.600	0.5	33.10

Source: The Author

Table 5.11: YOLOv8 RD

Class	Precision	Recall	mAP50
Glass Window	0.683	0.611	67.50
Door	0.778	0.658	68.70
Person	1.000	0.405	60.40
Wall	0.660	0.238	26.00
Pillar	0.544	0.556	52.30
Car	0.852	0.680	76.20

Source: The Author

Table 5.12: YOLOv8 RSD

Class	Precision	Recall	mAP50
Glass Window	0.798	0.690	76.80
Door	0.484	0.369	40.00
Person	0.658	0.200	32.90
Wall	0.248	0.124	07.90
Pillar	0.546	0.417	47.70
Car	0.753	0.556	64.40

Source: The Author

Table 5.13: YOLOv9 RD

Class	Precision	Recall	mAP50
Glass Window	0.719	0.589	68.40
Door	0.752	0.579	69.90
Person	1.000	0.438	52.40
Wall	0.551	0.238	24.40
Pillar	0.772	0.556	59.00
Car	0.821	0.540	75.70

Source: The Author

Table 5.14: YOLOv9 RSD

Class	Precision	Recall	mAP50
Glass Window	0.772	0.739	75.80
Door	0.388	0.357	35.50
Person	0.649	0.050	15.50
Wall	0.103	0.125	09.08
Pillar	0.496	0.417	37.00
Car	0.391	0.556	46.20

Source: The Author

Table 5.15: YOLOv10 RD

Class	Precision	Recall	mAP50
Glass Window	0.675	0.605	65.90
Door	0.763	0.447	59.70
Person	0.974	0.238	31.30
Wall	0.571	0.143	19.90
Pillar	0.588	0.667	66.30
Car	0.783	0.471	57.20

Source: The Author

Table 5.16: YOLOv10 RSD

Class	Precision	Recall	mAP50
Glass Window	0.683	0.789	76.70
Door	0.260	0.393	23.30
Person	1.000	0.000	02.26
Wall	0.518	0.125	19.10
Pillar	0.165	0.333	25.00
Car	0.126	0.444	13.10

Source: The Author

Table 5.17: YOLOv11 RD

Class	Precision	Recall	mAP50
Glass Window	0.745	0.590	67.90
Door	0.805	0.579	69.40
Person	1.000	0.371	56.60
Wall	0.565	0.190	24.90
Pillar	0.929	0.556	55.90
Car	0.742	0.529	74.70

Source: The Author

Table 5.18: YOLOv11 RSD

Class	Precision	Recall	mAP50
Glass Window	0.760	0.728	77.70
Door	0.538	0.458	49.20
Person	0.933	0.150	19.30
Wall	0.0747	0.0625	12.50
Pillar	0.538	0.500	47.60
Car	0.514	0.556	52.70

Source: The Author

Table 5.19: YOLOv12 RD

Class	Precision	Recall	mAP50
Glass Window	0.645	0.648	68.30
Door	0.611	0.579	65.20
Person	0.865	0.524	61.40
Wall	0.505	0.190	20.50
Pillar	0.778	0.778	82.30
Car	0.701	0.706	74.40

Source: The Author

Table 5.20: YOLOv12 RSD

Class	Precision	Recall	mAP50
Glass Window	0.741	0.796	80.00
Door	0.403	0.531	43.20
Person	0.713	0.350	45.70
Wall	0.156	0.375	11.90
Pillar	0.375	0.500	48.20
Car	0.679	0.589	62.80

Source: The Author

For the comparative analysis of the results, the tables were arranged side by side, providing a clearer visualization of the models performance. In this presentation, RD refers to the real data dataset, whereas RSD denotes the dataset composed of real and synthetic data. Tables 5.1, 5.3, 5.5, 5.7, 5.9, 5.11, 5.15, 5.17, and 5.19 present the results obtained with a dataset composed exclusively of real images, while Tables 5.2, 5.4, 5.6, 5.8, 5.10, 5.12, 5.16, 5.18, and 5.20 refer to the results of training with a combination of real and synthetic data generated with Stable Diffusion. Based on the graphs in the Figure 5.1 and 5.2, it is possible to evaluate the performance of the models for the "Glass Window" class, the focus of this study, more clearly.

5.1.1 Results Analysis for YOLOv2

Building on this analysis, the results presented in Tables 5.1 and 5.2 refer to the performance of the YOLOv2 architecture, as depicted in Figure 2.7. The network is structured with a convolutional backbone responsible for feature extraction, followed by a passthrough module that reorganizes and concatenates feature maps from deeper and shallower layers. This strategy enables the integration of semantic information with spatial details, which is essential for accurate object detection. The final convolutional layers refine these representations to produce predictions in the desired format, including bounding box coordinates, confidence scores, and class probabilities.

The precision reflects the ability of the final layers to correctly classify detected objects, although it is a result of the interaction across the entire network. Recall, in turn, is closely tied to the network's sensitivity in identifying all relevant instances, with

Figure 5.1: Performance of YOLO for class Glass Window

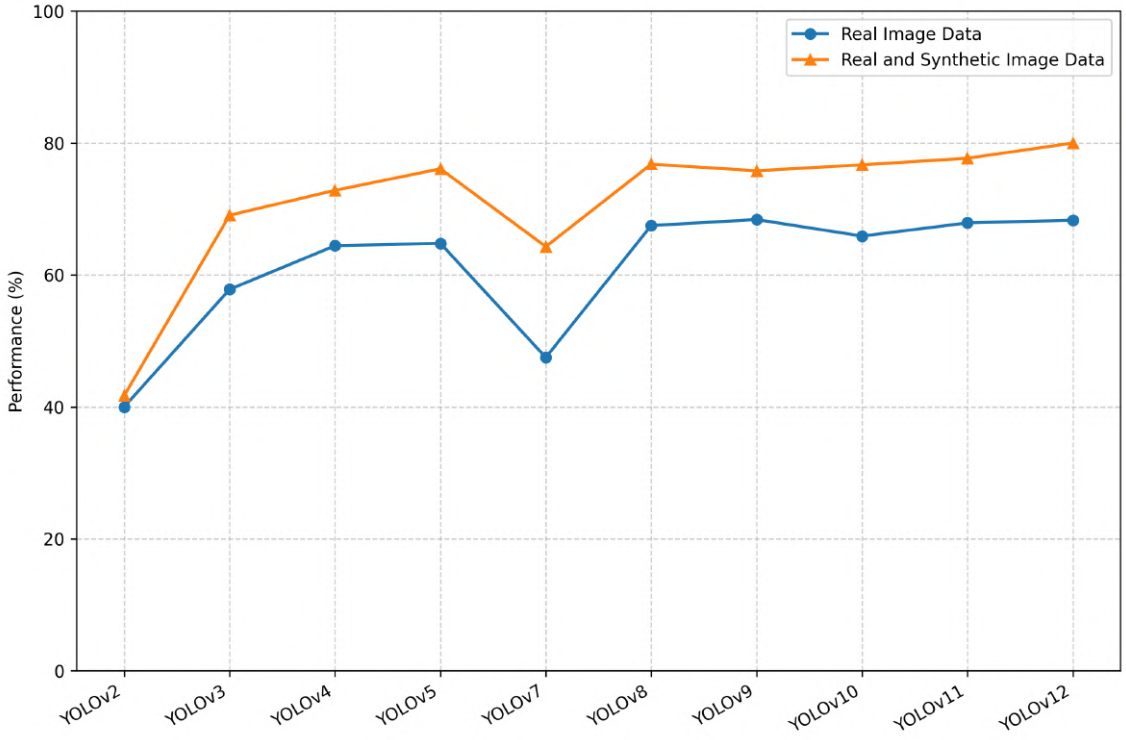
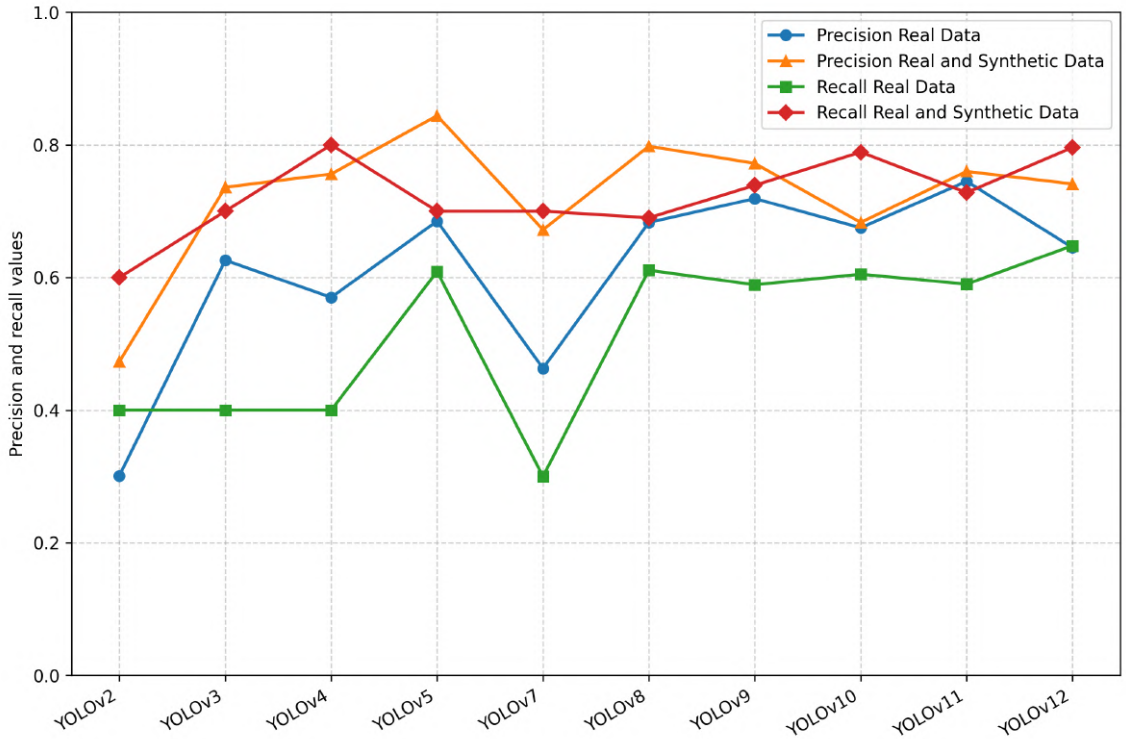


Figure 5.2: Precision and Recall for class Glass Window



the passthrough module being particularly influential, especially in detecting small or partially visible objects. The mAP50 metric (mean Average Precision at IoU=0.5) encapsulates the model's overall effectiveness in integrating the stages of feature extraction, concatenation, and classification, providing a comprehensive view of performance.

For the network under analysis, the results demonstrate that classes such as Glass Window, with a precision of 0.301, recall of 0.4, and mAP50 of 40% on the real data (RD) dataset, indicate that the integration of the convolutional backbone with the passthrough module was effective in generating discriminative representations for these categories. However, the inclusion of synthetic data did not lead to significant performance improvements.

5.1.2 Results Analysis for YOLOv3

The YOLOv3 architecture, illustrated in Figure 2.8, has its results presented in Tables 5.3 and 5.4. The network employs a deeper backbone, Darknet-53, combined with the Spatial Pyramid Pooling (SPP) module and the Feature Pyramid Network (FPN), significantly enhancing its ability to extract rich representations and perform detections across multiple scales.

In the analysis of the YOLOv3 architecture, significant improvements are observed compared to previous versions. The results demonstrate a notable increase in mAP50 for specific classes, such as Glass Window, which rose from 57.82% on the real data dataset to 69.06% with the inclusion of synthetic data in the dataset, alongside an increase in precision from 0.62 to 0.73. Similarly, the Car class, although not the primary focus of this study, showed an improvement from 81.71% to 85.41%. These metrics suggest that the optimizations introduced in the YOLOv3 architecture yield considerable advancements compared to the previous version, YOLOv2, where no significant improvements were observed across different datasets. However, the findings underscore that the quality and composition of the training dataset remain critical factors for the effectiveness of object detection performance.

5.1.3 Results Analysis for YOLOv4

The YOLOv4 architecture, shown in Figure 2.9, has its results displayed in Tables 5.5 and 5.6. The CNN backbone is responsible for extracting image features, learning filters that capture edges, textures, shapes, and patterns specific to each object. Next, the SPP (Spatial Pyramid Pooling) combines information from multiple scales, enabling the network to detect objects of different sizes, especially small ones or those with significant size variations. The PAN (Path Aggregation Network) performs feature fusion from different network levels, coming from the backbone and SPP, improving object localization and classification. Finally, the prediction head converts these processed features into bounding boxes, classes, and scores, generating the final output that represents each grid cell with its predictions.

By correlating this architecture with the results obtained on datasets, it is possible to better understand the network's strengths and limitations. On the RD dataset, for example, the Glass Window class achieved a precision of 0.57, recall of 0.4, and mAP50 of 64.44%. The low recall indicates that the network failed to detect all windows present, possibly because the backbone and SPP did not capture sufficient features. The moderate precision shows that when the network makes a prediction, it is correct most of the time, reflecting that the backbone learned discriminative features, although overall sensitivity is limited. On the RSD dataset, a general improvement is observed. For instance, Glass Window achieved a precision of 0.756 and recall of 0.8 and mAP50 of 78.83%, indicating that the SPP and PAN were able to aggregate features from multiple scales and levels more efficiently, increasing both sensitivity and precision.

5.1.4 Results Analysis for YOLOv5

The architecture of YOLOv5, represented in Figure 2.9, has its results presented in the Tables 5.7 and 5.8. The Backbone, comprising BottleNeckCSP modules and the Spatial Pyramid Pooling (SPP) block, is responsible for the initial extraction of visual features, capturing elements such as edges, textures, shapes, and patterns that enable object differentiation. The SPP, in particular, agregates information across multiple scales, enhancing the detection capability for objects of varying sizes, including small instances. The PANet, in turn, fuses features from different levels of the Backbone and SPP, improving object localization and increasing classification robustness. Finally, the prediction

Head, implemented with 1×1 convolutional layers, converts the processed features into bounding boxes, classes, and confidence scores, generating the model's final predictions.

In the combined real and synthetic data RSD dataset, a significantly improved performance was observed for the Glass Window class, with a precision of 0.844, a recall of 0.700, and an mAP50 of 76.10%, surpassing the results obtained in the real data RD dataset. This improvement suggests that the SPP and PANet were more effective in integrating multi-scale information, resulting in greater sensitivity for detecting glass windows while maintaining high classification precision, thereby reducing both false positives and false negatives.

5.1.5 Results Analysis for YOLOv7

The YOLOv7 architecture enhances object detection through a CNN backbone for feature extraction, presented in Figure 2.10, has its results presented in the Tables 5.9 and 5.10, a Spatial Pyramid Pooling (SPP) module for multi-scale feature aggregation, and a Path Aggregation Network (PAN) for feature fusion, improving localization and classification accuracy. Additionally, the Label Assignment Module (LVM) optimizes label assignment during training, contributing to model stability and robustness. Finally, the prediction head converts the processed features into bounding boxes, classes, and confidence scores, generating the final detection output.

Analyzing the performance of the YOLOv7 architecture across different datasets provides insights into the impact of each component. In the RD dataset, the Glass Window class achieved a precision of 0.463, a recall of 0.3, and an mAP50 of 47.50%. The low recall suggests that the model struggled to detect a significant number of instances of this class, likely due to the visual complexity of glass windows, characterized by reflections, transparency, and limited texture. This indicates that the backbone and SPP faced challenges in extracting consistent visual patterns for this class. However, the precision demonstrates that when detections occurred, they were relatively reliable, suggesting that some discriminative features were learned, albeit with limited generalization.

In RSD dataset, a significant improvement in performance is observed, particularly for the Glass Window class, which achieved a precision of 0.672, a recall of 0.7, and an mAP50 of 64.30%. These results reflect the positive impact of incorporating synthetic data, which provided greater diversity of examples, enabling the backbone to extract more robust and consistent patterns.

5.1.6 Results Analysis for YOLOv8

The YOLOv8 architecture, which integrates modules for efficient object detection, as shown in Figure 2.11, has its results presented in the Tables 5.11 and 5.12. The convolutional backbone extracts visual features like edges and textures. The Dense Connection Block enhances information flow, reducing detail loss. The Spatial Pyramid Pooling (SPP) block aggregates multi-scale features, improving robustness for objects of varying sizes, such as glass windows. The prediction block generates bounding boxes, classes, and confidence scores for final predictions.

In RD dataset, the Glass Window class achieved a precision of 0.683, a recall of 0.611, and an mAP50 of 67.50%. The moderate recall suggests that the model failed to identify all window instances, possibly due to limitations in the backbone and SPP in capturing scale variations or the Dense Connection Block's inadequacy in preserving critical details.

In the combined real and synthetic data RSD dataset, a significant improvement in the performance of the Glass Window class was observed, with a precision of 0.798, a recall of 0.690, and an mAP50 of 76.80%. These results indicate that the SPP and dense connections operated more efficiently, enabling the detection of window instances across different scales with greater reliability. The increase in recall demonstrates a reduction in false negatives, indicating that more windows were correctly identified. Meanwhile, the improved precision suggests that the model was able to better distinguish windows from other classes, such as doors or walls, with greater effectiveness, minimizing false positives.

5.1.7 Results Analysis for YOLOv9

Following the pipeline of previous versions, YOLOv9, Shown in Figure 2.12, describes on the results detailed in Tables 5.13 and 5.14, in this version introduces the PGI (Progressive Gradient Information) module, which refines gradients and enhances learning in deeper layers, reducing information loss before the output layer generates final predictions for bounding boxes and classes.

In results with RD dataset, the glass window class showed excellent performance, with an mAP50 of 68.40%, but was still limited by a recall of 0.589. This indicates that the CNN and SPP effectively extract features of this object, but some windows may have

occlusions, poor lighting, or challenging angles, making detection more difficult. With the introduction of synthetic data, significant improvements were observed across most classes. The "glass window" class achieved an mAP50 of 75.80% and a recall of 0.739, demonstrating that the use of synthetic data helped the PAN and PGI modules generalize better, recognizing windows in diverse contexts.

5.1.8 Results Analysis for YOLOv10

The YOLOv10 architecture, Figure 2.13 has it results summarized in Tables 5.15 and 5.16, uses a CNN backbone to extract features, SPP for multi-scale robustness, PSA for contextual attention, and PAN for feature integration and localization. Prediction heads balance hypothesis generation and redundancy, producing final bounding boxes, classes, and scores.

The analysis of the Glass Window class demonstrates that the integration of the SPP, PAN, and PSA modules in the YOLOv10 architecture is highly effective. On the RD dataset, the performance achieves an mAP50 of 65.90%, with a precision of 0.67 and a recall of 0.60. However, on the RSD dataset, a significant improvement is observed, reaching an mAP50 of 76.70%, accompanied by an increased recall from 0.605 to 0.789. This enhancement is attributed to the regular and standardized shapes of windows, which benefit from the multi-scale analysis provided by the SPP module, the feature integration across different levels performed by the PAN, and the contextual attention of the PSA, which aids in distinguishing windows from walls.

5.1.9 Results Analysis for YOLOv11

The YOLOv11 architecture, illustrated in Figure 2.14 has its results presented in Tables 5.17 and 5.18, optimizes object detection with a hierarchical structure. The Backbone, using convolutional layers and residual blocks like Conv and C3K2, extracts key features such as edges, textures, and shapes. The SPPF and C2PSA modules refine and fuse features across multiple scales and levels, enabling effective detection of objects with varying sizes and positions.

Analysis of YOLOv11 performance on the RD and RSD datasets provides insights into its capabilities and limitations. On the RD dataset, the Glass Window class achieved

an excellent precision of 0.745 compared to previous models, with a recall of 0.590 and an mAP50 of 67.90%. These results suggest that, while the model effectively identifies discriminative features, challenges remain in detecting objects with complex variations, possibly due to limitations in the Backbone's generalization or feature fusion in specific scenarios.

In contrast, on the RSD dataset, the Glass Window class showed slightly better performance, achieving a precision of 0.75, a recall of 0.80, and a significantly higher mAP50 of 77.70%. These results highlight the effectiveness of the Backbone, as well as the importance of the SPPF and C2PSA modules in multi-scale feature fusion, which enhance the model's ability to detect objects across different dimensions and positions. The improvement observed in the RSD dataset underscores the robustness of the YOLOv11 architecture in scenarios with more consistent visual characteristics, reinforcing its suitability for applications requiring precise detection in varied contexts.

5.1.10 Results Analysis for YOLOv12

The architecture of YOLOv12, demonstrated in Figure 2.15 has its results presented in Tables 5.19 and 5.20. The Backbone, comprising convolutional layers and R-ELAN (Reinforced Efficient Layer Aggregation Network) modules, is responsible for the initial capture of visual features, such as textures, edges, and patterns, extracted directly from input images. The Neck, incorporating R-ELAN+A2 blocks combined with concatenation and upsampling operations, facilitates the integration of information from different levels of granularity, enhancing the network's ability to combine contextual and spatial details. Finally, the Head, equipped with Flush Attention layers, refines the final predictions by dynamically adjusting the weights of the most relevant features, optimizing the generation of bounding boxes, classes, and confidence scores.

In the combined RSD dataset, the Glass Window class achieved the best performance among all evaluated categories and versions, with an mAP50 of 80.00% and a recall of 0.796. These results highlight YOLOv12 capability to detect objects with well defined contours and regular structures, such as glass windows. The findings suggest that the integration of R-ELAN+A2 blocks in the Neck and the attention mechanism in the Head were particularly effective for detecting objects with clear, repetitive, and highly structured characteristics, such as glass windows. Thus, the YOLOv12 architecture demonstrates robustness in identifying objects with well defined visual patterns, establishing its

effectiveness in detection tasks requiring high precision in modern architectural contexts.

5.1.11 YOLOv12 performance

Since YOLOv12 achieved the highest mAP (mean Average Precision) among all versions analyzed, a detailed evaluation of its performance was conducted. The analysis of the curves as a function of the confidence threshold enables understanding of the trade-off between precision and recall, facilitating the identification of the model's optimal operating point, as illustrated in Figures 5.4, 5.5, 5.6, and 5.7.

Figure 5.4 (F1-Confidence) shows the model's overall F1 score, represented by the thicker blue line, which reaches its maximum value of 0.49 at a confidence threshold of 0.16. This point corresponds to the best balance between precision and recall and is therefore considered the ideal operating range for general detection. Among the evaluated classes, the "glass window" class stands out remarkably, maintaining F1-score values above 0.8 across nearly the entire range of confidence thresholds examined.

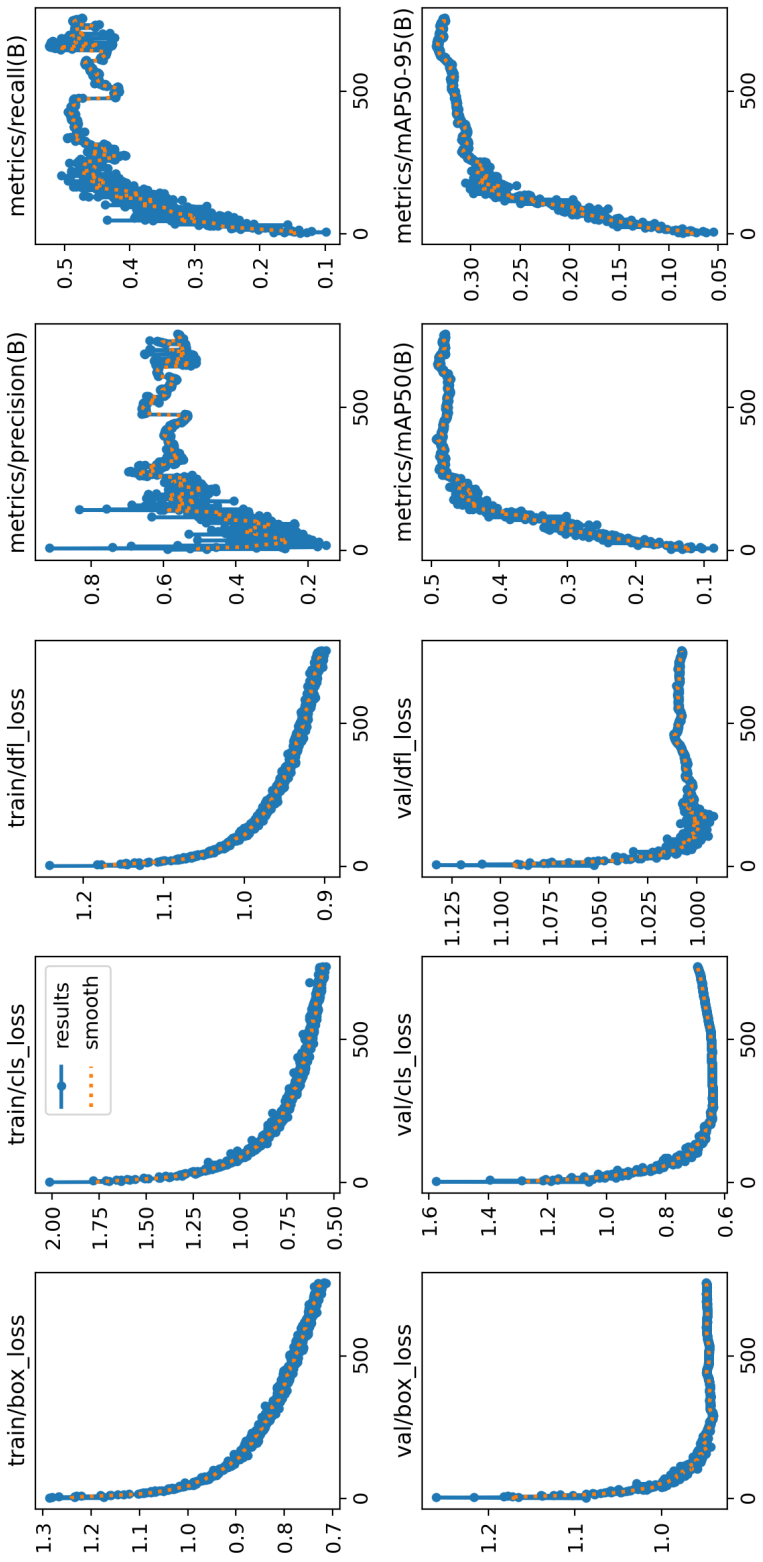
In Figure 5.5 (Precision-Confidence), a monotonic increase in average precision is observed as the confidence threshold is raised, reaching a value of 1.00 at a threshold of 0.967. This behavior is expected, as predictions associated with higher confidence levels tend to be more accurate, albeit fewer in number.

The Precision-Recall curve in Figure 5.6 confirms this trend, recording an mAP50 of 0.486, which reflects moderate overall model performance. Once again, the "glass window" class excels, achieving the highest individual Average Precision (AP) of approximately 0.799 (≈ 0.8).

In turn, the graph in Figure 5.7 (Recall-Confidence) reveals that the global average recall starts at 0.75 for a zero confidence threshold and gradually decreases as the threshold increases. This pattern demonstrates that the model is capable of identifying most true objects at low thresholds, albeit at the cost of a higher incidence of false positives. Again, the "glass window" class maintains recall values above 0.8 over much of the analyzed range, showing high consistency in its predictions.

In summary, the results indicate that YOLOv12 exhibits good generalization capability, with particular emphasis on the "glass window" class, the main focus of this research, which displays a satisfactory balance between precision and recall across the entire range of evaluated confidence thresholds.

Figure 5.3: Results YOLOv12



Source: The Author

Figure 5.4: F1-Confidence YOLOv12

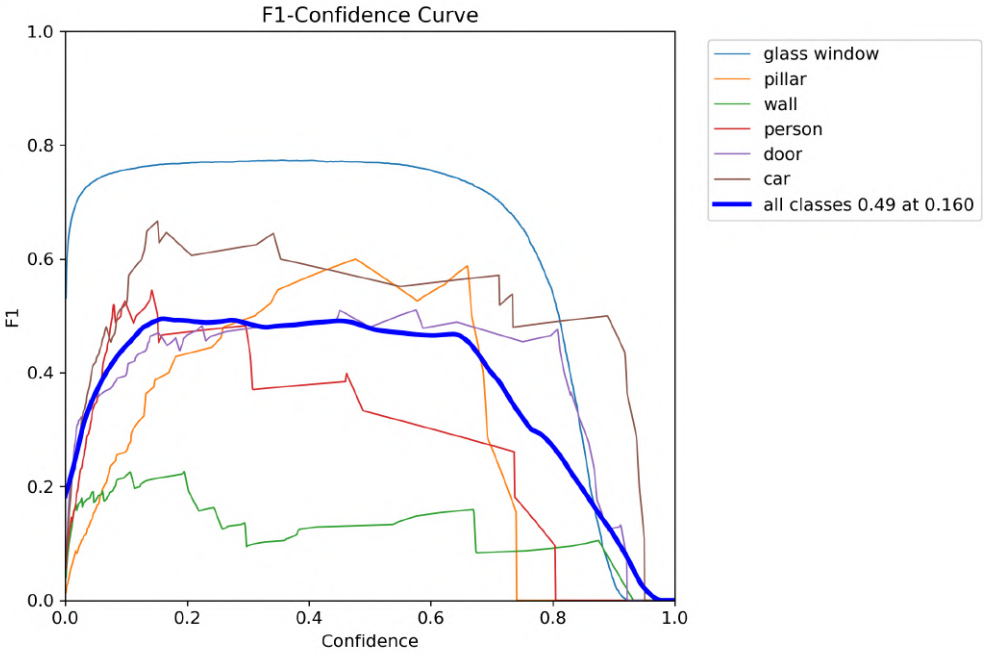


Figure 5.5: Precision-Confidence YOLOv12

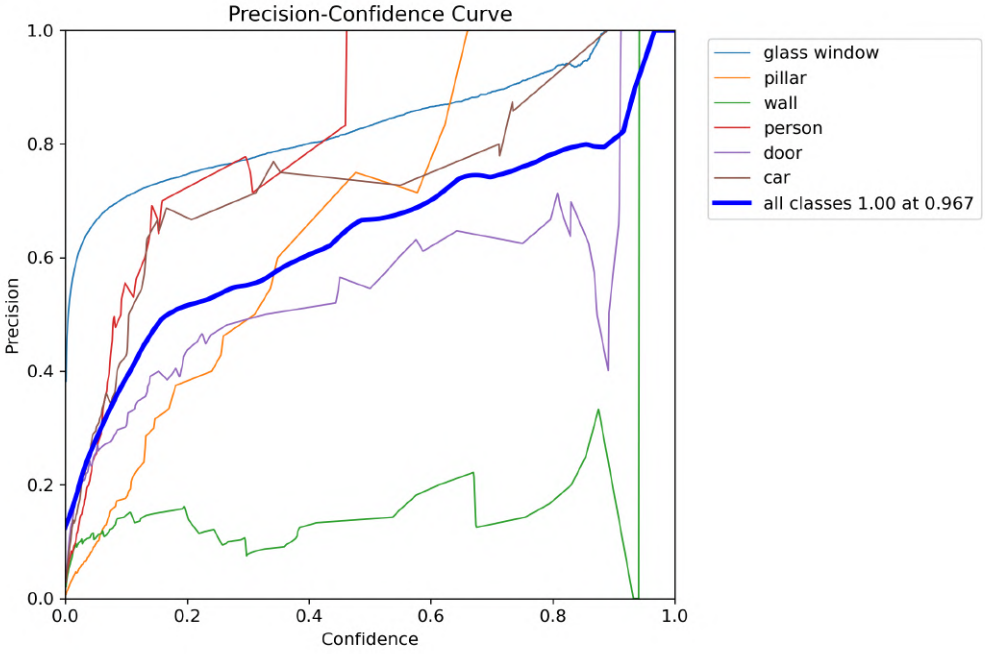


Figure 5.6: Precision-Recall YOLOv12

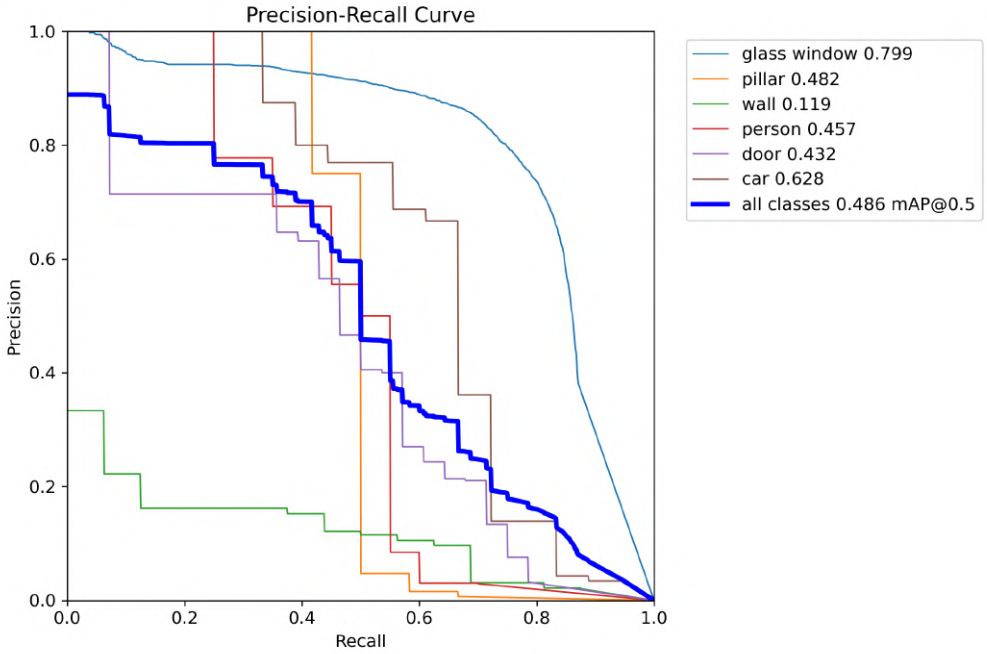
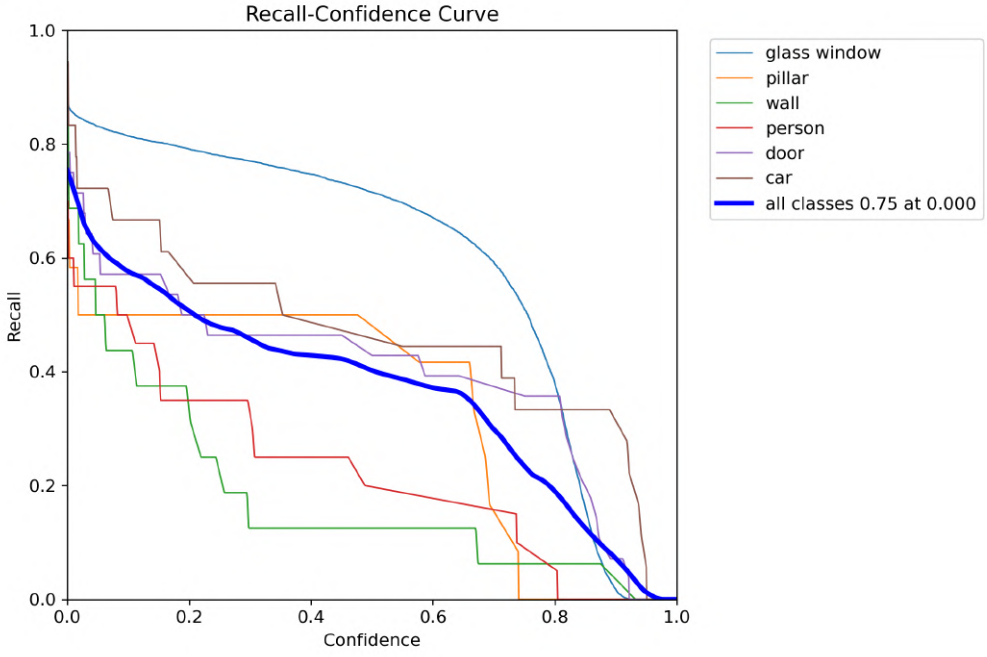


Figure 5.7: Recall-Confidence YOLOv12

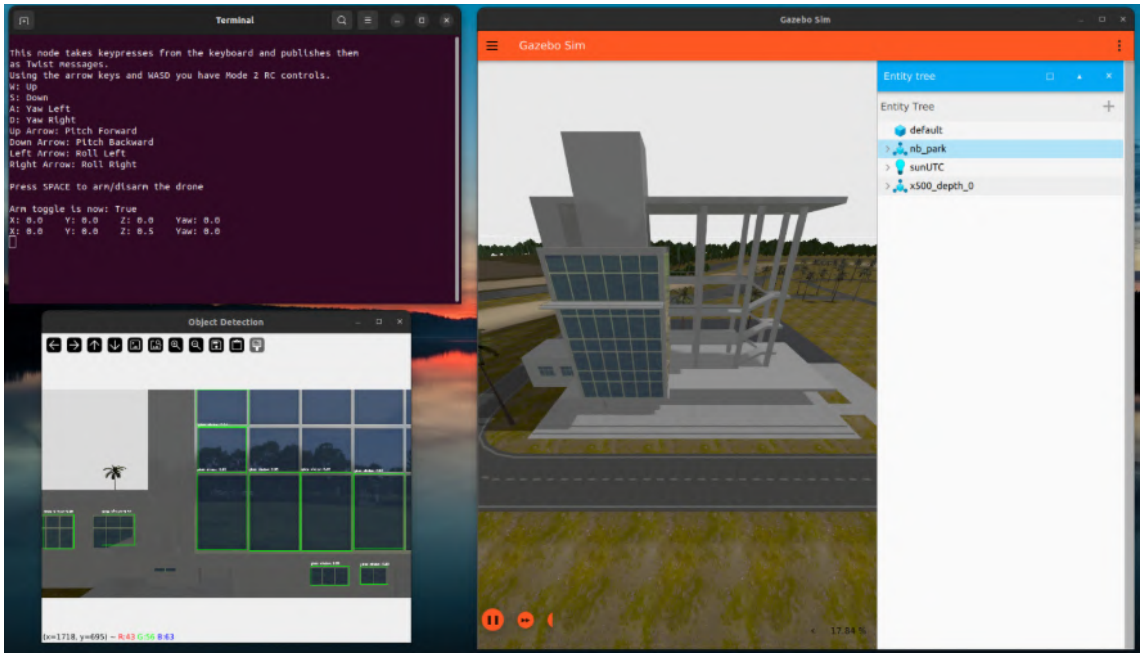


5.2 Validation by Simulation

This section presents the results obtained from the simulation of the integrated system, covering all stages from image acquisition in the virtual environment to the drone's actuation based on detections performed by the neural network. The experiments enable a rigorous evaluation of the window detection performance, the quality of integration and communication between the computer vision and control subsystems, as well as the behavior of the unmanned aerial vehicle during the execution of the proposed mission in the simulated urban scenario.

In the conducted experiments, the unmanned aerial vehicle was assigned an ordered sequence of coordinates (waypoints), referred to as a mission, to follow within a simulated urban environment composed of streets and buildings with windows. This mission was designed so that the drone's trajectory required frontal passage by all the windows of a target building, simulating the autonomous process. Throughout the simulated flight, every image frame processed by the neural network was recorded, allowing the subsequent assembly of a video reproducing the perspective of the onboard camera. The resulting video displays the captured optical flow, the detected windows, and the respective confidence scores associated with each detection. These tests enabled comprehensive validation of the system in a controlled virtual environment.

Figure 5.8: Urban environment drone simulation



Source: The Author

The simulation process began with image acquisition by the optical sensor (RGB camera) within the Gazebo environment, with the images being transmitted to ROS 2 through the gz-msgs and gz-transport communication packages, which are responsible for bridging data exchange between the simulator and the middleware, as illustrated in Figure 5.8. Upon reception, the neural network performed real-time inference on each frame, detecting and delimiting windows in the scene using bounding boxes while assigning the corresponding confidence score to each detection, as shown in Figure 5.9.

Figure 5.9: UAV point-of-view simulation



Source: The Author

Subsequently, control commands generated by the ROS 2 node based on the analysis of the inference results were forwarded to the PX4 flight stack via Micro XRCE-DDS, which serves as a translation layer between the high level ROS 2 environment and the embedded flight controller firmware. The PX4, in turn, interpreted these commands and executed them, thereby altering the drone's dynamic state and behavior within the simulated environment. In this manner, the complete perception decision action loop was virtually replicated, allowing integrated evaluation of the computer vision, autonomous control, and navigation modules in a realistic and reproducible scenario.

To enable the continuous acquisition and availability of images from the camera simulated within the Gazebo environment, the code referenced in Algorithm 1 implements a class responsible for establishing communication with the message transport system (gz.transport) and converting the received data into a format suitable for computer vision processing.

Initially, the `Camera` class is instantiated with the camera topic name (in this case, `/camera`) and the expected image resolution (1920×1080). During initialization, a communication node is created and subscribed to the specified topic, allowing it to receive messages of type `Image`. Whenever a new image is published by the simulator, the `ImageCallback` method is automatically triggered.

Inside `ImageCallback`, the raw image data is extracted and converted into a NumPy array of type `uint8`. The data is then reshaped according to the defined resolution, structuring the three-dimensional matrix corresponding to the color image (height × width × 3 channels). Subsequently, the color space is converted from RGB to BGR to ensure compatibility with the OpenCV library, which internally adopts this standard.

The code also incorporates a synchronization mechanism based on `threading.Condition`, allowing the main thread to wait until a new image becomes available. The `get_next_image()` method blocks execution until `ImageCallback` updates the internal variable storing the most recent frame. After retrieval, the image is returned and the buffer is cleared, ensuring that only new frames are processed.

In the main execution block, the system runs in a continuous loop in which each newly captured image is obtained through the `get_next_image()` method and immediately displayed using `cv2.imshow()`. This step enables real-time frame visualization and serves as the foundation for integration with additional processing modules.

The structure was designed in a modular manner, allowing each captured frame to be directly forwarded to an object detection model, such as YOLO, without disrupting the main image acquisition flow. When integrated with the code referenced in **??**, each image undergoes the inference process, in which the class “glass window” and other classes are identified, bounding boxes are extracted, and the corresponding image-plane coordinates are computed.

Based on these results, the system can generate high-level commands for autonomous navigation, trajectory adjustment, or external device actuation, establishing a continuous pipeline between visual perception, decision-making, and actuation. In this way, the main loop not only ensures constant image updates but also sustains the integration between computer vision and embedded system control.

Algorithm 1: Camera Image Acquisition and Visualization Algorithm

```
1  Function MAIN()
2      cam ← InitializeCamera(topic_name, resolution);
3      while true do
4          img ← GetNextImage(cam);
5          DisplayImage(img);
6      end
7  end
8  Function InitializeCamera(topic_name, resolution)
9      Create communication node;
10     Subscribe to camera topic;
11     image ← null;
12     Initialize condition variable;
13     return camera;
14 end
15 Function ImageCallback(received_image)
16     raw_data ← received_image data;
17     np_image ← reshape raw_data;
18     cv_image ← convert RGB to BGR;
19     Lock condition variable;
20     image ← cv_image;
21     Notify waiting threads;
22     Unlock condition variable;
23 end
24 Function GetNextImage(camera)
25     Lock condition variable;
26     if image is null then
27         Wait until image is available;
28     end
29     return_image ← image;
30     image ← null;
31     Unlock condition variable;
32     return return_image;
33 end
34 Function DisplayImage(img)
35     Show image in window;
36     Wait small time interval;
37 end
```

To perform object detection of the trained classes from images captured by the perception system, the code referenced in Algorithm 2 implements a function based on the YOLO architecture, using the OpenCV `cv2.dnn` module to execute inference of the trained neural network.

Initially, the function `detect_objects(image)` receives as input an image already acquired by the camera module. The code then automatically identifies the base directory of the executing file and builds the paths to three fundamental model files: the network configuration file (`yolo.cfg`), the trained weights file (`yolo_last.weights`), and the file containing the class names (`classes.txt`). The latter is read and stored as a list of labels that will be used for textual identification of detections.

Subsequently, the neural network is loaded using the `cv2.dnn.readNetFrom Darknet` function, which interprets the structure defined in the `.cfg` file and associates the previously trained weights. Once loaded, the input image is preprocessed to meet the model requirements. This preprocessing consists of creating a normalized blob (scale factor $1/255$), resizing it to 416×416 pixels, the standard YOLO input size, and reorganizing the color channels (BGR to RGB, when necessary). The blob is then set as the network input.

During the inference stage, the code retrieves the names of the unconnected output layers (`get_Layers_Names`) and performs the forward pass using `net.forward()`. The result consists of multiple detection vectors, each containing normalized bounding box coordinates, objectness probability, and class probabilities.

For each detection, the code extracts the class scores and identifies the one with the highest probability using `np.argmax`. If the associated confidence exceeds the threshold of 0.5, the detection is considered valid. The normalized coordinates are then converted into the original image pixel coordinate system by computing the top-left corner of the bounding box from the predicted center coordinates and estimated dimensions.

All bounding boxes, confidence scores, and class identifiers are stored in lists. Then, the Non-Maximum Suppression (NMS) algorithm is applied using `cv2.dnn. NMSBoxes` to eliminate redundant detections and retain only the most representative bounding boxes for each identified object.

Finally, for each remaining detection after NMS, the code draws a green rectangle around the detected object and adds a label containing the class name and the estimated confidence value. The annotated image is then returned by the function.

In the context of glass window detection in a simulated environment, this code

corresponds to the inference module of the perception system. It receives frames previously acquired through the Gazebo interface, processes them using the YOLO network trained for the classes *Glass Window*, *Pillar*, *Wall*, *Person*, *Door*, and *Car*, and returns the image with visual detection annotations. In this way, it integrates into the complete perception pipeline, enabling the system to automatically identify classes in the scenario and use this information for decision-making, autonomous navigation, or actuator triggering in the drone or ground robot.

Automation arises when the output of the perception module is connected to the mission execution layer. The mission is defined in an external file (`mission.txt`) composed of sequential commands in the format `go:x,y,z;`. These commands represent movement references along the three spatial axes and define a structured inspection trajectory, typically resembling a horizontal sweep with progressive vertical displacement. To ensure that mission updates are immediately reflected during execution, the `loadmission()` function copies the edited mission file from the development directory to the path of the installed package used at runtime. In a *ROS 2* environment, executable nodes access files from the installation directory rather than directly from the source tree; therefore, this copy operation ensures synchronization between the defined mission and the running system.

During operation, the control node reads the mission file, parses each `go` command, converts the parameters into numerical references, and publishes them as offboard setpoints to the autopilot running on PX4. Since the aircraft operates in offboard mode, all trajectory references originate from external control logic. When integrated with the perception module, detection results can dynamically influence these references. Once the *Glass Window* class is identified in the image and its bounding box coordinates are calculated, the system can adjust velocity commands to center the target in the field of view, refine the inspection trajectory, or trigger additional actions, such as hovering, descending, or activating external devices.

In this configuration, automation is not limited to executing pre-defined trajectories; instead, it is guided by real-time visual feedback. The continuous acquisition loop ensures that recent images are always available, the YOLO-based module extracts semantic information from the environment, and the offboard controller converts high-level decisions into movement commands. This establishes a closed-loop autonomous system, in which perception directly influences navigation behavior, enabling adaptive inspection of glass facades in the simulated environment without human intervention.

Algorithm 2: YOLO Object Detection Algorithm

```
Function DetectObjects(image)
    base_dir ← GetScriptDirectory();
    cfg_file ← "yolo.cfg";
    weights_file ←"yolo_last.weights";
    class_file ← "classes.txt";
    classes ← ReadLines(class_file);
    net ← LoadYOLO(cfg_file, weights_file);
    (height, width) ← GetImageDimensions(image);
    blob ← CreateBlob(image, scale=1/255, size=416×416);
    SetNetworkInput(net, blob);
    output_layers ← GetOutputLayers(net);
    detections ← ForwardPass(net, output_layers);
    boxes ← ∅;
    confidences ← ∅;
    class_ids ← ∅;
    foreach output ∈ detections do
        foreach detection ∈ output do
            scores ← detection[5:];
            class_id ← ArgMax(scores);
            confidence ← scores[class_id];
            if confidence > 0.5 then
                (cx, cy, w, h) ← ScaleBox(detection, width, height);
                x ← cx − w / 2;
                y ← cy − h / 2;
                Append(boxes, (x, y, w, h));
                Append(confidences, confidence);
                Append(class_ids, class_id);
            end
        end
    end
    indices ← NonMaximumSuppression(boxes, confidences, 0.5, 0.4);
    if indices is not empty then
        foreach i ∈ indices do
            (x, y, w, h) ← boxes[i];
            label ← classes[class_ids[i]] + confidence[i];
            DrawBoundingBox(image, x, y, w, h);
            DrawLabel(image, label);
        end
    end
    return image;
end
```

The complete simulation implementation is also publicly available and can be accessed via the project simulation repository[1].

5.3 Field Experiments with real UAVs

In this section, the validation stage of the proposed system under real-world operating conditions is described, in which the drone was subjected to experimental trials in a controlled real-world environment. The tests aimed primarily to evaluate the performance of the onboard system during actual operation, with special attention to flight stability, target detection accuracy, and the unmanned aerial vehicle's responsiveness to commands issued by the ground control module.

Additionally, relevant environmental factors were analyzed, such as external interference, variations in illuminance, and the behavior of the onboard sensors, in order to verify the system's robustness and reliability in practical operational scenarios.

Due to regulatory restrictions that make it difficult to obtain authorization for medium sized drone flights in urban areas, the experiments were conducted in two stages: a preliminary one carried out inside the Federal University of Rio Grande do Sul, at the downtown campus, using an 3DR Iris+ drone on May 8, 2025, and another at the private airport facilities of Skydrones, located in Eldorado do Sul, Rio Grande do Sul, a municipality in the metropolitan region of Porto Alegre, approximately 17 km from the capital, on October 30, starting at 16:00 (Brasília time, Brazil).

The meteorological conditions recorded on May 8, 2025 were: average temperature of 20.3 °C, average relative humidity of 83%, average dew point of 17.34 °C, average atmospheric pressure of 1005 hPa, no precipitation was recorded. No wind data were found on the INMET (INMET, 2025) website, but the authors state that there were no significant winds that could interfere with the flight on that day.

And the meteorological conditions recorded on October 30, 2025 were: average temperature of 19.56 °C, average relative humidity of 76%, average dew point of 14.56 °C, average atmospheric pressure of 1017.4 hPa, predominant wind of 2 m/s from direction 130°, and precipitation of 0.2 mm (INMET, 2025).

[1] <https://github.com/monymon/UAV-Window-Detection>

5.3.1 Experiments with the 3DR Iris+ drone

For the preliminary test conducted with the Iris+ drone, a visual indicator consisting of an LED connected to the ASUS Tinker Board via its GPIO ports was implemented. The LED was activated whenever the computer vision model detected the presence of a window in the captured frame, even though in a digital environment, as demonstrated by Figure 5.10. This test was performed manually, without any flight automation or decision making mechanisms, with the following objectives: to validate the real-time inference performance of the neural network under actual flight conditions and to confirm the proper integration and communication between the Pixhawk flight controller, the Tinker Board, and the Intel RealSense camera.

In Figure 5.11, the drone can be observed in flight over one of the buildings on the Centro campus, with the green LED illuminated, indicating positive window detection. In turn, Figure 5.12 shows the detection output generated during the test, enabling a preliminary analysis of the model's behavior in an outdoor environment

Figure 5.11: UAV identifying windows



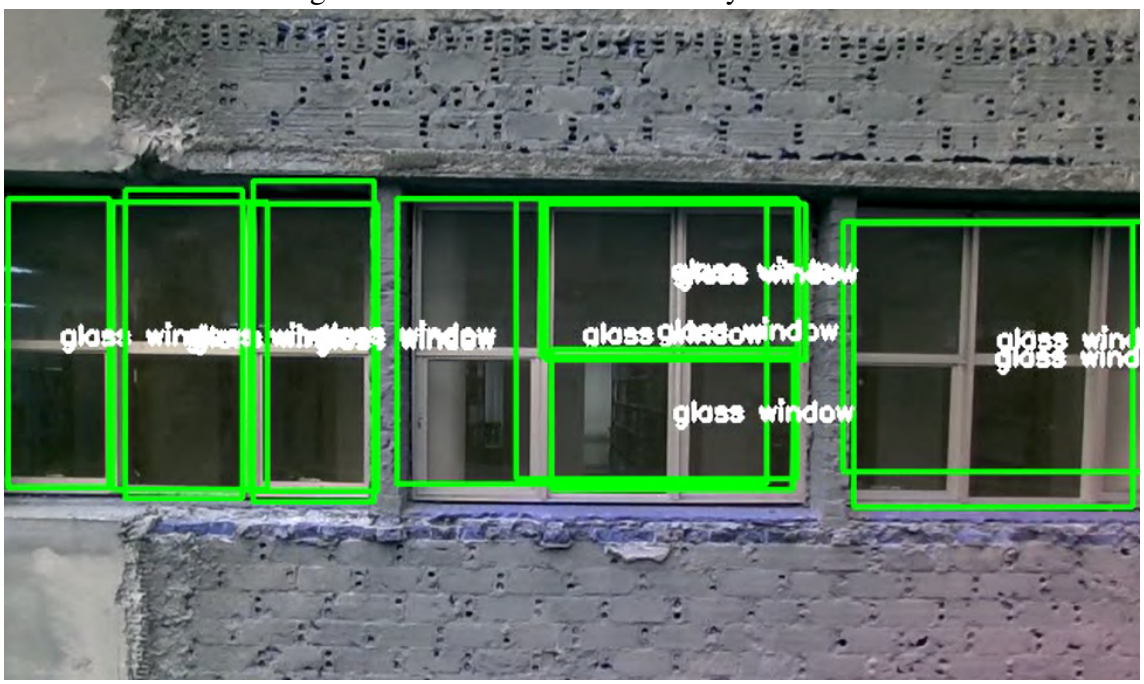
Source: The Author

Figure 5.10: Detection of a window in a digital image



Source: The Author

Figure 5.12: Detection identified by the UAV



Source: The Author

5.3.2 Experiments with the Skyclean drone

For the validation of the proposed system, the drone performed a reciprocal (back-and-forth), as shown in Figure 5.13 flight trajectory at an average altitude of approximately 7 m, as shown in the graph 5.15, which was generated from the fused data of the two barometric sensors embedded in the unmanned aerial vehicle.

Initially, an altitude oscillation between 5 m and 7 m is observed, followed by an abrupt drop to approximately 3 m. This behavior corresponds to the initial approach test, during which the detection performance showed significant limitations, as evidenced in frames 1 and 2 (see Figure 5.14).

Next, the graph 5.15 records a gradual descent to approximately 3 m, which corresponds to the height of the windows relative to the ground. In this phase, a substantial improvement in detection performance is observed, as illustrated in frames 3 and 4. This improvement is associated with the reduced distance between the drone and the facade (around 2.5 m to 3 m) and with the more centered alignment of the camera relative to the building, a visual comparison between frames 1 and 4 highlights this effect.

Frames 5 and 6 correspond to the moment when the drone performs a yaw maneuver to the right. Abrupt lateral movements of this nature introduce temporary instability

Figure 5.13: Skyclean performing a detection mission



Source: The Author

in altitude control, negatively affecting detection quality. Once the trajectory is stabilized again, as shown in frames 7, 8, and 9, the detection system returns to a stable and consistent performance.

Overall, the test shows that lateral distance, flight altitude, and camera alignment relative to the target directly influence detection accuracy.

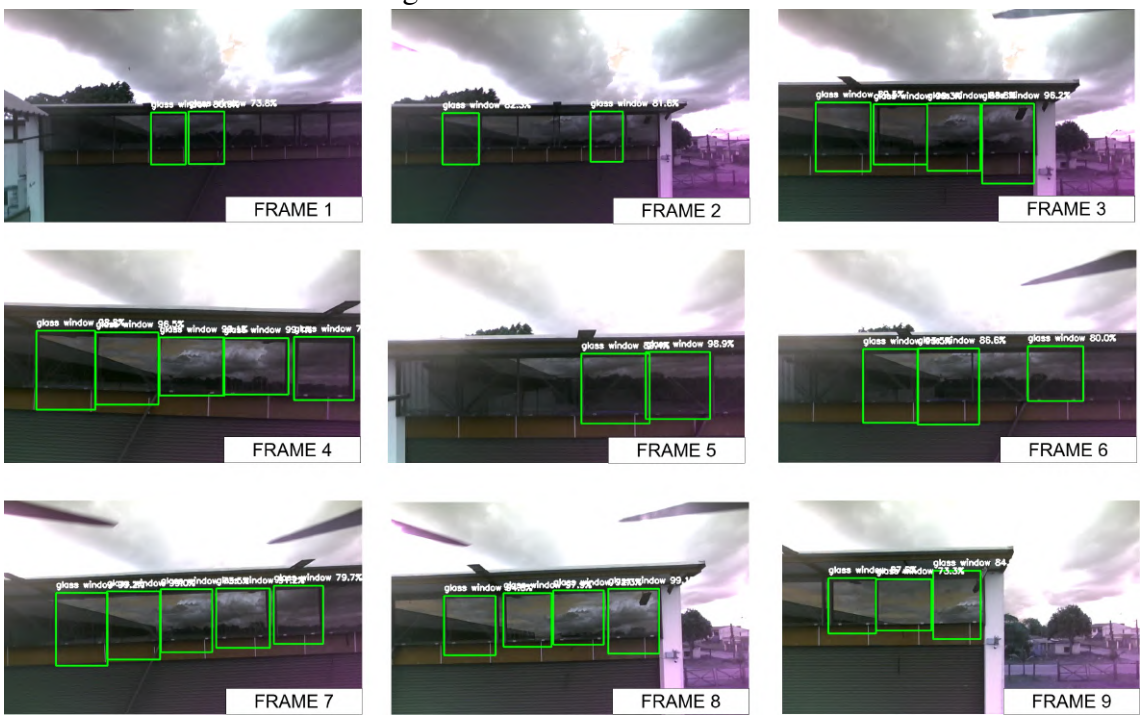
Another critical factor identified by the authors concerns exposure to direct sunlight. Previous tests conducted under intense solar illumination conditions revealed the complete inability to detect any windows, demonstrating that significant variations in ambient lighting constitute a relevant limiting factor for the robustness of the proposed model's inference process.

To address this issue, the possibility of using optical filters was considered; however, this approach did not yield satisfactory results, which explains why the images available in the video present a purplish coloration. Another potential alternative would be the use of solar lenses attached to the camera; nevertheless, this hypothesis was not tested in the present study.

The code was designed to consider only detections with confidence above 70% in the predictions. As a result, some values reach around 90%, while others remain slightly below that, but never below the established threshold. This strategy was adopted to reduce the likelihood of false positives, ensuring that when the system identifies a glass window, it does so with high reliability.

The Figure 5.16 shows the record of the device's internal temperature throughout

Figure 5.14: Frames Detection



Source: The Author

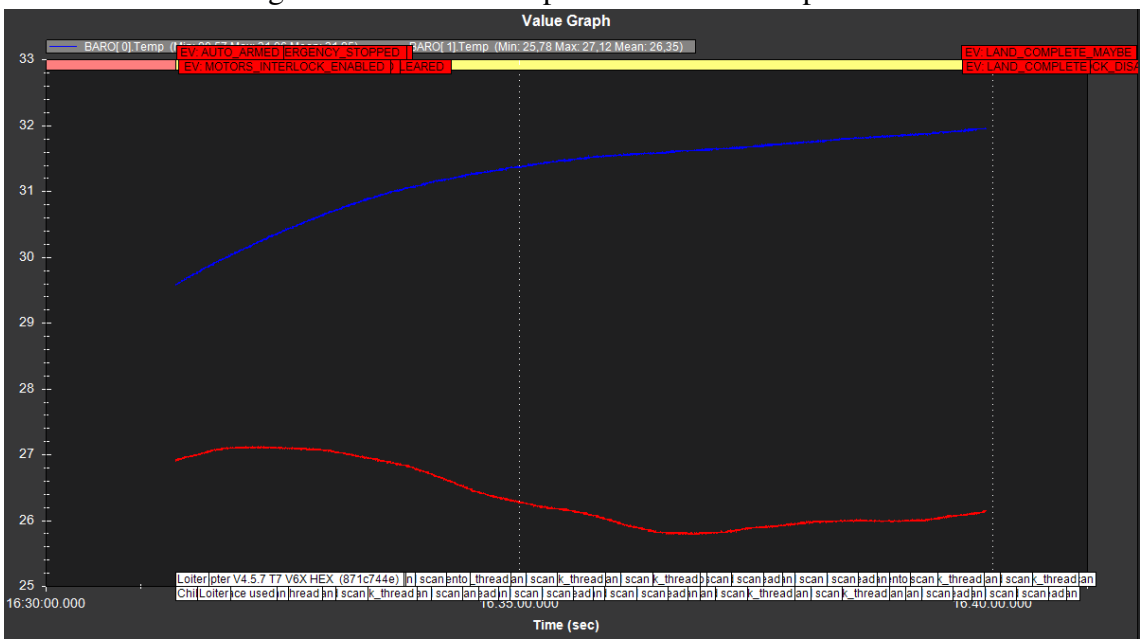
the entire mission, obtained through two independent temperature sensors. An average temperature of approximately 29 °C is observed, with reduced variations during the test, which demonstrates that the drone maintained stable thermal behavior and was fully compatible with the conditions expected for the conducted test scenario.

Figure 5.15: Drone altitude relative to the ground



Source: (SKYDRONES, 2025)

Figure 5.16: Drone temperature and its components



Source: (SKYDRONES, 2025)

The Figure 5.17 illustrates the facade cleaning procedure. In this specific test, the operation was performed by directing the water jet toward a wall without windows, with the purpose of evaluating the pump's performance.

The automation architecture will not be presented in this work, as it is the company's intellectual property (SKYDRONES, 2025).The conceptual operation is presented in Figure 4.7 in Chapter 4, while the corresponding discussion is addressed in the simulation in Section 5.2.

Figure 5.17: Test of the Drone Using the Pump



Source: The Author

6 DISCUSSIONS AND LESSONS LEARNED

The initial development of this work was conducted based on the YOLOv7 architecture, which, at the start of the project, represented the most recent and advanced version of the YOLO series that was widely and publicly available. However, with the release of newer versions and the emergence of evidence questioning the superiority of YOLOv7, it was decided to carry out a systematic comparative evaluation of all relevant versions of the YOLO family, with the aim of identifying the architecture that delivered the best performance on the specific dataset proposed.

In this evaluation process, it was observed that YOLOv4, trained under the Darknet framework, outperformed YOLOv7 in detection metrics for the problem at hand. Despite this result, uncertainty remained regarding the existence of an even more suitable version, which led to the decision to train and evaluate all contemporary versions of the YOLO series.

Initially, the public CMP Facade Database was used, a dataset widely referenced for segmentation and detection tasks on building facades. However, experiments revealed that this dataset was not sufficiently representative of the target glass windows of the project, featuring outdated architecture and low variability compared to the real-world scenarios of interest. Given the impossibility of achieving significant precision gains with this dataset, it became necessary to build a custom dataset specifically tailored for detecting glazed windows in modern buildings.

Manual image collection from the web proved excessively laborious and time consuming. To accelerate this stage, generative image models, particularly Stable Diffusion, were employed to synthetically create realistic facade images. Although this strategy helped increase data volume, the manual annotation of bounding boxes remained the main bottleneck, requiring approximately one month per complete annotation cycle a process that had to be repeated three times. Attempts to automate annotation using artificial intelligence models were made, but the results obtained were not satisfactory.

Regarding integration into embedded systems, the main challenges lay in model portability and adaptation to different hardware platforms and drones. Each change to the model or inference pipeline required new rounds of field testing, which consumed considerable time. In this regard, the partnership with the Skydrones team proved essential, providing critical technical support to overcome the embedded implementation difficulties.

Regarding the contributions obtained in this work, it is possible to perform a comparative analysis with recent literature results in order to validate potential improvements. For instance, in the study by Pandey et al., different versions of YOLO (v8, v11, and v12) were evaluated for defect detection and classification in glass facades captured by UAVs. The best performance was achieved with YOLOv12, reaching 70% accuracy on the test set. Although this result demonstrates the feasibility of the approach for structural monitoring, the performance still leaves room for improvement, especially considering the visual complexity of glass facades (reflections, transparency, lighting variations, and perspective).

In contrast, this work presented a broader comparative evaluation, involving ten versions of the YOLO family (v2 to v12), applied specifically to the detection of the "glass window" class. The best result was also obtained with YOLOv12; however, the model achieved 80% mean Average Precision (mAP) on the combined real and synthetic dataset. This value represents an improvement of approximately 10 percentage points compared to the performance reported by Pandey et al.

This advancement can be attributed to several key factors:

- **Dataset specialization** – In this work, a dataset was developed specifically targeting contemporary facades with a predominance of glass, whereas Pandey's work focuses on images aimed at defect identification. The curation and standardization contributed to greater consistency of the samples.
- **Use of synthetic data generated by diffusion models** – The incorporation of synthetic images generated by Stable Diffusion expanded the dataset variability without compromising the structural coherence of the facades. This favored model generalization and mitigated typical data scarcity issues.
- **Rigorous annotation validation process** – This work describes a structured stage of visual validation of bounding boxes and manual review of annotations, reducing noise during training.
- **Systematic evaluation of multiple architectures** – The comparative analysis across ten YOLO versions allowed selecting the architecture most suitable for the specific domain, optimizing performance before embedded deployment.

In addition to the quantitative gain, it is important to highlight that the metric used in this dissertation (mAP) is more robust for evaluating object detection than simple overall accuracy, as it considers precision and recall across different confidence thresholds.

Therefore, besides presenting a higher numerical value, the result also reflects a more rigorous assessment of detector performance.

Thus, the best performance achieved in this dissertation, 80% mAP with YOLOv12, not only surpasses the 70% accuracy reported by Pandey et al., but also demonstrates the positive impact of constructing a specialized dataset, integrating synthetic data, and applying structured methodological validation on improving glass facade detection in UAV-embedded systems.

Finally, real world testing with the Skyclean robot faced significant obstacles related to obtaining authorizations for operation in urban areas, due to risks associated with proximity to vehicles and pedestrians. This limitation was circumvented by using private or temporarily unoccupied spaces, as occurred during experiments conducted on the premises of the Federal University of Rio Grande do Sul (UFRGS).

The execution of all experiments was properly recorded, and the corresponding videos are publicly available[1]. The dataset is publicly available and can be accessed through the project repository[2]. The complete simulation implementation is also publicly available and can be accessed via the project simulation repository[3].

In summary, the development of the system required iterative methodological adaptation, the construction of a custom dataset, and the overcoming of operational and regulatory constraints factors that, although they extended the timeline, were essential to achieving the robustness and accuracy required by the final application.

[1] <https://drive.google.com/drive/folders/1TN7hUDsLMf6oiF-GDIC8PhBS90KLsACn>
[2] <https://github.com/monymon/dataset_glasswindow>
[3] <https://github.com/monymon/UAV-Window-Detection>

7 CONCLUSION

The present research provided a deep understanding of how vehicles that initially lack intelligent capabilities can be transformed into systems capable of self governance through the implementation of artificial intelligence techniques.

The use of synthetic data generated by Latent Diffusion Models (LDMs) demonstrated the high potential of such data to revolutionize the training of detectors for novel object classes, provided these classes do not exhibit excessive complexity. By addressing the scarcity of annotated data for glass facades, this work made a significant contribution to the advancement of computer vision applied to contemporary architecture, enabling precise and computationally efficient solutions to real-world problems.

Among the ten variants of the YOLO algorithm evaluated, the YOLOv12 model achieved the best performance, reaching a mean Average Precision (mAP) of 80% for the “glass window” class when trained on the combined dataset (real + synthetic). These results reinforce the superiority of the most recent YOLO versions in terms of accuracy. However, the comparative analysis of the training processes revealed that the latest version is not always the optimal choice for all practical applications. In the case studied, although YOLOv12 exhibited the highest mAP, YOLOv4 stood out due to its native implementation in the C language, offering practical performance advantages.

Simulation tests conducted in a Gazebo and ROS2 environment successfully validated the integration of the perception, decision making, and control modules, demonstrating that, in a fully controlled environment, the drone was able to perform the automation task with excellence, successfully completing the proposed mission and correctly identifying the windows.

Additionally, field experiments confirmed the operational robustness of the developed system. The embedded platform performed reliable detections, proving the system’s effectiveness under real-world conditions. Tests carried out on different drone platforms highlighted its scalability, adaptability, and high potential for application across a wide range of operational scenarios.

This work received (technical/infrastructure) support from the Center For Embbeded Devices and Research in Digital Agriculture (CEDRA) - SENAI-RS.

7.1 Future Research Directions

Future work includes expanding the dataset to encompass a broader range of lighting conditions, viewing angles, and architectural styles, as well as incorporating more advanced data augmentation strategies.

Another research direction involves the development of hybrid models that integrate object detection and material classification to enhance scene understanding.

Additionally, the implementation of more sophisticated navigation and cleaning trajectory planning algorithms is planned.

Future steps also include the development of an autonomous and efficient method to streamline the data annotation process, as well as the extension of the proposed approach to other objects and scenarios.

7.2 Publications and Submissions

1. MONTEIRO, M. G.; POHREN, D. H.; SANTOS ROQUE, A. and FREITAS, E. P. **A preprocessed and labeled image dataset of building facades for UAV-Based maintenance applications**.

   Overview – This paper presents a curated dataset of contemporary building facades, consisting of 3,805 images that were preprocessed and manually annotated in YOLO format, including components such as windows, doors, and structural elements. The dataset was collected through web searches, filtered for quality and relevance, and validated for consistency. This work fills a gap in publicly available labeled image databases and supports the development of AI-based UAV systems for tele-operated and autonomous maintenance, while also enabling real-time visual analysis for smart city and IIoT applications.

   Notes – The Paper was submitted and accepted in conferece **IFAC-TA 2025**

2. MONTEIRO, M. G.; POHREN, D. H.; SANTOS ROQUE, A. and FREITAS, E. P. **Enabling Facade Cleaning Drones with Enhanced Window Detection via YOLO**.

   Overview – This work evaluates YOLO based architectures for detecting glass windows on building facades to support autonomous drone cleaning. A custom dataset of 3805 images, combining real, synthetic, and augmented data with six

annotated classes, was used for training. Results show that synthetic data improved performance, with YOLOv12 achieving the highest accuracy (mAP50 of 80%), while YOLOv4 remains the best choice for embedded deployment due to its C based implementation. The study highlights the importance of dataset augmentation and advanced deep learning for safe and efficient UAV facade maintenance.

Notes – The Paper was submitted and currently in review phase to the journal **Multimedia Systems**, under the ID b0d2e88a-4251-4602-a906-aa54124e75db.

3. MONTEIRO, M. G.; POHREN, D. H.; SANTOS ROQUE, A. and FREITAS, E. P. **Real-time Embedded Systems, Autonomous Drones, Computer Vision, Building Facades, Object Detection**.

   Overview – Unmanned Aerial Vehicles (UAVs), traditionally used in military operations, have increasingly found applications in civilian areas due to advances in technology. Their ability to carry various sensors makes them ideal for tasks in hard-to-reach or hazardous locations. This paper focuses on UAV-based building facade maintenance, specifically cleaning and inspection. Facade analysis is treated as a semantic segmentation problem, facing challenges like complex environments, variable lighting, and obstructions. To address these, the study proposes an AI-based system using neural networks and convolutional architectures such as YOLOv7. The system achieved 98% accuracy in detecting windows, outperforming previous works and demonstrating its potential to enhance drone autonomy and safety in facade maintenance.

   Notes – The Paper was submitted and currently in review phase to the Workshop **Next Generation Real-Time Embedded Systems**

4. MONTEIRO, M. G.; KOHL, G. and FREITAS, E. P. **UAV-based Building Facade Cleaning Using Autonomous Windows Detection**.

   Paper in conclusion phase. *Overview* – This paper presents an autonomous unmanned aerial vehicle (UAV)-based system for cleaning glass building facades. The proposed solution combines a real-time computer vision subsystem, built upon the YOLO (You Only Look Once) object detection architecture for accurate window localization, with an embedded control module that actuates high-pressure water nozzles to perform the cleaning task. A custom dataset composed of both real-world and synthetic images of glass facades was built and used to train and evaluate several models from the YOLO family. Among them, YOLOv12 achieved the highest performance, attaining a mean Average Precision (mAP50) of 80% for the

"glass window" class. The integrated system was thoroughly validated through simulations in Gazebo/ROS2 and real-world field experiments using a dedicated UAV platform (Skyclean drone), demonstrating reliable detection performance and safe operation. Results confirm that the approach effectively automates facade maintenance, significantly improving the cleaning operation, diminishing costs and human risk exposure, while providing a robust and extensible framework for future fully autonomous facade cleaning systems.

Notes – The Paper was submitted and currently in review phase to the journal **Unmanned Systems**, under the manuscript number US-D-25-00291.

Appendix A - Resumo Expandido

Sistema de Limpeza de Fachadas de Edifícios com Drones Baseado em Detecção Autônoma de Janelas

A crescente verticalização dos centros urbanos e a predominância de fachadas envidraçadas na arquitetura contemporânea têm ampliado a demanda por soluções automatizadas para inspeção e manutenção predial. A limpeza dessas superfícies representa um desafio operacional significativo, envolvendo riscos humanos, alto custo logístico e limitações de acesso. Nesse contexto, Veículos Aéreos Não Tripulados (UAVs) emergem como uma alternativa promissora, especialmente quando integrados a sistemas de visão computacional capazes de identificar, segmentar e interpretar elementos estruturais em tempo real.

Nesse contexto, os desafios inerentes às fachadas modernas predominantemente envidraçadas justificam a abordagem proposta neste estudo. Essas estruturas apresentam obstáculos significativos para análise computacional devido à variabilidade de iluminação, reflexos intensos, mudanças de perspectiva, oclusões e ampla diversidade arquitetônica. Além disso, observa-se a escassez de bases de dados específicas voltadas para fachadas de vidro contemporâneas, o que reforça a relevância da geração de dados sintéticos e da avaliação sistemática de diferentes arquiteturas de detecção.

Para superar essa limitação, foi construído um dataset especializado composto por imagens reais coletadas da internet e imagens sintéticas geradas por Modelos de Difusão Latente (LDMs), utilizando a arquitetura Stable Diffusion. A geração sintética permitiu ampliar a variabilidade do conjunto de treinamento e melhorar a capacidade de generalização dos modelos.

O desenvolvimento foi estruturado em cinco etapas principais:

- Construção do dataset (real + sintético);
- Treinamento e comparação de diferentes versões do YOLO;
- Implementação do modelo em sistema embarcado;
- Validação em ambiente simulado (ROS 2, Gazebo e PX4);
- Testes experimentais com UAVs reais.

Foram coletadas 1.040 imagens reais, anotadas manualmente, complementadas por imagens sintéticas geradas com Stable Diffusion e técnicas de *data augmentation* para aumentar a robustez do modelo. As classes incluíram *glass window*, *door*, *wall*, *pillar*, *car* e *person*. Diferentes versões do YOLO foram treinadas e comparadas quanto à precisão, *recall*, *mAP* e custo computacional, permitindo a seleção da arquitetura mais adequada ao sistema embarcado. O modelo foi implementado para execução em tempo real e integrado ao pipeline de percepção do drone. A validação foi realizada em ambiente simulado (ROS 2, Gazebo e PX4), seguida por testes com UAVs reais, confirmando a viabilidade prática da abordagem.

Foi realizada uma análise comparativa entre dez versões do YOLO:

- YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOv7, YOLOv8, YOLOv9, YOLOv10, YOLOv11 e YOLOv12.

Os modelos foram treinados com dados reais (RD) e dados combinados reais + sintéticos (RSD), utilizando como métrica principal o *mean Average Precision* (mAP). O YOLOv12 apresentou o melhor desempenho, alcançando 80% de mAP para a classe *glass window*. No entanto, o YOLOv4 demonstrou vantagens práticas devido à sua implementação nativa em C, tornando-se adequado para sistemas embarcados com restrições computacionais.

O sistema integra uma câmera Intel RealSense para captura contínua de imagens, o modelo YOLO embarcado para detecção em tempo real e um subsistema de controle responsável pelo acionamento automático dos jatos de limpeza. A validação foi realizada inicialmente em ambiente simulado utilizando ROS 2, Gazebo e PX4 Autopilot, confirmando a integração entre percepção, tomada de decisão e controle. Posteriormente, testes reais foram conduzidos com os drones 3DR Iris+ e Skyclean, confirmando robustez operacional, adaptabilidade e estabilidade.

As principais contribuições incluem:

- Criação de dataset especializado em fachadas de vidro;
- Uso estratégico de dados sintéticos via LDM;
- Avaliação comparativa extensa entre versões do YOLO;
- Integração completa entre visão computacional e controle embarcado;
- Validação em simulação e ambiente real.

Os resultados demonstram que a combinação entre geração de dados sintéticos, modelos avançados de detecção e integração embarcada permite transformar UAVs convencionais em sistemas autônomos inteligentes para limpeza de fachadas. Como trabalhos

futuros, sugere-se a expansão do dataset com imagens capturadas diretamente por UAVs, integração com sistemas SLAM para navegação mais refinada e otimizações adicionais para execução embarcada.

A execução de todos os experimentos foi devidamente registrada, e os respectivos vídeos estão disponíveis para acesso [1]. O conjunto de dados está disponível publicamente e pode ser acessado no repositório do projeto [2]. A implementação completa da simulação está disponível publicamente e pode ser acessada através do repositório de simulação do projeto [3].

[1] https://drive.google.com/drive/folders/1TN7hUDsLMf6oiF-GDIC8PhBS90KLsACn
[2] https://github.com/monymon/dataset_glasswindow
[3] https://github.com/monymon/UAV-Window-Detection

REFERENCES

ALPARSLAN, O.; CETIN, O. Fast and effective identification of window and door openings for uavs' indoor navigation. In: **2021 International Conference on Unmanned Aircraft Systems (ICUAS)**. [S.l.: s.n.], 2021. p. 175–181.

ANAC, A. N. de A. C. **Agência Nacional de Aviação Civil ANAC**. 2025. Https://www.gov.br/anac/pt-br/assuntos/drones/regras-de-todos-os-orgaos-brasileiros-sobre-operacao-de-drones. Acesso em: 12 out. 2025.

ASUS. **Asus Tinker Board 2s**. 2025. Https://tinker-board.asus.com/series/tinker-board-2s.html. Acesso em: 12 jul. 2024.

BASSO, M.; FREITAS, E. P. de. A uav guidance system using crop row detection and line follower algorithms. **Journal of Intelligent & Robotic Systems**, February 2019.

BASSO, M.; FREITAS, E. Pignaton de. A uav guidance system using crop row detection and line follower algorithms. **Journal of Intelligent & Robotic Systems**, Springer, v. 97, n. 3, p. 605–621, 2020.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. **Arxiv**, Apr 2020.

BORTOLOTI, F. D. et al. An annotated image database of building facades categorized into land uses for object detection using deep learning. **Machine Vision and Applications**, Aug 2022.

CHEN, M. et al. A uav-based energy-efficient and real-time object detection system with multi-source image fusion. **Journal of Circuits, Systems and Computers**, v. 31, n. 09, p. 2250166, 2022. Available from Internet: <https://doi.org/10.1142/S0218126622501663>.

CHOLLET, F. **Deep Learning with Python**. Shelter Island, NY 11964: Manning Publications Co, 2021.

FENG, A. et al. Yield estimation in cotton using uav-based multi-sensor imagery. **ELSEVIER**, v. 193, February 2020.

FLOREANO, D.; J.WOOD, R. Science, technology and the future of small autonomous drones. **Nature**, Nature Publishing Group UK London, v. 521, n. 7553, p. 460–466, 2015.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2014. p. 580–587.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 4nd. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1992. ISBN 0201508036.

GRANDINI, M.; BAGLI, E.; VISANI, G. Metrics for multi-class classification: an overview. **arXiv preprint arXiv:2008.05756**, 2020.

HABILI, N. et al. A hyperspectral and rgb dataset for building façade segmentation. **Computer Vision - ECCV 2022 Workshops.**, p. 258–267, 2022.

HUANG, B. et al. An unmanned aerial vehicle implemented network for real-time crack detection in stone cladding. **Computer-Aided Civil and Infrastructure Engineering**, v. 40, n. 24, p. 4014–4034, 2025. Available from Internet: <https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.70021>.

INMET. **INMET**. 2025. Https://tempo.inmet.gov.br/TabelaEstacoes/A001. Acesso em: 30 out. 2025.

JABLONOWSKI, M. **Dronie Citizenship?" In Selfie Citizenship**. [S.l.]: Springer Nature, 2017.

KAUFMANN, M. **Drone/Body: The Drone's Power to Sense and Construct Emergencies." In The Good Drone**. London: Routledge, 2016.

KHANAM, R.; HUSSAIN, M. Yolov11: An overview of the key architectural enhancements. **arXiv preprint arXiv:2410.17725**, 2024.

KHANAM, R.; HUSSAIN, M. A review of yolov12: Attention-based enhancements vs. previous versions. **arXiv preprint arXiv:2504.11995**, 2025.

KORČ, F.; FÖRSTNER, W. **eTRIMS Image Database for Interpreting Images of Man-Made Scenes**. [S.l.], 2009. Available from Internet: <http://www.ipb.uni-bonn.de/projects/etrims_db/>.

LEE, H. Y.; HO, H. W.; ZHOU, Y. Deep learning-based monocular obstacle avoidance for unmanned aerial vehicle navigation in tree plantations. **Journal of Intelligent & Robotic Systems**, 2021.

LIU, H. An improved yolo-based algorithm for optical building facade crack detection. In: SPIE. **Second International Conference on Intelligent Transportation and Smart Cities (ICITSC 2025)**. [S.l.], 2025. v. 13682, p. 592–601.

LIU, H. et al. Deepfacade: A deep learning approach to facade parsing with symmetric loss. **IEEE TRANSACTIONS ON MULTIMEDIA**, v. 22, n. 12, December 2020.

LIU, Y. et al. A dual-stream learning framework for weakly supervised salient object detection with multi-strategy integration. **The Visual Computer**, January 2025.

MA, Y. et al. Avs-yolo: Object detection in aerial visual scene. **International Journal of Pattern Recognition and Artificial Intelligence**, v. 36, n. 01, p. 2250004, 2022. Available from Internet: <https://doi.org/10.1142/S0218001422500045>.

MAO, Z. et al. A dataset and ensemble model for glass façade segmentation in oblique aerial images. **IEEE Geoscience and Remote Sensing Letters**, IEEE, v. 19, p. 1–5, 2022.

MARCINóW, A. et al. Building a dataset of wrocław's historic tenements. image annotation for machine learning applications. **Architectus**, Politechnika Wrocławska, 2024.

MEI, H. et al. Large-field contextual feature learning for glass detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 45, n. 3, p. 3329–3346, 2022.

MORAES, R. S. de; FREITAS, E. P. de. Multi-uav based crowd monitoring system. **IEEE Transactions on Aerospace and Electronic Systems**, v. 56, n. 2, p. 1332–1345, 2020.

MORTAZ, E. Imbalance accuracy metric for model selection in multi-class imbalance classification problems. **Knowledge-Based Systems**, Elsevier, v. 210, p. 106490, 2020.

NORDMARK, N.; AYENEW, M. Window detection in facade imagery: A deep learning approach using mask r-cnn. **University of Gothenburg**, July 2021.

PAN, W.; YANG, Z. A lightweight enhanced yolov8 algorithm for detecting small objects in uav aerial photography. **The Visual Computer**, December 2024.

PANDEY, N.; BINGI, K.; IBRAHIM, R. Drone-based imaging and automated classification for glass facade condition monitoring. In: **2025 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)**. [S.l.: s.n.], 2025. v. 1, p. 333–338.

PASANDIDEH, F. et al. A review of flying ad hoc networks: Key characteristics, applications, and wireless technologies. **Remote Sensing**, v. 14, n. 18, 2022. ISSN 2072-4292. Available from Internet: <https://www.mdpi.com/2072-4292/14/18/4459>.

PRINCE, S. J. D. **Computer Vision, Models, Learning and Inference**. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2012.

REDMON, J. et al. You only look once: Unified, real-time object detection. **University of Washington and Allen Institute for AI and Facebook AI Research**, May 2016.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. **Arxiv**, Dec 2016.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **Arxiv**, Apr 2018.

REIS, D. et al. Real-time flying object detection with yolov8. **Arxiv**, May 2024.

ROMBACH, R. et al. High-Resolution Image Synthesis with Latent Diffusion Models . In: **2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. Los Alamitos, CA, USA: IEEE Computer Society, 2022. p. 10674–10685. Available from Internet: <https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.01042>.

SKYDRONES. **Skydrones**. 2025. <https://www.skydrones.com.br>. Acesso em: 12 out. 2025.

SORDO, Z. et al. Synthetic scientific image generation with vae, gan, and diffusion model architectures. **Journal of Imaging**, v. 11, n. 8, 2025. ISSN 2313-433X. Available from Internet: <https://www.mdpi.com/2313-433X/11/8/252>.

SOUSA, A. D. P. d.; SOUSA, G. C. L. d.; MAUéS, L. M. F. Using digital image processing and unmanned aerial vehicle (uav) for identifying ceramic cladding detachment in building facades. **Ambiente Construído**, Associação Nacional de Tecnologia do Ambiente Construído - ANTAC, v. 22, n. 2, p. 199–213, Apr 2022. ISSN 1678-8621. Available from Internet: <https://doi.org/10.1590/s1678-86212022000200601>.

SRIVASTAVA, G.; SRIVASTAVA, R. User-interactive salient object detection using yolov2, lazy snapping, and gabor filters. **Machine Vision and Applications**, v. 31, p. 17, Mar 2020.

TETILA, E. C. et al. Automatic recognition of soybean leaf diseases using uav images and deep convolutional neural networks. **IEEE GEOSCIENCE AND REMOTE SENSING LETTERS**, v. 17, p. 903–907, July 2020.

TIAN, Y.; YE, Q.; DOERMANN, D. Yolov12: Attention-centric real-time object detectors. **arXiv preprint arXiv:2502.12524**, 2025.

TYLECEK, R. **The CMP Facade Database**. [S.l.]: CTU Publishing House, Czech Technical University in Prague, 2012.

VANEGAS, F. et al. A novel methodology for improving plant pest surveillance in vineyards and crops using uav-based hyperspectral and spatial data. **Sensors**, v. 260, January 2018.

VIVEKANANTHAN, S. Comparative analysis of generative models: Enhancing image synthesis with vaes, gans, and stable diffusion. **arXiv**, 2024. Available from Internet: <https://arxiv.org/abs/2408.08751>.

WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: **2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2023. p. 7464–7475.

WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: **2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2023. p. 7464–7475.

WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: **2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2023. p. 7464–7475.

WANG, C.-Y.; LIAO, H.-Y. M. Yolov1 to yolov10: The fastest and most accurate real-time object detection systems. **APSIPA Transactions on Signal and Information Processing**, Now Publishers, v. 13, n. 1, p. e29, 2024.

WANG, D. et al. Uav environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. **Computers and Electronics in Agriculture**, v. 175, 2020.

WANG, F. et al. A deep-reinforcement-learning-based uav autonomous navigation and collision avoidance in unknown environments. **Chinese Journal of Aeronautics**, p. 237–257, June 2023.

WANG, Z. et al. Robust blind image watermarking based on interest points. **Virtual Reality & Intelligent Hardware**, August 2024.

WEI, J. et al. Irregular facades: A dataset for semantic segmentation of the free facade of modern buildings. **Buildings**, MDPI, v. 14, n. 9, p. 2602, 2024.

XIANG, Y. et al. Measuring software modularity based on software networks. **Entropy**, v. 21, n. 4, 2019. ISSN 1099-4300. Available from Internet: <https://www.mdpi.com/1099-4300/21/4/344>.

YANG, J. et al. Distributed cooperative onboard planning for the conflict resolution of unmanned aerial vehicles. **JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS**, October 2018.

YOON, J. et al. Cnn- and uav-based automatic 3d modeling methods for building exterior inspection. **Buildings**, v. 14, n. 1, 2024. ISSN 2075-5309. Available from Internet: <https://www.mdpi.com/2075-5309/14/1/5>.

ZHANG, G.; PAN, Y.; ZHANG, L. Deep learning for detecting building facade elements from images considering prior knowledge. **Automation in Construction**, v. 133, 2022.

ZHANG, H.; PENG, Y.; LIU, Y. li. Uav aerial photography target detection based on improved yolov9. **The Journal of Supercomputing**, Frebuary 2025.

ZHAO, Y. et al. Dsd-matchingnet: Deformable sparse-to-dense feature matching for learning accurate correspondences. **Virtual Reality & Intelligent Hardware**, October 2022.

ZHOU, K. et al. An improved yolov10 algorithm for automated damage detection of glass curtain-walls in high-rise buildings. **Journal of Building Engineering**, v. 101, p. 111812, 2025. ISSN 2352-7102. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2352710225000488>.

ZHU, X. et al. Tph-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios. In: **2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)**. [S.l.: s.n.], 2021. p. 2778–2788.