Refining Runway Instance Segmentation Masks via Post-Processing for Vision-Based Landing Systems

Júlia Santos Moura*, João Paulo Lara Pinto*, João Pedro Klock Ferreira*, Cristiano Leite de Castro* Gabriel Lott*,
*Graduate Program of Electrical Engineering, Universidade Federal de Minas Gerais

***Abstract*—Instance segmentation masks for airport runway-related elements—such as runways, thresholds, and aiming points—often exhibit irregular contours and fail to accurately capture the geometric properties of the target instances. To address this limitation, we propose a model-agnostic post-processing pipeline designed to refine the shapes of these instances using only the predicted masks. Our approach comprises three specialized pipelines tailored to each instance type, followed by alignment and inference stages to ensure geometric consistency and recover missing elements. Inspired by the Smoothing Post-processing Module (SPM) from BARS, our method improves mask shapes across varying conditions, including changes in lighting, rotation, and scale, without requiring access to the internals of the segmentation model. The refined masks maintain high overlap with the originals while exhibiting improved shape regularity, significantly reducing the need for manual annotation. This enables more efficient dataset creation and supports the development of more realistic and accurate vision-based landing systems.**

***Index Terms*—instance segmentation, runway, masks post-processing, plane landing**

I. Introduction

The flight of commercial fixed-wing aircraft is largely autonomous. However, during the landing phase—where most aviation accidents occur—pilot intervention remains essential. Human error is the leading cause of these incidents [1], making landing the most critical phase of flight in terms of safety [9].

Fixed-wing aircraft are equipped with sensor-based systems that provide horizontal and vertical guidance, or even enable global positioning assistance. However, these systems involve substantial installation and maintenance costs. Moreover, they are vulnerable to interference, as their parameters rely on ground-based or satellite signals.

In this context, computer vision-based landing systems have emerged as a cost-effective alternative. With a forward-facing camera mounted in the cockpit, the relative pose of the runway can be inferred from visual cues during the landing phase. This reduce the pilot's workload, as there is no need for continuous manual control and decision-making, allowing them to focus on other aspects of flight safety. Another positive aspect of Vision-based Autonomous Landing System (VALS) is that relying solely on onboard aircraft sensors the need for costly ground-based infrastructure is eliminated, reducing overall costs and increasing safety.

In recent years significant efforts have been made to build runway datasets for use in computer vision-based landing systems. Most of these datasets [2], [4], [3] are generated using flight simulators and require manual or semi-automatic annotation. Since computer vision solutions typically require large volumes of labeled data, annotating these images becomes a time-consuming and labor-intensive task. Furthermore, the lack of standardization across annotation formats complicates research and limits dataset interoperability.

The performance of computer vision-based landing systems is highly dependent on the quality of the predicted segmentation masks. Although instance segmentation models have shown significant improvements, the resulting masks often exhibit uneven edges and fail to accurately capture the geometric properties of the target instances, as shown in Figure 1. To address these limitations, we propose a post-processing pipeline specifically designed for masks of airport runway-related instances. Our method operates solely on the predicted masks and enhances the geometric accuracy of contour-based annotations for three distinct instance types: runways, thresholds, and aiming points, as illustrated in Figure 2. This pipeline aims to reduce the need for manual annotation, accelerate dataset creation, and enable researchers to focus on improving model realism and performance rather than on time-consuming labeling tasks.

Fig. 1. Example of a predicted segmentation mask with irregular edges, highlighting limitations in geometric accuracy.

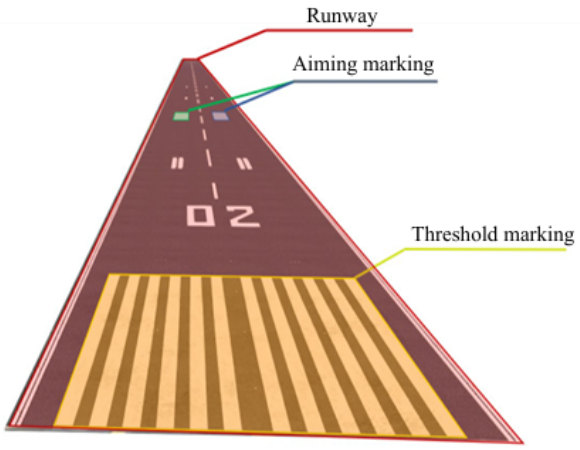

Fig. 2. Three instances types: runway, aiming marking/points and threshold. - Figure from [2]

II. Related Works

The **Benchmark for Airport Runway Segmentation (BARS)** [2] is one of the most comprehensive publicly available datasets in this domain. It contains 10,256 images of airport runways from a cockpit perspective, generated using the X-Plane simulator. The dataset encompasses runway images from various airports, aircraft views, weather conditions, and time intervals. Annotations are provided in the LabelMe [8] format for runway, aiming points, and threshold instances.

BARS employs a semi-automatic pipeline for labeling. All images of a specific runway are grouped, and a single image is manually labeled to capture all instances. Since the instances in real life appear as rectangles from a top-down perspective, a rotation matrix is computed based on this reference image, and the proportions of the instances are calculated. In the second stage, only one instance needs to be localized by an instance segmentation model. With the known perspective matrix and preserved proportions, the other instances can be accurately inferred. Although this method is effective, it requires prior knowledge to group images by runway, which is impractical for AI-generated datasets that lack consistent runway identifiers. Moreover, each new runway still requires manual annotation.

The **Runway Landing Dataset (RLD)** [3], also based on X-Plane, contains 12,239 images, including a subset sourced from real YouTube flight footage. It features 30 airports across all six continents. Annotations—created using LabelMe—are limited to the runway instance, which restricts its use in fully automatic landing systems that require aiming points and threshold information for precise trajectory control.

The **FS2020 Dataset** [4], created using Microsoft Flight Simulator 2020 (FS2020), comprises 5,587 images. It is notable for offering two types of annotations: instance segmentation masks (for the runway, threshold, and aiming points) and line annotations (2D coordinates) for six categories: left edge (LEDG), right edge (REDG), center line (CTL), aiming point front (AimP), threshold rear (THR), and PAPI lights (PAPI). Although the dataset provides a rich set of annotation categories, it contains a relatively small number of images compared to other datasets and lacks any documented methodology for extension. This limitation hinders its usability and adaptability in broader research contexts.

The **LARD Dataset** [5] adopts a distinct methodology by utilizing Google Earth Studio to simulate flight trajectories from a cockpit perspective, thereby generating synthetic images of real-world runway instances. The annotations, which consist exclusively of the runway's corner points, are automatically produced by the same script that defines the trajectory. However, this type of annotation is highly sensitive to occlusions and can introduce significant instability in aircraft pose estimation. This sensitivity is a key reason why such corner point annotations are not employed in the previously discussed datasets. As a result, the uniqueness of LARD's annotation format hinders direct comparison between methods trained or evaluated on this dataset and those using other benchmarks.

All of these datasets provide valuable contributions to the task of automatic landing. However, several challenges remain, including the annotation of large-scale data, inconsistent annotation formats, and varying definitions of instances across datasets. Moreover, most datasets still require extensive manual labeling, which is time-consuming and labor-intensive. As instance segmentation models continue to improve, the idea of automatically annotating datasets becomes increasingly feasible. To support this, a robust post-processing framework is essential—one that can transform the often irregular masks produced by these models into clean, geometrically accurate shapes. Additionally, since these datasets are typically developed for use in flight automation pipelines, improvements in the quality and structure of the final segmentation masks directly enhance their utility in such applications.

III. Proposed Framework

Segmentation masks of runway-related instances often exhibit irregular contours that fail to accurately represent the underlying geometric structures. To overcome this limitation, we propose a post-processing framework consisting of three specialized pipelines, each tailored to a specific instance type: (1) entire runway, (2) threshold, and (3) aiming points. These instances are initially processed independently. After it, we align the runway and threshold masks to ensure geometric consistency. Finally, in the final stage, we infer missing aiming points when only one instance is detected. The complete post-processing pipeline is illustrated in Figure 3. Our approach draws inspiration from the Smoothing Post-processing Module (SPM) introduced in BARS [2], which utilizes spatial moments to estimate the centroid of the detected instance and applies linear fitting to enhance the geometric accuracy of the runway mask.

A. Runway Post-Processing

The first stage, illustrated in Figure 4, involves the following

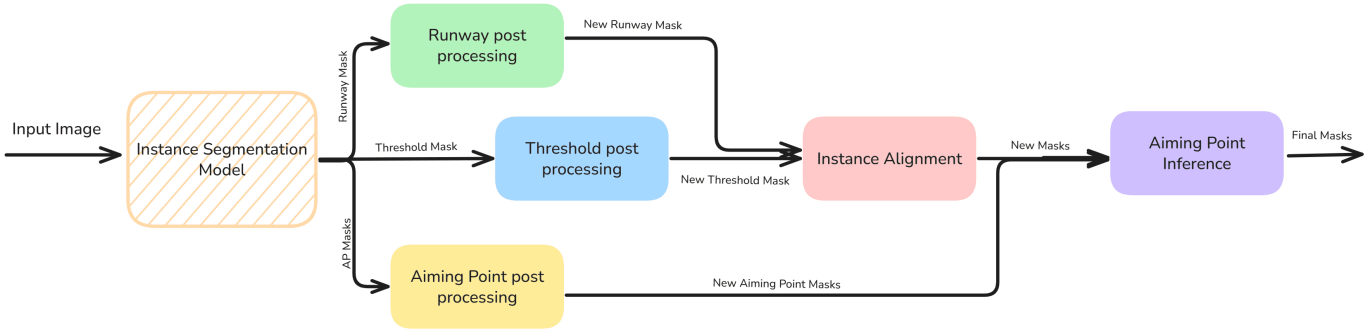

Fig. 3. Overview of the complete post-processing framework.

steps:

a. **Morphological Opening:** Apply three iterations of morphological opening to remove noise:

$$\text{Opening}(M, B) = (M \ominus B) \oplus B$$

where M is the mask, B is the structuring element, and $\ominus$, $\oplus$ represent erosion and dilation, respectively.

b. **Contour and Moments:** Extract the outer contour and compute raw spatial moments:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

where $f(x, y)$ is the binary mask value. The centroid (c_x, c_y) is obtained as:

$$c_x = \frac{m_{10}}{m_{00}}, \quad c_y = \frac{m_{01}}{m_{00}}$$

Based on this centroid, we define four points above and four below, spaced by $\frac{1}{8}$ of the image height. The intersections of horizontal lines through these points with the contour are averaged to create nine reference points for fitting a centerline via linear regression.

c. **Quadrant Classification:** Classify contour points into four quadrants based on their relative positions to the centroid and the centerline.

d. **Corner Extraction:** Identify four key corners by selecting extreme points in each quadrant: top-left (min x, min y), top-right (max x, min y), bottom-left (min x, max y), bottom-right (max x, max y).

e. **Side Fitting:** Assign contour segments to each side. Fit linear regression to the bottom edge and adjust bottom y coordinates using the most extreme x values.

B. Threshold Post-Processing

Thresholds often exhibit striped textures that confuse models, leading to misclassification as runway edges. Our approach, shown in Figure 5, parallels the runway pipeline, with key differences:

- Contour quadrants are assigned using only the centroid, due to the threshold's small vertical extent.

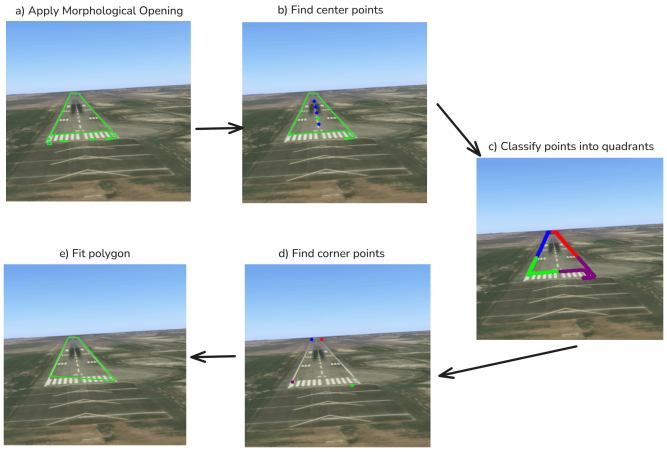

Fig. 4. Runway post-processing pipeline.

- Based on these quadrants, we identify corners and classify side points.
- We then fit linear regressions to the top and bottom edges and reconstruct the corners using the resulting endpoints.

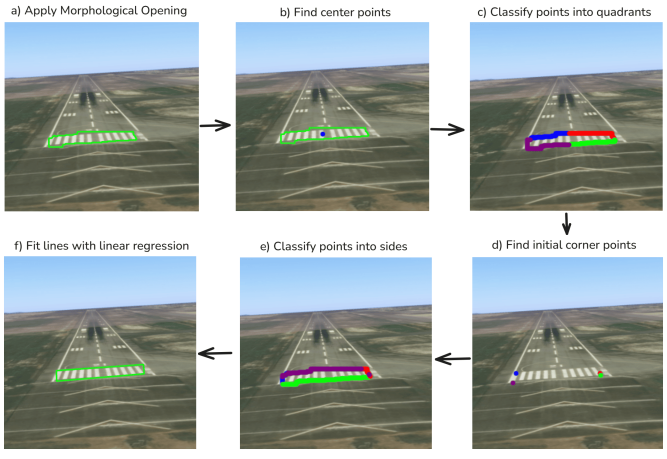

Fig. 5. Threshold post-processing pipeline.

C. Aiming Point Post-Processing

Aiming points are typically small rectangles and require minimal post-processing. We perform:

- Morphological opening to denoise the mask.
- Contour extraction followed by simplification using the Douglas-Peucker algorithm [7], constrained to retain four vertices.

D. Instance Alignment

To ensure geometric consistency, we enforce parallelism between the top and bottom edges of both the runway and threshold, as expected from a top-down aerial view. We compute angular coefficients (slopes) for each edge and filter outliers that deviate more than twice the standard deviation from the mean. The average slope is used for final alignment, as outlined in Algorithm 1.

Algorithm 1 Unify Runway and Threshold Masks Ensuring Geometric Consistency

Input: `runway_corners`, `threshold_corners` (4 points each)
Output: `new_runway_corners`, `new_threshold_corners`
Compute slopes a for top and bottom edges of both instances
Discard NaN or infinite slopes
if no valid slopes **then**
    $a \leftarrow 0$
else
    Compute mean and standard deviation
    Filter values within 2 standard deviations
    $a \leftarrow$ mean of filtered slopes
end if
Update bottom coordinates:
$bl_y = \max(y_{\text{runway\_bl}}, y_{\text{threshold\_bl}})$
$br_y = \max(y_{\text{runway\_br}}, y_{\text{threshold\_br}})$
$bl_x = \min(x_{\text{runway\_bl}}, x_{\text{threshold\_bl}})$
$br_x = \max(x_{\text{runway\_br}}, x_{\text{threshold\_br}})$
Adjust using slope:
$bl_y \leftarrow a \cdot bl_x + bl_y$
$br_y \leftarrow a \cdot br_x + br_y$
Construct `new_runway_corners` using original top and updated bottom points
Compute runway top corner angles $\theta_{left}, \theta_{right}$
Adjust threshold corners to align with these angles
if threshold width $\geq 0.8\times$ runway width **then**
    Use threshold bottom corners in runway polygon
end if
return `new_runway_corners`, `new_threshold_corners`

E. Aiming Point Inference

To eliminate false positives, we retain only one aiming point per runway half, defined by the centroid of the processed runway. Among multiple candidates, we choose the one with the highest vertical coordinate, since aiming points typically occur farther from the runway end.

When only one aiming point is detected, we infer the second based on geometric symmetry. We measure its distance and size relative to the nearest runway edge and replicate it on the opposite side. Its vertical placement is corrected using the runway's angular coefficient. The result of the full post-processing pipeline is illustrated in Figure 6.

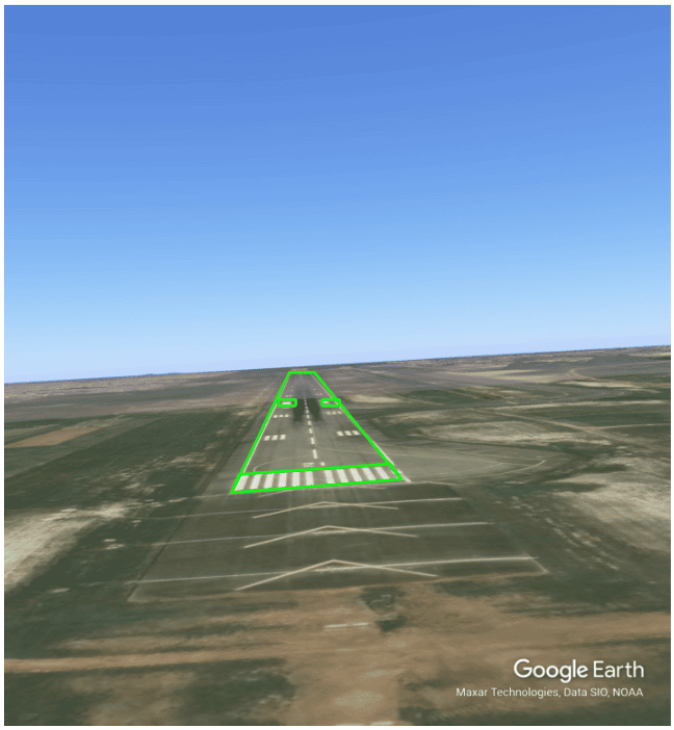

Fig. 6. Contours of all instances after final post-processing.

IV. Experimental Setup and Results

To assess the effectiveness of our approach, we trained the YOLOv11-seg model [6] for 300 epochs using a batch size of 16. We selected the FS2020 dataset [4], which is publicly available and includes all three key categories—runway, threshold, and aiming point—distributed across 5,587 images.

The original FS2020 annotations are provided as color-coded masks, where each instance is represented by a distinct color. To adapt these annotations for the YOLOv11 instance segmentation model, we converted the masks into YOLO-compatible text format through the following preprocessing pipeline (illustrated in Figure 7):

- For each class, a binary mask was generated by mapping the target class's color to white and all others to black.
- Morphological closing was applied to eliminate holes and connect disjoint regions.

- The outer contours of the binary mask were extracted.
- The normalized coordinates and class labels were saved in YOLO-compatible `.txt` files.

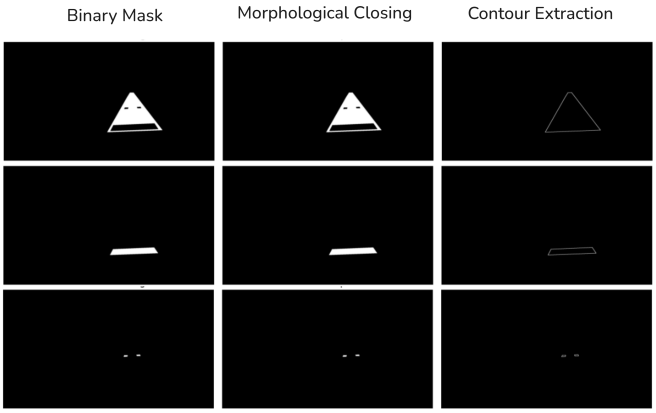

Fig. 7. Illustration of the preprocessing pipeline: from raw mask to contour extraction for runway, threshold, and aiming point instances.

To increase the number of annotated instances, we also incorporated the BARS dataset [2]. Due to regional restrictions with Baidu, we only had access to its validation and test splits (available via GitHub). These subsets were merged, shuffled, and re-split into new training and testing sets. As BARS annotations are already provided in polygon format (JSON), we directly converted them to the YOLO format without further preprocessing.

The final training set comprised 4,987 images, and the test set included 1,850 images across all three categories. Training was performed on a system equipped with an NVIDIA GeForce RTX 4070 Ti GPU and an AMD Ryzen 9 7900X 12-core CPU. Table I reports the model's performance in terms of precision (P), recall (R), and mAP (IoU thresholds 0.5 and 0.95) for both bounding box and mask predictions.

TABLE I
PERFORMANCE METRICS BY CLASS FOR BOUNDING BOX AND MASK PREDICTIONS.

Class	Instances	Type	P	R	mAP50/95
Threshold	628	Box	0.885	0.699	0.786 / 0.611
		Mask	0.802	0.565	0.664 / 0.394
Aiming Point	1228	Box	0.909	0.690	0.779 / 0.619
		Mask	0.814	0.567	0.673 / 0.378
Runway	3696	Box	0.951	0.975	0.973 / 0.865
		Mask	0.952	0.959	0.960 / 0.709

The qualitative results of the post processing pipeline are presented in Figure 8. The first column shows ground truth annotations (from BARS or FS2020), the second displays the YOLOv11-seg predictions, and the third presents our post-processed output. The pipeline demonstrates robust generalization under diverse conditions, including varying lighting, runway orientations, and distances. It is also capable of detecting previously unannotated runways, inferring missing aiming points, and correcting incorrect annotations—such as the first example, where the ground truth annotation improperly extends beyond the paved surface.

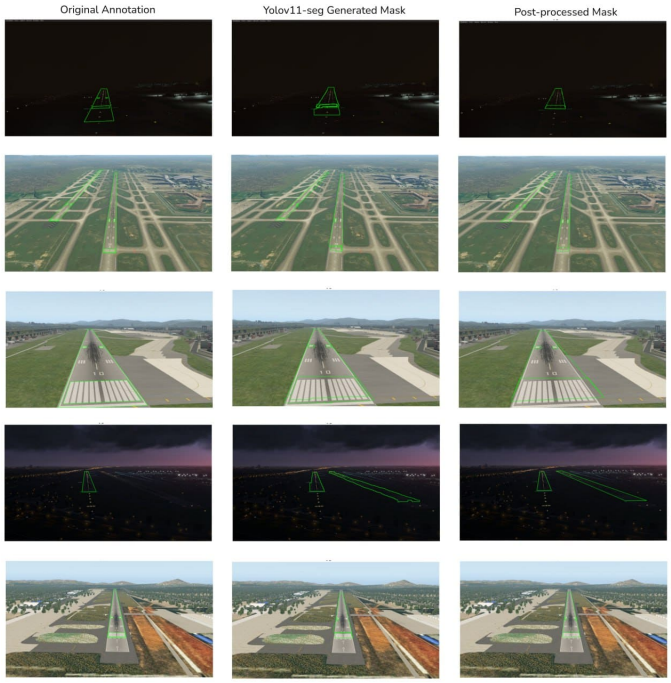

Fig. 8. Comparison of original annotation, YOLOv11-seg prediction, and post-processed output under different scenarios.

To quantitatively evaluate the post-processing pipeline, we computed the Intersection over Union (IoU) between predicted masks and ground truth annotations (from BARS and FS2020) across the entire validation set of 1,850 images. For images with multiple runways, each instance was evaluated independently.

In some cases—especially when runways appear small due to distance—even slight changes in mask geometry result in substantial IoU penalties. Figure 9 exemplifies such a scenario, where only the right-side runway is originally annotated. While the YOLO-predicted mask achieved a 74.91% IoU, the refined post-processed mask obtained 59.07%, as improvements in shape increased the union without improving the intersection.

To better capture model performance in such cases, we introduce a new metric—**Weighted IoU**—which accounts for object size. We apply a softmax function over the areas of ground truth masks, giving more weight to larger objects:

$$\text{WeightedIoU} = \sum_{i=1}^{n} \text{IoU}_i \cdot \frac{e^{A_i}}{\sum_{j=1}^{n} e^{A_j}} \tag{1}$$

Where:

- n is the number of valid instances;
- IoU_i is the IoU of the i-th instance;
- A_i is the area of the i-th ground truth mask;
- $\frac{e^{A_i}}{\sum_{j=1}^{n} e^{A_j}}$ is the softmax weight.

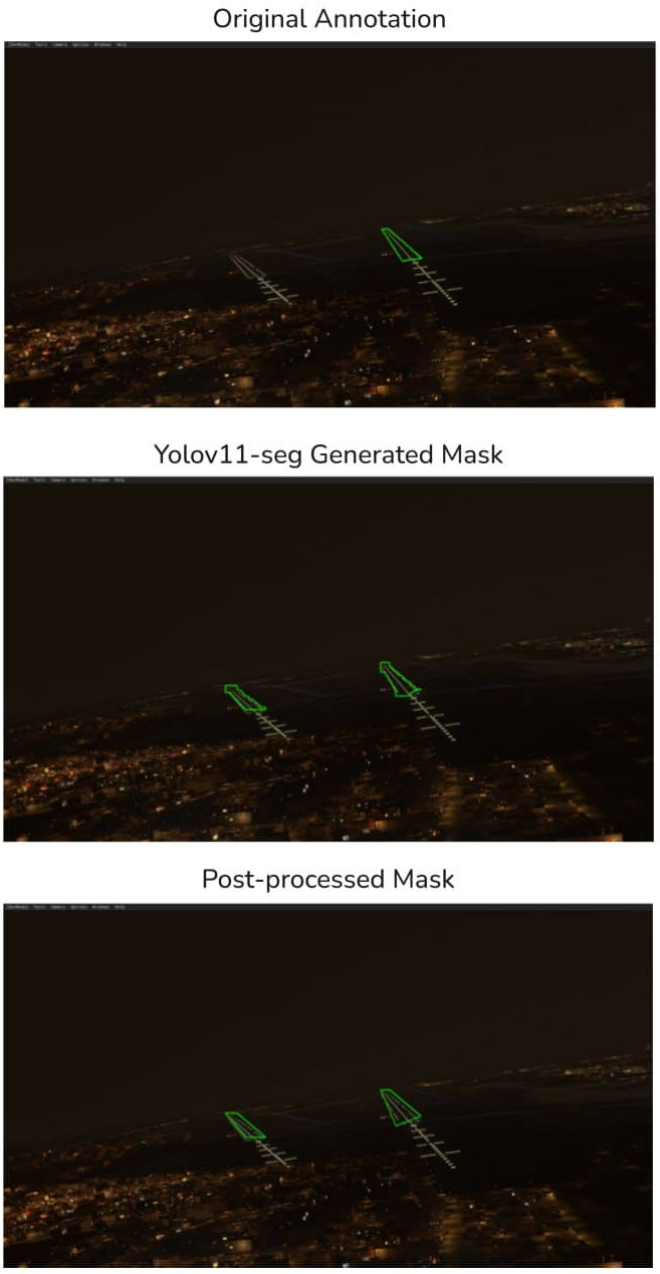

Fig. 9. IoU comparison between YOLOv11-seg prediction and post-processed mask in a challenging case with small target area.

Figure 10 provides a schematic comparison between standard IoU and Weighted IoU to aid interpretation. Table II reports the average and weighted IoU scores per class, comparing the original YOLOv11 predictions with our post-processed masks. While there is a trade-off between geometric refinement and raw IoU/Weighted IoU values, the results indicate that our pipeline preserves the overall intersection and union quality relative to the original predictions. Moreover, the Weighted IoU offers a more reliable measure of how well the masks represent the actual data, especially in scenarios involving objects of varying sizes.

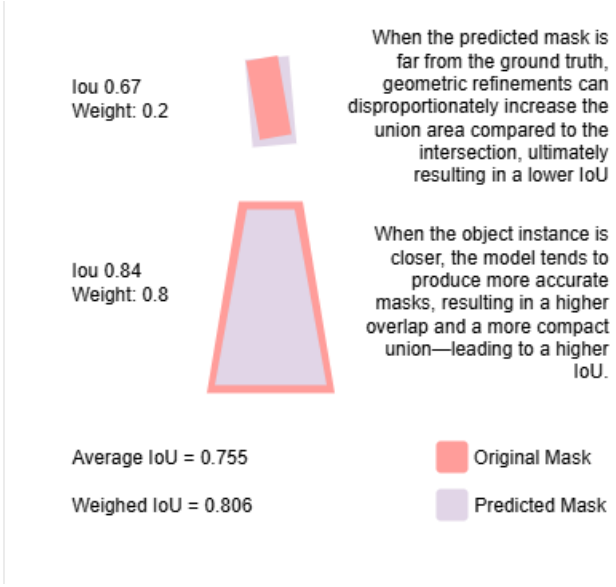

Fig. 10. Schematic comparison of IoU and Weighted IoU

TABLE II
AVERAGE AND WEIGHTED IOU SCORES BY CLASS FOR YOLOV11 AND POST-PROCESSED MASKS.

Metric	Method	Runway	Threshold	Aiming Point
IoU	Post-processed	0.6383	0.1993	0.2257
	YOLOv11	0.6591	0.2173	0.2396
Weighted IoU	Post-processed	0.9209	0.7830	0.7113
	YOLOv11	0.9304	0.6680	0.8344

A common trade-off is observed between mask realism and quantitative accuracy. While our method produces geometrically refined masks, they may lead to slightly lower raw IoU scores due to stricter polygonal constraints. Nevertheless, our post-processing approach—particularly for thresholds and aiming points—leverages contextual cues from the predicted runway, which enables improvements in some challenging cases where YOLOv11 alone produces poor results.

Figure 11 illustrates the weighted IoU of post-processed and YOLOv11 masks across 10 bins of increasing average IoU values, helping to visualize performance differences across prediction confidence levels.

V. CONCLUSION

We presented a post-processing pipeline capable of improving the geometric accuracy of runway, threshold, and aiming point instances using only the predicted instance masks, while maintaining a high IoU with the original masks. The approach is robust across varying conditions, including day and night imagery, different camera angles, and distances. Since the post-processing steps are model-agnostic and rely solely on the provided masks, researchers can freely integrate different segmentation models or refine existing ones without altering the pipeline. By significantly reducing the manual effort required to refine generated masks, our method enables researchers

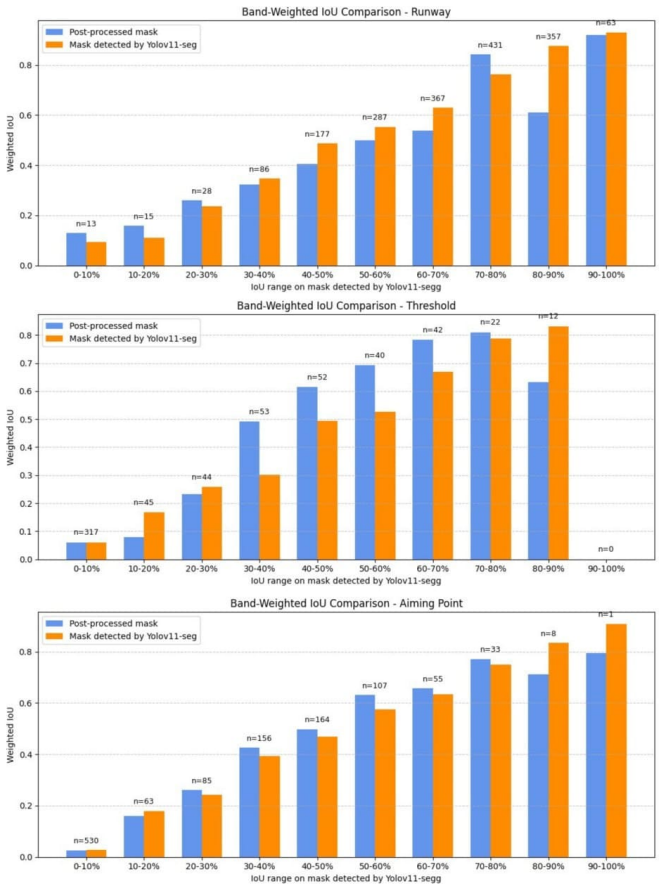

Fig. 11. Weighted IoU comparison across 10 bins of increasing instance-level IoU scores for YOLOv11 and post-processed masks.

to focus on developing more realistic datasets and advancing instance segmentation techniques for aviation applications.

VI. Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

VII. Affiliation

Graduate Program of Electrical Engineering

References

[1] S. Shappell, C. Detwiler, K. Holcomb, C. Hackworth, A. Boquet, and D. A. Wiegmann, "Human error and commercial aviation accidents: An analysis using the human factors analysis and classification system," *Human Factors*, vol. 49, no. 2, pp. 227–242, Apr. 2007.

[2] W. Chen *et al.*, "BARS: a benchmark for airport runway segmentation," *Applied Intelligence*, vol. 53, no. 17, pp. 20485–20498, 2023.

[3] Q. Wang *et al.*, "VALNet: Vision-based autonomous landing with airport runway instance segmentation," *Remote Sensing*, vol. 16, no. 12, pp. 2161, 2024. doi: 10.3390/rs16122161.

[4] M. Chen and Y. Hu, "An image-based runway detection method for fixed-wing aircraft based on deep neural network," *IET Image Processing*, vol. 18. Mar. 2024. doi: 10.1049/ipr2.13087.

[5] M. Ducoffe *et al.*, "LARD—Landing Approach Runway Detection—Dataset for vision-based landing," *arXiv preprint arXiv:2304.09938*, 2023.

[6] G. Jocher and J. Qiu, "Ultralytics YOLO11," version 11.0.0, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[7] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[8] A. Torralba, B. C. Russell, and J. Yuen, "LabelMe: Online image annotation and applications," *Proceedings of the IEEE*, vol. 98, no. 8, pp. 1467–1484, 2010.

[9] Airbus, "A statistical analysis of commercial aviation accidents 1958–2023," Airbus S.A.S., Blagnac, France, Feb. 2024. [Online]. Available: https://accidentstats.airbus.com/wp-content/uploads/2024/02/20230873_A-Statistical-analysis-of-commercial-aviation-accidents-2024-version.pdf