Mini Drone Race – The Perception, Planning, And Control Saga!

Team Nimbus Navigators

Chaitanya Sriram Gaddipati
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: cgaddipati@wpi.edu

Ankit Talele
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: amtalele@wpi.edu

Shiva Surya Lolla
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: slolla@wpi.edu

***Abstract*—In this project, we developed a sophisticated perception stack for DJI Tello EDU quadcopter to enable precise navigation through multiple windows, whose 3D poses were approximately known in advance. To achieve this, we employed simulation-to-reality (sim2real) techniques to generate synthetic data, creating a robust training set for our neural network model. The chosen architecture, YOLOv8, was trained to identify and segment the front window from a complex environment of multiple windows. Once the segmentation mask was detected, we extracted the corners and applied Perspective-n-Point (PnP) algorithm to calculate the relative pose of the front window which is essential for guiding the quadcopter through the windows safely.**

I. Introduction

Autonomous navigation of quadcopters demands a robust perception stack for effectively sensing and understanding the environment, avoiding obstacles, planning paths, and executing precise movements. In our project, we focused on building the perception pipeline of the DJI Tello EDU quadcopter for its navigation through multiple windows in an environment where the approximate 3D poses of these windows were known as apriori.

The core challenge for the quadcopter is to detect and navigate through the correct window when presented with several similar options, which necessitates advanced recognition abilities. To address this, our approach harnessed the power of simulation-to-reality (sim2real) transfer, where synthetic data generated using Blender and data augmentation resulted in a diverse and robust dataset. This dataset was instrumental in training the state-of-the-art neural network model, YOLOv8, for the segmentation of the target window against a backdrop of multiple potential distractors.

Once the target window was obtained, corners were inferred from it by using computer vision techniques like contour detection, erosion, and dilation. Subsequently, these features along with the approximate 3D locations of the window's corners were processed using the Perspective-n-Point (PnP) algorithm to calculate the window's pose relative to the quadcopter for its effective navigation through the window.

The subsequent sections highlight our methodology to achieve our objectives. Our project's success showcases the potential of sim2real methodologies in refining neural networks, which could significantly enhance the autonomous navigation systems of quadcopters.

II. Data Generation

To train a neural network for window segmentation, we needed a dataset of images and labels consisting of windows in an environment to train the network.

A. Design decisions

1) We had the choice of either choosing window corners or the window segmentation mask to be the output of our network. We chose to predict the entire window mask as the output because occluded corners would make it difficult for the network to detect them if we had chosen corners as the output. Even if the segmentation mask of the window was not perfect, we were of the opinion that we could use image processing techniques to refine the mask and infer corners from it.
2) The other design choice was choosing between two potential approaches to window mask detection. The first involved predicting a combined mask for all windows present in the environment and subsequently identifying the front window. This would have necessitated a two-network architecture: one network to detect all window masks and a subsequent network, or a set of computer vision algorithms to single out the front window.

However, upon careful consideration, we determined that a single-network solution directly predicting the front window mask from the environment image was more efficient. This decision relied on our ability to compile a sufficiently diverse and comprehensive dataset to effectively train the network. By doing so, we successfully streamlined the system, eliminating the need for a second network and additional processing stages. This not only simplifies the architecture but potentially reduces inference time. The success of this approach relies on our dataset's quality, which we curated to ensure the

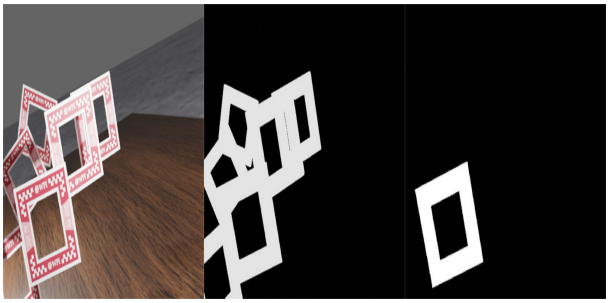

Fig. 1. (Left to Right): 1. Frame 2. Full segmentation mask 3. Front window mask

network's ability to discriminate the front window amidst similar structures.

B. Data generation using blender

For the project, we developed an automated data generation pipeline using Blender, with a custom script to control rendering settings, camera and lighting placement, and the manipulation of window objects within the scene. Six windows were modeled to specific aspect ratios, and the script varied the lighting by adjusting the intensity of two light sources. Additionally, the camera pose was dynamically changed through the script to capture the scene from a range of angles and positions. The blender setup can be seen in fig 7 and the compositing node tree is in fig 8.

This procedure generated a diverse dataset of 4600 rendered images, which were saved to designated directories. This collection of images, featuring varying lighting conditions and perspectives, is essential for training a robust neural network for window detection. An example set of images generated is shown in fig 1.

To create accurate ground truth labels for the neural network, we utilized Blender's Pass Index feature, which allowed us to extract segmentation masks of the front window. This was particularly useful in complex scenes with multiple windows, ensuring the network would be trained on the correct window mask. This comprehensive dataset, complete with precise segmentation masks, provides a strong foundation for the neural network to learn from and perform effectively in window detection tasks.

III. Window detection

As we had our data now, we had to train a neural network using our data for window mask inference.

A. Design decision

Our deep learning model had to be lightweight, fast, and accurate so as to run smoothly on our NVIDIA Jetson Orin Nano. We considered the Segment Anything Model (SAM) and YOLOv8 models as they were instance segmentation models and could segment out objects from the environment. We chose YOLOv8 because of its better real-time inference capability.

B. Training

A Python script was utilized to convert segmentation masks into YOLOv8-compatible labels. It processes binary mask images, extracts contours, and normalizes these into polygon coordinates relative to image size. Each mask's coordinates are saved as a text file with a class identifier and contour points, creating a formatted dataset ready for YOLOv8's object detection training.

Out of the 4600 images, we split the dataset into 4500 images for training and 100 images for testing. We could see that the loss decreased with every epoch of training.

We wanted to test the inference on our local machine first. So initial tests on local machines yielded successful inference with the model capable of generating masks for all test images, albeit with some imperfections such as unwanted patches and holes. We were confident however that we could use image processing techniques to infer corners from them.

C. Generalization

The principal challenge encountered with our model was the lack of generalization to diverse test cases. Although the network worked well on some images it failed most of the times when tested on real world data. This issue likely stems from the training dataset being only made up of images rendered on a single Blender environment without much warping of the window. To enhance the model's performance, we needed to diversify our training data to better mimic the varied angles and environments that a drone would encounter. Employing data augmentation techniques on window images is a viable solution to enrich our dataset without incurring the significant computational costs associated with rendering additional images in Blender.

IV. Data augmentation and retraining

Fig. 2. A small set of real world data with manual labels

In the initial stage of our dataset augmentation, we leveraged approximately 100 single-window images, each paired with its mask, created using Blender. This setup allowed us to simulate diverse lighting conditions and camera angles, thereby mimicking the complexity of real-world scenarios.

To further enrich the dataset, we included real-world single-window images with varying degrees of occlusion (see fig 2). Unlike the synthetic Blender data, these images required manual annotation to generate accurate window masks. For this, we utilized CVAT, a versatile data annotation platform, to compensate for the lack of Pass Index functionality available in Blender.

The heart of our pipeline is the perspective transformation: by applying calculated distortions to window contours, we created a suite of images that mirror the challenging perspectives a drone would face in its operational setting. This variety in training data is critical for developing a model that can reliably identify windows from any angle or distance.

We then used the alpha matting technique for a seamless blend of the windows into five different backgrounds, creating composite images that maintain realism. These processes collectively contribute to the creation of a rich and diverse dataset. The complete pipeline can be seen in fig 3.

Integrating these synthetically generated images with our pre-existing dataset resulted in a robust collection of 14,100 image and mask pairs. Subsequent retraining of our YOLOv8 network with this comprehensive dataset for 10 epochs yielded excellent results. The network demonstrated remarkable performance, accurately detecting window masks in all tested images and exhibiting significant generalization capabilities when used in real-world scenarios. The loss metrics are plotted at each epoch in fig 9. The network when run on the jetson orin nano had an inference times around 33ms which is sufficient and prevents the need to export the model to tensorrt or onnx formats. One observation made while implementing the network on jetson nano is the need to carefully install the proper versions of all the necessary libraries for it to work.

In conclusion, our data augmentation pipeline is instrumental in introducing realistic variability. The strategic implementation of perspective transformations and environmental compositing has been critical in developing an advanced window detection model that promises high efficiency in practical applications.

V. Corner inference

During the post-processing phase of our window detection algorithm, we encountered a challenge with the preliminary rectangular masks generated by the network. These masks were full of rough edges and presented sporadic patches, resembling holes, which impeded direct corner detection using traditional methods such as the Harris Corner Detection algorithm available in OpenCV. To resolve this, we employed a series of morphological operations — specifically erosion and dilation. These operations were instrumental in refining the masks by eliminating the undesired patches and smoothing the edges. Once the masks were preprocessed, we harnessed the findContours function in OpenCV to delineate the periphery of the windows accurately. Subsequently, we utilized the approxPolyDP function, which enabled us to approximate the contours to a polygon and in doing so, accurately identify and demarcate the corners of the rectangular masks. In fig 4 the masks generated without any post-processing can be seen along with the corners detected for a sample frames obtained from the drone in flight.

VI. Calibration

Before performing pose estimation, the drone camera needs to be calibrated to get the camera intrinsic matrix and distortion coefficients. We used a series of checkerboard images, taken from multiple angles and distances, to map the camera's intrinsic parameters. These parameters include the camera's lens properties and its spatial orientation in relation to the observed scene. The calibration results can be seen in fig 5.

A. Camera Matrix

The camera matrix we obtained contains intrinsic parameters that are used for the geometric interpretation of images. This matrix enables us to convert three-dimensional coordinates into two-dimensional ones through perspective projection and is tailored to our drone camera.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$K = \begin{bmatrix} 917.35271 & 0 & 480.97134 \\ 0 & 917.10434 & 365.57078 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

B. Distortion Coefficients

These coefficients indicate the lens distortion present in our camera. Correcting for radial and tangential distortions is necessary for better pose estimation. With these coefficients, we can ensure that the images we capture are a true representation of the environment.

$$D = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3}$$

$$D = \begin{bmatrix} 0.02456 & -0.59580 & -0.00039 & -0.00017 & 1.84865 \end{bmatrix} \tag{4}$$

VII. Pose estimation using PnP

With corners inferred the pose of the window relative to the drone is estimated. We implemented Perspective-n-Point (PnP) pose computation using solvePnP function of OpenCV to do this. The solvePnP estimates the object pose given a set of 3D object points, their corresponding 2D image projections, the camera intrinsic matrix and the distortion coefficients. We have the 2D corners of the window that we had inferred. We also have the camera intrinsic matrix and the distortion coefficients from our calibration process. We had to determine the object point locations in 3D so as to use solvePnP.

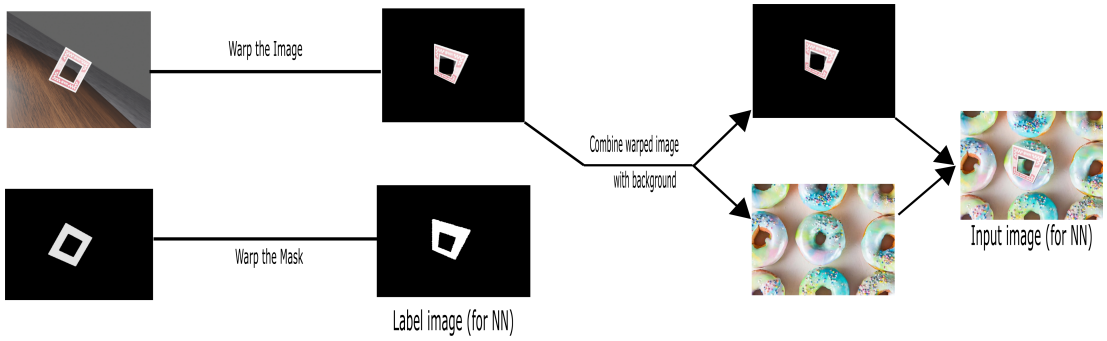

Fig. 3. Data augmentation pipeline to generate more diverse data

Fig. 4. Row 1: input image frames; Row 2: YOLOv8 prediction mask Row 3: Inferred corners after post-processing

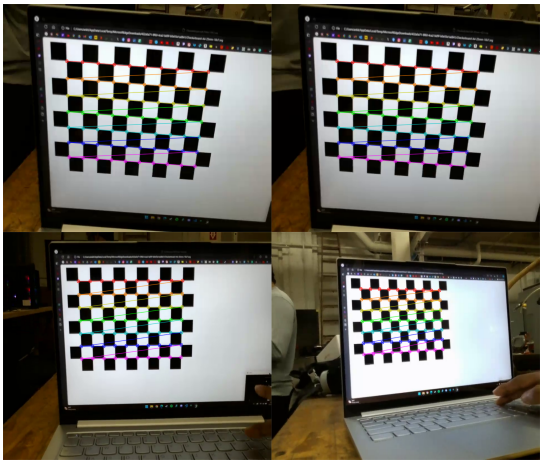

Fig. 5. Calibration on checkerboard viewed by the drone camera from different angles

A. Design decision

Our main goal is to determine the center location of the window relative to the quadcopter for us to use the `go_xyz_speed` command of DJI Tello to navigate the drone to the center location. To get the 3D locations of all the corners a real world coordinate system is placed at the center of the window as shown in fig 10. Approximate length and breadth of the window are known apriori so the 3D coordinates of the window can easily be obtained. **This deliberate positioning is critical as it allows for the direct extraction of the window's center location from the solvePnP's output pose: which is the translation vector ($tvec$) of the pose information that encapsulates the relative positioning of the drone with respect to the window.**

Once the PnP problem is solved a rotation vector and translation vector is obtained that can be used to estimate the pose of drone with respect to the window. The results after applying PnP to frames in fig 4 is shown in fig 6.

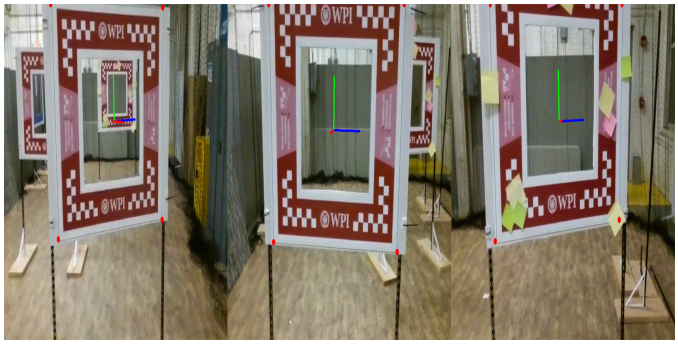

Fig. 6. Window center pose obtained from solving PnP problem

VIII. Planning and Control

Now that the perception stack is working properly and the drone is able to give the relative pose of the window center in 3D. The next task is to implement a method to robustly navigate a given map with multiple windows from the start to goal location while passing through the windows in a desired order.

To do this we are give an approximate map of the environment. Using this map initially we tried to go to a defined waypoint from the start location, identify the window center, and go through it, and next repeat this process of going to a given waypoint and identifying the center for rest of windows. But this method is not reliable at all. This is because our window center estimation from PnP algorithm is not very accurate

since we are only using a single image to identify the pose. To overcome this issue we followed an averaging approach as shown in fig 10. We have three different coordinate frames in our problem as shown in fig 10. $X^{world} - Y^{world} - Z^{world}$ is a world frame present at the start of the environment and the approximate ground truth map of the environment is given in this frame. So the center of the first window in this frame is $[X^{groundtruth}, Y^{groundtruth}, Z^{groundtruth}]$.
Next we have the $X^{tello} - Y^{tello} - Z^{tello}$ drone frame fixed on the drone and all the positional control commands given through the `go_xyz_speed` command are given in this frame relatively. The third is the $X^{window} - Y^{window} - Z^{window}$ frame that we placed at the window center for obtaining the 3D points in PnP problem. Once the PnP problem is solved we get the rotation and translation vectors ($[Rvec, tvec]$) as shown in fig 10. Since the center of the window is at origin in window frame the translation vector can directly be used to get the location of the drone in this frame. By reordering the translation vector from $[t_x, t_y, t_z]$ to $[t_z, t_y, t_x]$ we can convert it to the tello frame. As described initially this alone is not sufficient for us, so the relative groundtruth of the window center is calculated in tello frame from $[X^{groundtruth}, Y^{groundtruth}, Z^{groundtruth}]$ values and simple vector algebra. This is represented as Pos^{gt}_{rel} in fig 10. An average of $[t_z, t_y, t_x]$ and Pos^{gt}_{rel} is taken to get better estimate of window center with respect to the drone location. This value is sent to the drone to cross a window with an additional depth added to pass the window.

IX. Conclusion

In this project we were able to demonstrate that a robust deep learning neural net for front window segmentation can be trained using the synthetic data generated from blender and augmenting it through domain randomization. A sim2real transfer is done by using this network on a DJI Tello drone to navigate through windows in a given map.
Please use the following links to look at videos of successful run of the drone in different views: link1: drone POV, link2: Camera man view.
Some of the challenges faced across the project were discussed thoroughly in the report. Additionally since the odometry of the Tello drone is not perfect some of the runs are unsuccessful because the drone won't accurately go to the specified coordinates. This is a hardware limitation to which the solutions should further be explored to create more reliable trajectories. Overall the performance of the drone is good and it passes through all the windows present in the map.

References

[1] Training Yolov8 on custom datasets.
[2] Yolov8 ultralytics
[3] RBE 595 Lectures.
[4] Opencv calibtateCamera()

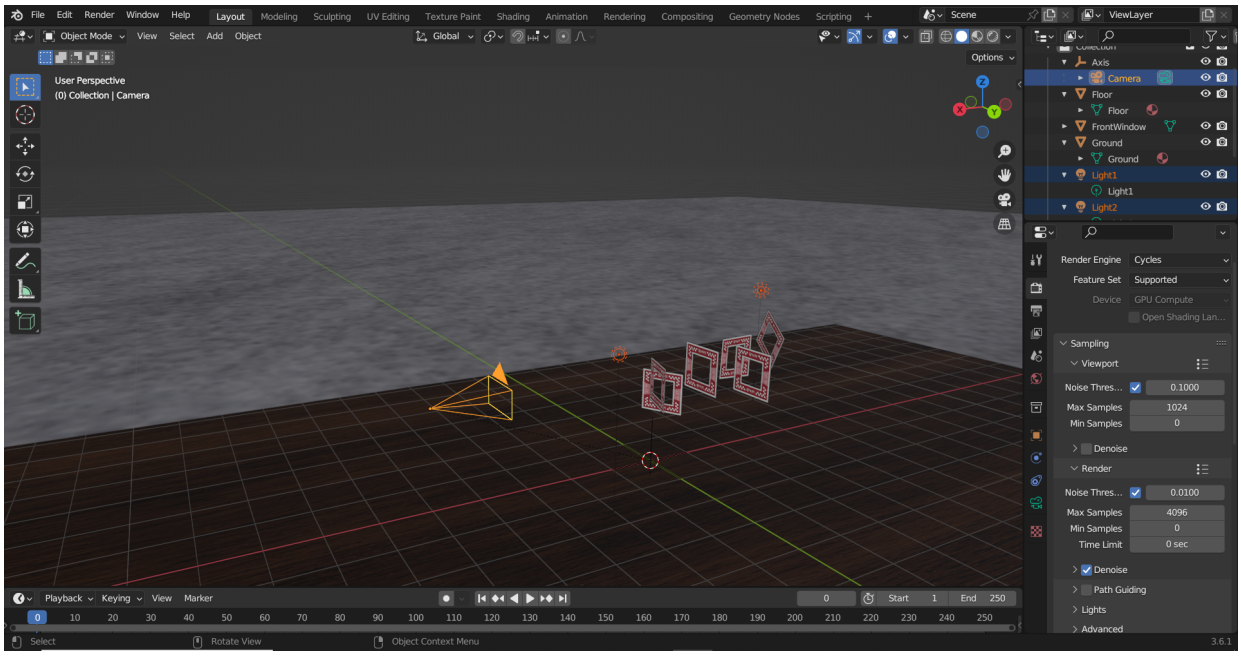

Fig. 7. Blender environment setup

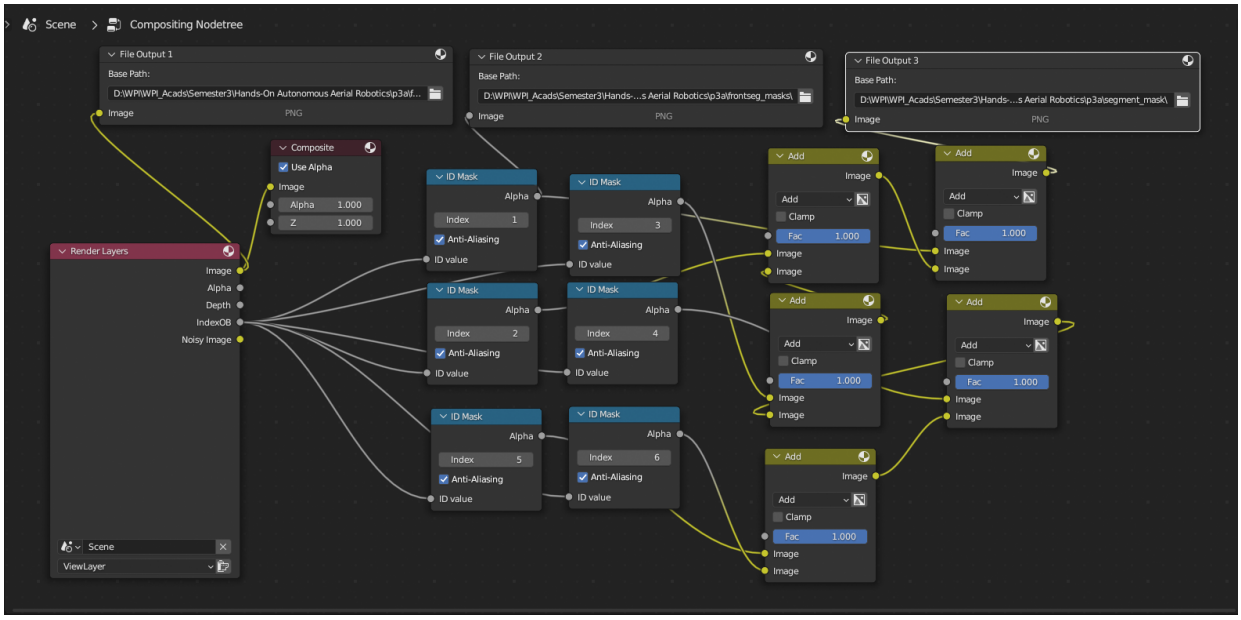

Fig. 8. Compositing Setup

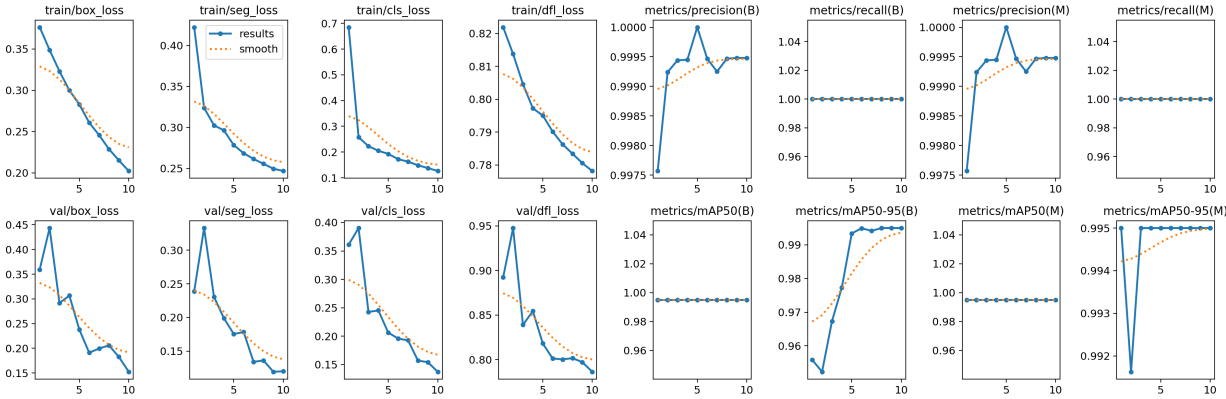

Fig. 9. training and validation results of the network

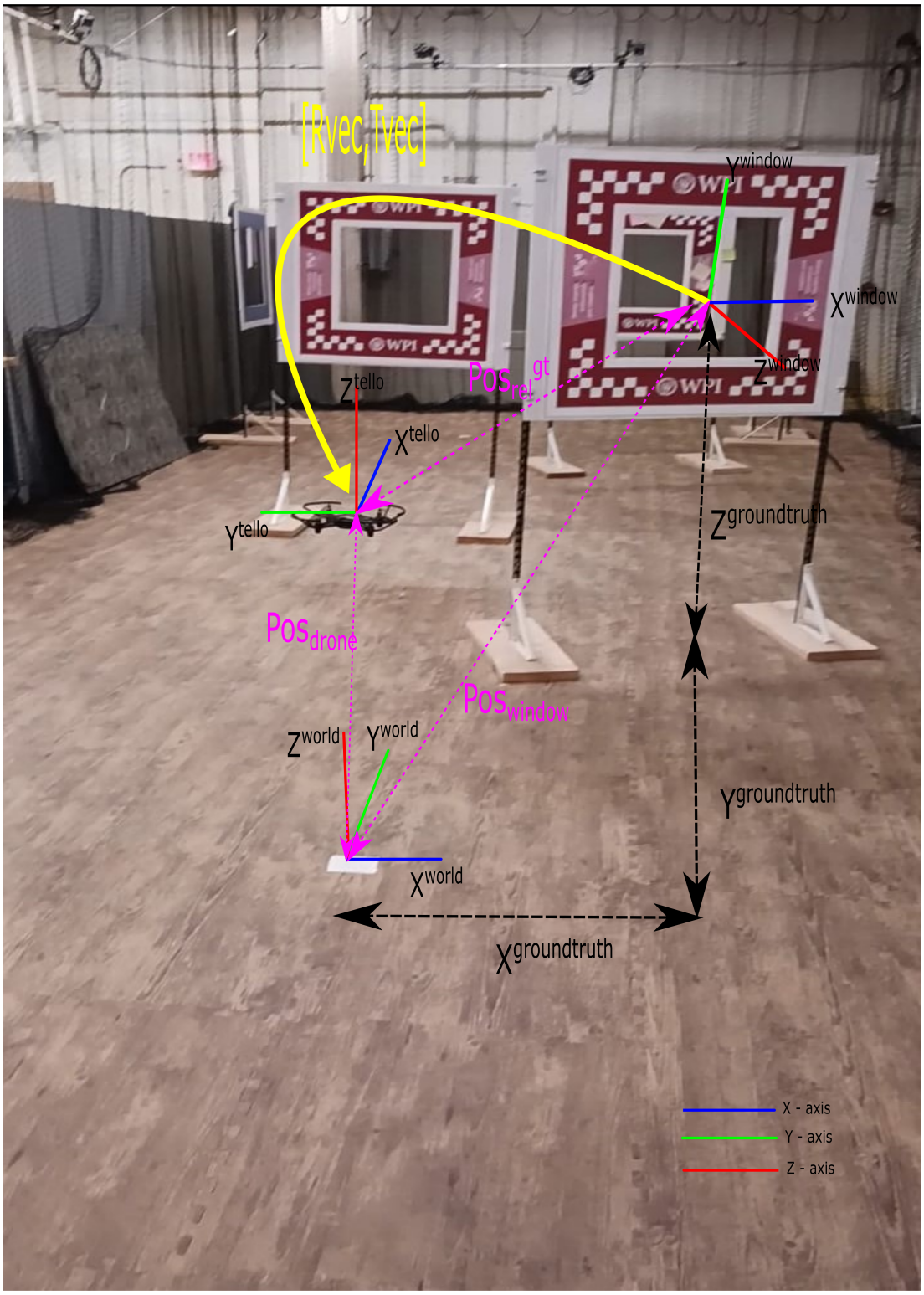

Fig. 10. Navigating through the window center