P3: Mini Drone Race!

Shakthibala Sivagami Balamurugan
Robotics Engineering
Worcester Polytechnic Institute
Email: sbalamurugan@wpi.edu

Aditya Patwardhan
Robotics Engineering
Worcester Polytechnic Institute
Email: apatwardhan@wpi.edu

Late Days Used: 6

***Abstract*—The aim of the project is to demonstrate a vision-based approach for autonomous drone navigation through obstacles of a known shape and size. We developed a U-Net neural network to perform semantic segmentation on windows, accurately identifying the obstacle's approximate center using only RGB imagery. This center point is then fed to a visual servoing controller, which guides the drone's path. The system successfully enables the drone to traverse the obstacles without relying on depth information. To ensure model robustness, the U-Net was trained exclusively on a synthetic dataset generated in Blender, which featured varied backgrounds and data augmentations.**

I. Introduction

This project entails building an autonomy stack to navigate through multiple windows inspired by the AlphaPilot competition. The project has two parts: the first part being the perception stack and the second being the planning, control and integration stack, all deployed on the Vizflyt simulator. To detect windows in the environment requires advanced segmentation capabilities for which we leveraged the power of neural nets and Sim2real. The perception stack is built using a Unet with attention modules which segments the windows detected. The dataset was synthetically generated in Blender. This helped build real-time detection and segmentation, which is critical for the autonomous navigation task. The data is split into training, validation, and test sets, and performance is evaluated using the Dice score metric. We build on this by processing the detected window mask to detect the center of the windows and use Visual servoing to control and guide the drone through the windows. We discuss our methodology over the next sections. Our project's success showcases the potential of sim2real methodologies in refining neural networks, which significantly enhance the autonomous navigation systems of quadcopters.

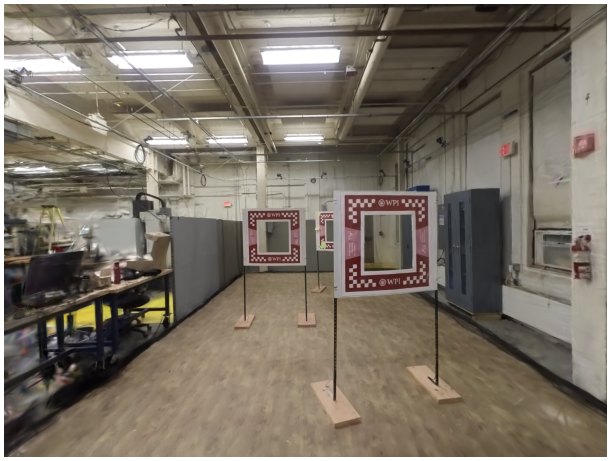

Fig. 1. FPV view of drone in vizflyt

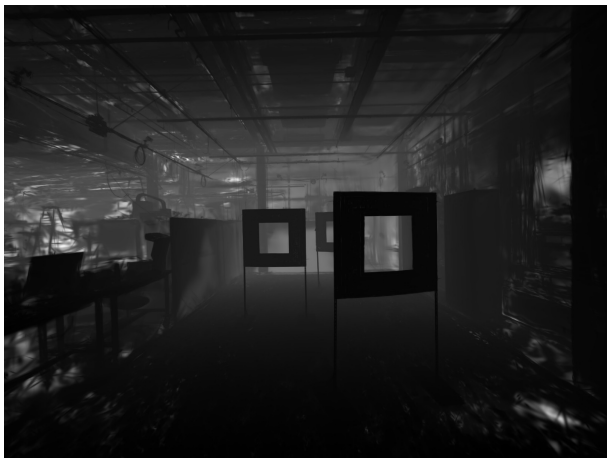

Fig. 2. Depth from Vizflyt

II. Data generation

.The data generation section contains two section, One is Dataset generation using blender and the other is Neural Network using Unet

A. Dataset Generation using Blender

The Window template is given from which the dataset generation has to begin. To train our neural net model for window segmentation we had to build our dataset containing images and labels. The images used in the dataset is trained on 3D blender environment. A multitude of scenarios were considered with varying camera angles, lighting conditions, occlusions and number of windows. Then the dataset created was augmented with varying camera placements and lighting conditions. Pictures of Laboratory kind of setup were chosen to make the model robust to handle any kind of scenarios for detection.

Initially, We did on random backgrounds which affected our model accuracy, The windows shown acted as out-of-distribution dataset from the drone fpv view, Hence to address

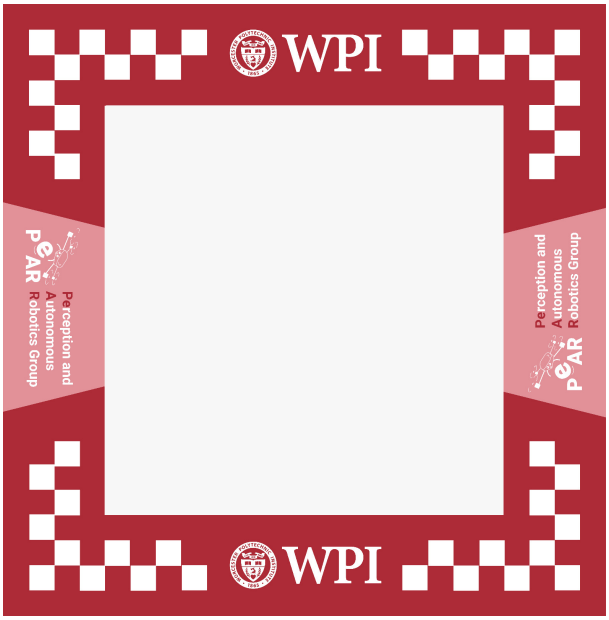

Fig. 3. Window Template

Fig. 4. Domain Randomization with the template

this problem, We chose laboratory with netted setup which was similar background as in vizflyt, With this small change and increase the dataset size, Our model could perform way better than previously it was.

B. Dataset Characteristics

The final dataset used for training had the following properties:

- **Total images:** 12000
- **Image resolution:** 640×640 pixels
- **Format:** Paired RGB images and binary segmentation masks
- **Theme:** Laboratory-style backgrounds for domain alignment

C. Data Augmentation Pipeline

To ensure the model's robustness and its ability to generalize to diverse, real-world conditions, we implemented an extensive augmentation pipeline using the Albumentations library.

These augmentations were applied only during the training phase. The validation and test sets used only resizing and normalization to accurately evaluate the model's true performance.

D. Augmentation Techniques

The pipeline included a combination of spatial, color, and noise-based augmentations:

- **Spatial Augmentations:**
  - Horizontal flip ($p = 0.5$)
  - Vertical flip ($p = 0.5$)
  - Random rotation ($\pm 45°$, $p = 0.5$)
- **Color Augmentations:**
  - Grayscale conversion ($p = 0.1$)
  - Random brightness and contrast adjustment ($p = 0.3$)
- **Noise and Blur Augmentations:**
  - Gaussian noise ($p = 0.4$)
  - Gaussian blur with kernel size 3–7 ($p = 0.4$)
  - Motion blur with kernel size 7 ($p = 0.3$)
  - Sharpening with $\alpha \in [0.2, 0.5]$ ($p = 0.4$)
- **Dropout Augmentation:**
  - Coarse dropout with 3–8 holes ($p = 0.4$)
  - Hole dimensions: 5–20 pixels
- **Normalization:**
  - Normalization using ImageNet statistics:
    - mean=$[0.485, 0.456, 0.406]$
    - std=$[0.229, 0.224, 0.225]$

E. Neural Network using Unet

For the segmentation component, we adopted the U-Net architecture, a fully convolutional encoder–decoder network designed for dense prediction tasks. The encoder path captures progressively abstract feature representations through convolution and pooling, while the decoder reconstructs the spatial resolution using transposed convolutions. Skip connections between mirrored levels of the encoder and decoder facilitate the fusion of contextual and spatial information, enabling precise delineation of structure boundaries. Given the task-specific nature of our dataset, the model was implemented in PyTorch and trained from scratch using custom dataloaders.

F. Initial Challenges and Post-processing

Initial experiments with the standard U-Net architecture yielded high Dice scores but failed to properly segment window corners—a critical requirement for accurate pose estimation. The network struggled particularly with low-contrast backgrounds where window edges were subtle.

Additionally, the model produced multiple detections on the same frame. This required a post-processing step to filter these detections and obtain a single, robust mask for the window.

G. Boundary Loss Function

To improve corner detection, we introduced a boundary-aware loss function that heavily penalizes prediction errors at window edges. This formulation ensures the network learns sharp, well-defined boundaries while maintaining global shape accuracy.

The boundary loss computes edge maps using Sobel operators and applies an increased penalty ($5\times$) at the boundary pixels. The process is as follows:

- Compute horizontal (G_x) and vertical (G_y) gradients using Sobel kernels.
- Calculate the edge magnitude: edges $= \sqrt{G_x^2 + G_y^2}$.
- Apply a binary threshold to the edge map (e.g., threshold = 0.5).
- Weight the Binary Cross-Entropy (BCE) loss: loss $=$ BCE(pred, target) $\times$ $(1 + 5 \times \text{edges})$

H. Attention U-Net Architecture

To further improve performance on low-contrast backgrounds, we integrated attention modules into the skip connections of the U-Net. The attention mechanism helps the network focus on relevant spatial regions during feature concatenation from the encoder to the decoder path.

Each attention block takes two inputs:

- A gating signal (g) from the decoder path.
- Skip connection features (x) from the encoder path.

This allows the model to learn to suppress irrelevant background regions while amplifying features relevant to the window segmentation task.

We trained the model on dataset of 12k images, for 10 epochs with a batch size of 4 . We got Test Dice Score: 0.94 and Test IoU Score: 0.89

III. Post Processing Technique

This post-processing pipeline refines the raw output of the neural network to accurately identify and isolate window regions in each image.

After the network predicts a probability mask, it is first thresholded and resized back to the original image resolution. Because multiple windows may overlap or touch, the code then applies morphological operations and the watershed algorithm to separate them into individual, non-overlapping masks. If watershed fails, a fallback erosion–dilation method is used to break connected regions apart. Each separated mask is analyzed to find the largest contour, assuming that the biggest window corresponds to the one closest to the drone. From this contour, the script fits a rectangle to the contour and computes its centroid, area, and corner points using OpenCV's contour and geometry tools. The centroid of the rectangle is calculated using cv2 moments

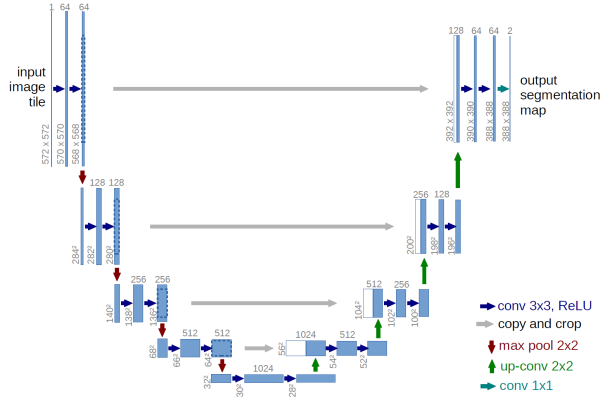

Fig. 5. Unet architecture

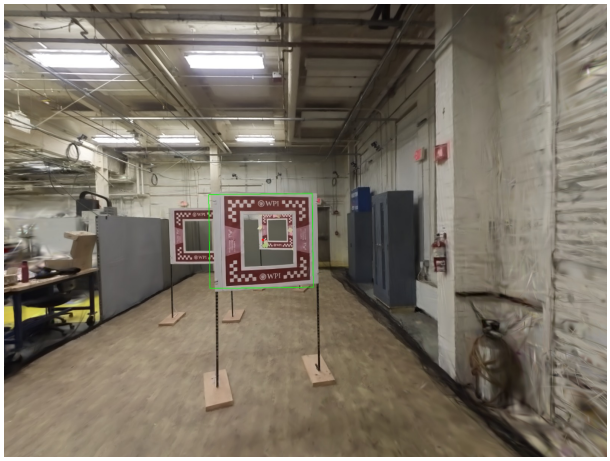

Fig. 6. Detected Window by Unet

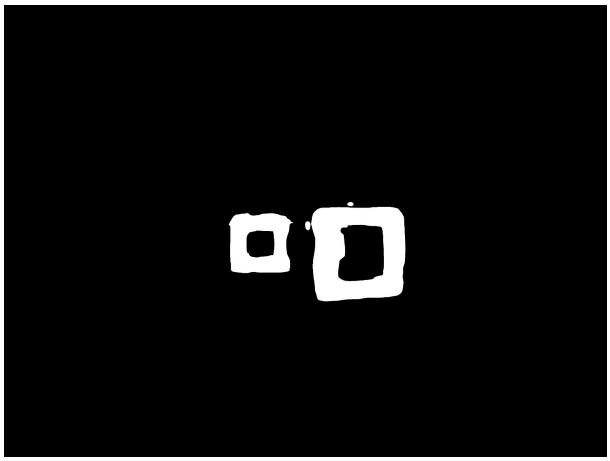

Fig. 7. Model's Output mask

$$c_x = \frac{M_{10}}{M_{00}}$$

$$c_y = \frac{M_{01}}{M_{00}}$$

Finally, a new clean mask is drawn showing the detected window outline and center. This post-processing cleans up the neural network's mask, splits overlapping windows, and extracts precise geometric information (position, shape, and size) in the image frame for visual servoing and alignment.

IV. Algorithm for Waypoint Navigation using Monocular camera

Depth is not used

- The first step is to run the model at 5Hz with CPU and obtain the post processed mask at each iteration.
- Once post processed mask is obtained after every 0.2 seconds, We then calculate the error from the centre and in x, y direction on the pixel coordinate frame and align the drone, This method is called Visual Servoing.
- After multiple trials, we observed that six iterations are sufficient for the drone to align with the center. We avoid defining a fixed error range because the parameters vary—at times the deviation exceeds the range, while at other times it falls short.
- Once the drone aligns, We then move forward in x direction, The logic here is to calculate area of the detected window and once the drone starts to move forward, The area increases in pixel coordinate frame, hence we make the drone to move for 3 seconds.
- With this logic the window 1 and window 2 can be passed, But for window 3 the drone strugged because the window 3 was not in the field of view.
- We made the drone yaw in y direction if a window is not detected.
- With this algorithm all three windows were passed autonomously by the drone.

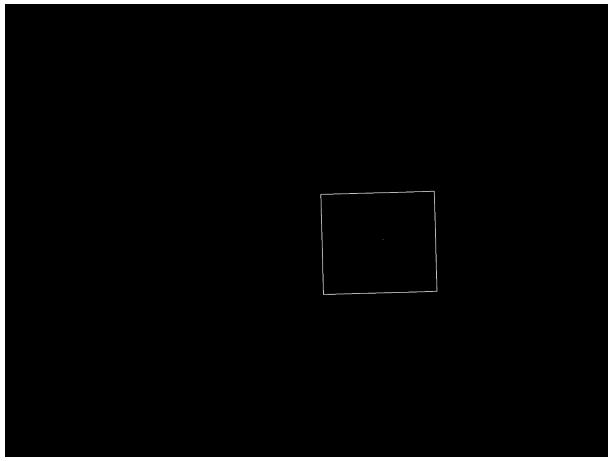

Fig. 8. Post processed mask

V. Assumptions

- It is assumed that Quadrotor is a point object.
- We have decoupled the camera orientation from the drone's body so that the Splat view always stays upright by sending only the yaw (heading) angle to the renderer while keeping roll and pitch fixed at zero. This prevents the video from flipping when the drone starts inverted, without affecting its dynamics or control behavior.

VI. Results and Analysis

Two videos were shown, One with just drone maneuvering and the other with the detection and post processed mask shown, With the Unet model, The drone was able to navigate and pass through all three windows autonomously. The model's accuracy is x% and drone's tolerance was +/-0.005m in a 2m width map. The drone's velocity was 0.2 m/scale in vizflyt simulator.

VII. Conclusion

With the algorithm mentioned in section 3, We were able to pass through all the 3 windows with the drone and the perception stack for drone with monocular camera is written successfully.

VIII. Supplementary Material

The videos of the Assignment 3 is given along with both detections overlay and without overlay: https://bit.ly/4oVMH2s

References

.

[1] https://github.com/milesial/Pytorch-UNet
[2] Ramana's Blender Tutorial