Team Apache Stealth: Mini Drone Race - Perception Saga!

Ankit Mittal
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: amittal@wpi.edu

Rutwik Kulkarni
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: rkulkarni1@wpi.edu

***Abstract*—This project targets the creation of a deep learning-based perception stack for a quadrotor to navigate through artificially designed windows, a challenge inspired by Lockheed Martin's AlphaPilot competition. The perception stack uses a neural network for real-time detection and segmentation of windows at various angles and light conditions. The network is trained on a dataset created in Blender, which ensures the model accurately represents the windows' visual features. The stack includes an algorithm to select the nearest window and a 3D pose estimation process utilizing Perspective-n-Point methods, improved by detailed camera calibration. When implemented on a DJI Tello drone, the system effectively estimates the window pose in real-time, which is essential for autonomous navigation. This project contributes to the field of autonomous drone racing, showcasing the application of machine learning and computer vision in dynamic and challenging environments.**

I. Introduction

This project aims to develop a perception stack for a quadrotor drone to autonomously navigate through windows, a task inspired by the conditions of the AlphaPilot competition. The initial step involves detailing the hardware specifications, focusing on the type of image data retrieved from the drone's camera and the methods used to interface with the drone via the DJITellopy API.

Within the Perception Stack, the project features a Window Detector that uses a U-net neural network architecture trained on a dataset created in Blender, followed by a corner detection and filtering process to identify the most accessible window for navigation. The network's training process is comprehensive, with a division of data into training, validation, and test sets and the use of a Dice score metric for performance evaluation.

The final steps include camera calibration to address lens distortion and the use of the Perspective-n-Point method in the OpenCV library for pose estimation. These components are tested to verify the drone's ability to accurately estimate window poses in real-time, with results demonstrating the practical application of the perception stack for drone navigation. The introduction provides a summary of each section, highlighting the use of machine learning and computer vision to enable autonomous drone flight through a controlled environment.

Link To Videos: **Click Here**

II. Test Window Setup

The testing framework comprises three distinct window configurations to assess the drone's navigation system:

1) **Test Case 1:** A window angled relative to the drone's approach path, introducing a more complex test case for the drone's three-dimensional pose estimation capabilities.
2) **Test Case 2:** A window that is partially obscured from the drone's view, challenging the detection and segmentation algorithm to maintain performance despite incomplete visual information.

Fig. 1: Test Case 1- Tilted Window

Fig. 2: Test Case 2- Occluded Window

III. Window Detection using Deep Neural Network

A. *Data Generation (sim2real)*

For the training of deep learning models, a substantial and diverse dataset is essential. Generating such datasets from real-world scenarios is typically resource-intensive. To mitigate this, we utilize Blender for synthetic dataset creation, offering a cost-effective and scalable solution.

The dataset consists of 6000 images, with each image dimension being (480x360). Data augmentation techniques were applied to ensure the dataset captures a wide range of environmental conditions. Mask images corresponding to each window scene were also produced in Blender. These masks are critical for the segmentation algorithm's training, providing a benchmark for accuracy. This approach of generating window images alongside their masks equips the perception system to generalize effectively in various settings.

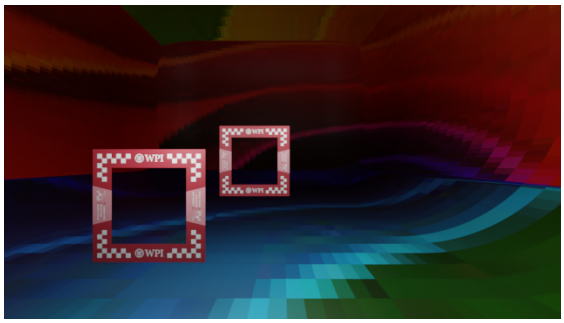

Fig. 3: Simulated Environment in Blender

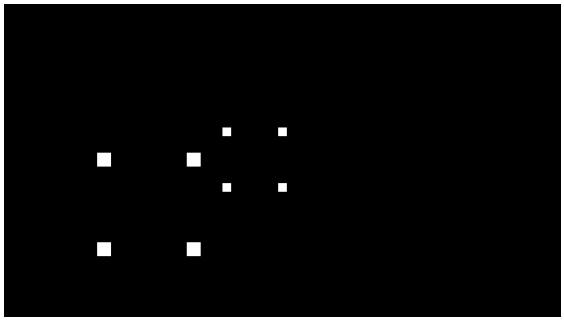

Fig. 4: Corresponding Label Image of the of the Simulated Environment

B. *DNN for Segmentation : U-NET*

The U-Net architecture is structured as an encoder-decoder network with a characteristic "U" shape, which is where it gets its name. It consists of a contracting path (encoder) to capture context and a symmetric expanding path (decoder) that enables precise localization.

*1) **Model Architecture**:* The U-Net architecture processes images in the format $[batch\_size, channels, height, width]$. The architecture encodes an input image ([2, 3,

360, 480]) through four successive down-convolution blocks, reducing spatial dimensions while increasing feature channels from 64 to 512. Max pooling is applied between encoder blocks to downsample the image. The encoder's last block outputs a feature map of [2, 512, 23, 30], which is then passed through a latent layer ([2, 1024, 23, 30]) that serves as a bottleneck capturing the image's most abstract representation.

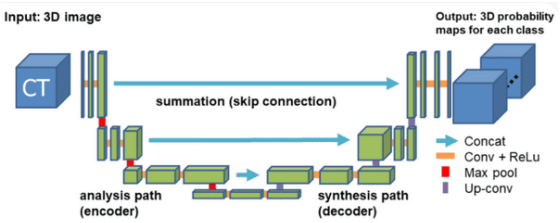

Fig. 5: U-NET Architecture

The decoder reverses the process with up-convolutions, halving the channels and doubling the dimensions at each stage, utilizing skip connections from the corresponding encoder outputs to preserve detail. The final decoder output is [2, 64, 368, 480], which is slightly larger than the original due to (0,0,4,4) padding applied to match the input's spatial dimensions post-processing. The network concludes with an output layer that uses a 1x1 convolution to generate a single-channel segmentation map of the same resolution as the padded input ([2, 1, 368, 480]).

*2) **Training, Loss Function, and Optimization.**:* The U-Net model is trained using the dataset of simulation images from Blender, divided into training, validation, and test sets, with distribution being 94 percent, 3 percent, and 3 percent respectively out of a total dataset size of 6000 images. The network is implemented in PyTorch and is set up to train on a GPU if available. It employs the above-described U-net architecture. Training is driven by the BCEWithLogitsLoss function, suitable for binary classification, and uses the Adam optimizer with a learning rate of 0.0001 and weight decay regularization set at 0.00001. Model parameters are iteratively updated through backpropagation during training epochs, with performance assessed on the validation set after each epoch and final model generalization tested on the test set post-training.

C. Evaluation (Results from DNN)

In evaluating our deep neural network, the Dice score is employed as the primary metric due to its suitability for segmentation tasks, measuring the overlap between predicted segmentation and ground truth annotations. It ranges from 0, indicating no overlap, to 1, denoting perfect agreement. For our test set, we utilize genuine drone-captured images to ensure realistic assessment conditions. The network has achieved an impressive Dice score of 0.915 for a single image, indicating high accuracy in segmentation. Moreover, when considering all images in the test set, the network maintains a high level of performance, with an average Dice score of 0.8, reaffirming its reliability in processing real-world data.

Fig. 6: Image going into the Network

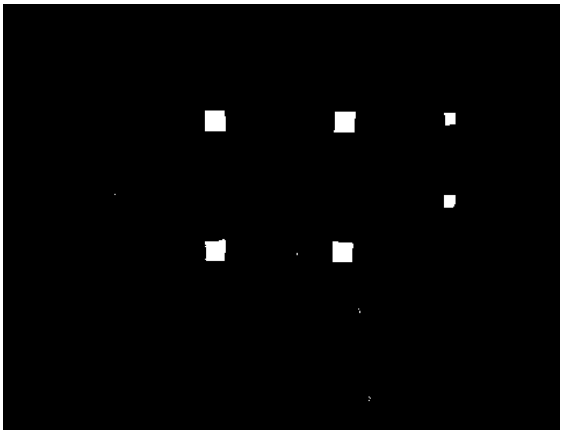

Fig. 7: Image coming out of the Network

IV. Post-processing and Pose Estimation

A. Corner Detection

This section delves into a detailed examination of an advanced image processing method developed for the precise detection of corners within digital images, with a particular focus on identifying the corners of windows. Initially, the process entails a pre-processing phase, where the image is enhanced to optimize quality, thereby facilitating more accurate analysis in subsequent steps. The core of the detection technique employs the findContours algorithm from the OpenCV library, which is adept at detecting all significant patches within the image. The algorithm proceeds to compute the centroid of each detected patch, which serves as a provisional location of the window corners.

Given the prevalence of noise in the binary mask image, the method incorporates a crucial non-maximum suppression step. This step is pivotal as it selectively filters out less prominent features, effectively discarding false positives that lack the strong intensity variations typically associated with genuine corner points. By doing so, the technique ensures that only the most salient corners—those with a pronounced intensity gradient and geometrical alignment with the window structure—are retained for further analysis.

To identify the nearest window from an image, our image processing system employs the findContours function of OpenCV on a pre-processed binary mask to detect contours, which signify potential windows. By calculating the area of these contours and identifying the largest one through the contour area function, we assume the largest contour to represent the closest window due to the perspective correlation between size and distance. This contour is then segmented, focusing our analysis on the most proximate window for detailed feature analysis or further computer vision tasks. While the initial method for identifying the nearest window relies on the contour area, a more sophisticated approach could utilize re-projection error, given that the approximate real-world coordinates of the windows are known. However, this method proved challenging due to the drone's imprecise odometry and slight physical deviations in the camera's angle, which led to inconsistencies in the results that did not align with the expected outcomes.

when only three corners are visible, we used the geometric properties of parallelograms estimating the position of the fourth corner of a window. The concept is based on the fact that in a parallelogram, the sum of the vectors of adjacent sides equals the vector of the opposite side.

B. Pose Estimation

Using the known dimensions of the window, we established world coordinates at the window's center and extracted the corresponding image points from the captured image via a neural network. These image points represent the corners of the window in the image. We then applied a perspective-n-points (PnP) algorithm to estimate the window's pose by aligning the world points with the image points, effectively determining the window's position and orientation relative to the camera. This method, while sophisticated, faced challenges due to inaccuracies in the drone's odometry and slight camera tilts, leading to re-projection errors. We refined the process by enhancing camera calibration and error correction to improve the pose estimation

V. Results

Below is the link to a live demonstration showcasing the capabilities of our perception stack in action. **LIVE DEMO!!**

A. *Test Image 1*

Fig. 8: Raw Image of the Tilted window

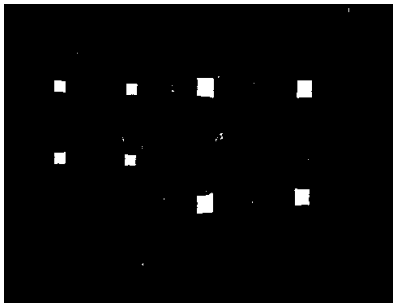

Fig. 9: Output from the DNN

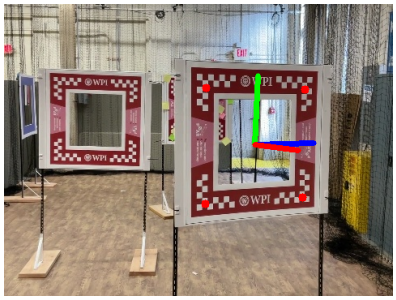

Fig. 10: Output after Pose Estimation

B. *Test Image 2*

Fig. 11: Raw Image of the Occluded Window

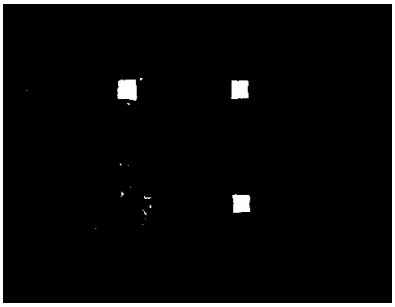

Fig. 12: Output from the DNN

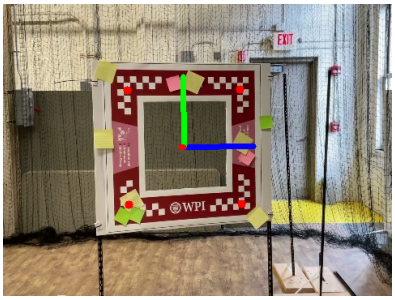

Fig. 13: Output after Pose Estimation

VI. Observations

- In the context of pose estimation using Perspective-n-Point, accuracy is contingent upon the complete visibility of the window within the field of view of the drone camera. Field tests reveal a limitation where the drone captures only the lower half of the window when positioned centrally and at a consistent distance, resulting in a tilted image.
- Furthermore, to improve pose estimation accuracy, it is suggested to incorporate reprojection error calculations from drone odometry. This step is expected to enhance the robustness of pose estimation during the drone's maneuvering phases.

VII. Conclusion

In conclusion, the project has integrated a deep learning perception system into a drone, allowing it to autonomously navigate through windows. The U-Net neural network, trained on a dataset created with Blender, was effective for real-time window detection and segmentation. The system's accuracy was confirmed through camera calibration and the use of Perspective-n-Point methods for 3D pose estimation. Tests in controlled environments showed the system to be accurate, as indicated by the Dice score metrics. These results suggest that the system could be used in real-world autonomous drone applications and provide a basis for further research in this area.

VIII. Acknowledgment

The author would like to thank Prof. Nitin Sanket and the TA of this course RBE595.

References

[1] DJITelloPy **Link**
[2] U-Net: Convolutional Networks for Biomedical Image Segmentation **Link**