Team Apache Stealth: Mini Drone Race - Perception Saga!

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

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

***Abstract*—This project aims to develop a planning and control stack for a quadrotor, leveraging a deep learning-based perception system initially created for navigating through artificially designed windows. This challenge draws inspiration from Lockheed Martin's AlphaPilot competition. The perception stack, which includes a neural network, is adept at real-time detection and segmentation of windows under varying angles and lighting conditions and has been trained on a dataset generated in Blender to ensure an accurate representation of the windows' visual features. Building upon this, the project introduces a layer for path planning and control that utilizes the DJITelloPy package. This allows the drone to not only select the nearest window but also to compute and execute a flight path through it, considering real-time pose estimation facilitated by Perspective-n-Point methods and refined through precise camera calibration. Implemented on a DJI Tello drone, the integrated system is capable of autonomous navigation, adjusting its trajectory to successfully navigate through the windows.**

I. Introduction

This project focuses on developing an integrated perception, planning, and control stack for a quadrotor drone to autonomously navigate through windows. Inspired by the AlphaPilot[1] competition by Lockheed Martin, the work involves a detailed understanding of the drone's hardware capabilities, specifically the image data from the drone's camera and interfacing with the drone using the DJITellopy[2] API.

The perception stack features a Window Detector using a U-net neural network architecture trained on a synthetic dataset from Blender[3]. This facilitates real-time detection and segmentation, which is critical for the autonomous navigation task. The network undergoes rigorous training, with the data split into training, validation, and test sets, and performance is evaluated using the Dice score metric.

Building on this foundation, the project introduces a planning and control component. Utilizing a map with rough 3D coordinates of the windows, the system computes a preliminary trajectory for the drone, taking into account the odometry to reach a vantage point for clear window visibility. Once in position, the perception stack's real-time pose estimation, fine-tuned through camera calibration and Perspective-n-Point methods from OpenCV[4], allows the drone to precisely navigate through the windows.

The integrated system's ability to estimate window poses accurately and navigate accordingly is confirmed through extensive testing. This outlines the project's trajectory from machine learning model training for window detection to the seamless integration of planning and control algorithms for autonomous flight in a structured environment.

Link To Videos: **Click Here**

II. Map Reader and Test Environment Setup

To evaluate the drone's autonomous navigation capabilities, the system is tested against three specific window configurations, each designed to challenge different aspects of the drone's perception and control stack:

The Map Environment file provided for testing follows a specific format, as shown in the example

Fig. 1: Test Environment Image

below. However, for navigation, the system primarily utilizes the approximate x, y and z coordinates of the windows. These coordinates are instrumental in bringing the window into the drone's field of view (FOV), which does not require precise positioning due to the drone's perception stack capable of refining the pose estimation.

Link To Test Environment Video: **Click Here**

III. Hardware Specifications

The DJI Tello Edu drone[2], equipped with a 5-megapixel camera, captures photos at a resolution of 2592x1936 pixels and streams video at 960x720 pixels. It has an 82.6-gram takeoff weight and offers a flight duration of approximately 13 minutes, powered by a 1.1 Ah/3.8 V battery. With a flight range of up to 100 meters and a maximum altitude of 10 meters, the Tello Edu is operated using the DJITelloPy Edu library in Python, enabling a broad array of programmable controls and interactions for educational and development purposes.

IV. 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[3] 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[3]. 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.

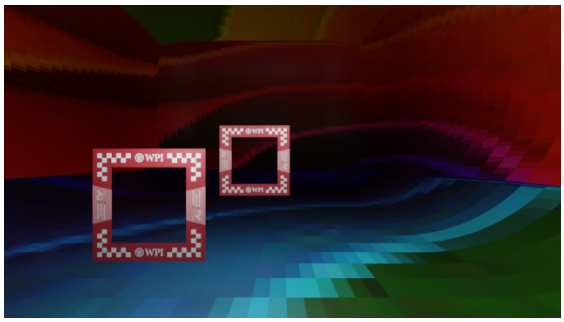

Fig. 2: Simulated Environment in Blender

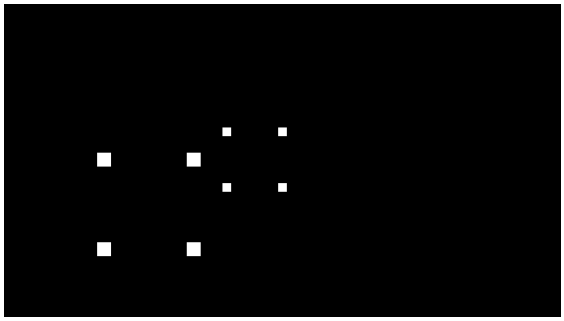

Fig. 3: Corresponding Label Image of the of the Simulated Environment

B. *DNN for Segmentation : U-NET*

The U-Net[5] 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[5] 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.

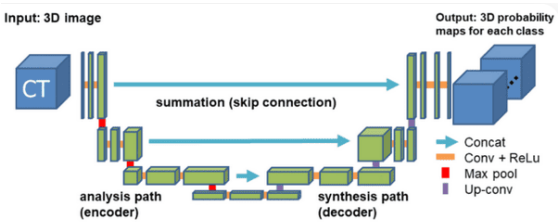

Fig. 4: U-NET Architecture

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[5] 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. 5: Image going into the Network

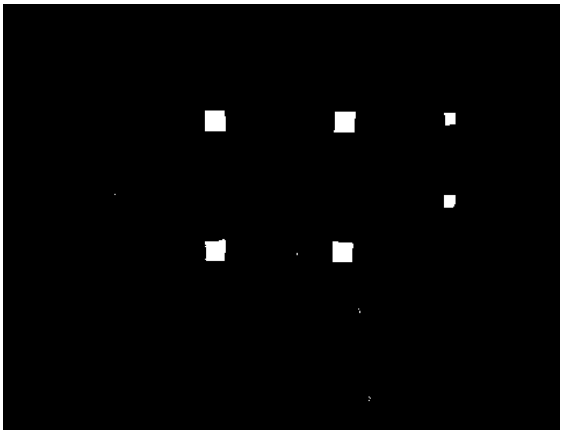

Fig. 6: Image coming out of the Network

V. 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[4] 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[4] 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 to estimate 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. *Camera Calibration*

Camera calibration is a critical step in ensuring that the pose estimation is based on accurate information about the camera's geometric and optical characteristics. To achieve this, we have employed the camera calibration functions available in OpenCV[4]. By taking multiple images of a known calibration pattern (such as a chessboard or grid) from different angles, we can compute the camera matrix and distortion coefficients which are essential for correcting lens distortion and obtaining reliable measurements from the image data.

The calibration process yields the camera matrix K, which includes the focal lengths f_x and f_y, and the optical centers c_x and c_y. An example of a camera matrix might look like the following:

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

To determine the camera matrix for a drone, we first resized the camera image to half its original dimensions, resulting in a new resolution of 480 x 360 pixels. We ensured that the aspect ratio of the image remained unchanged during this resizing process. As a result, the following values were obtained:

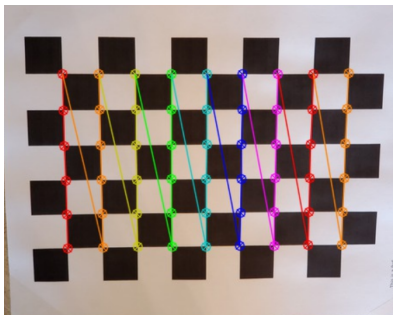

Fig. 7: Camera Calibration

$$K = \begin{bmatrix} 453.756 & 0 & 236.286 \\ 0 & 454.004 & 181.620 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion coefficients account for radial and tangential lens distortions. These are denoted as k_1, k_2, p_1, p_2, and sometimes k_3, in the case of severe radial distortion. Sample values for these coefficients might be given as:

Distortion Coefficients =

$$= [k_1, k_2, p_1, p_2, k_3]$$

$$= [-0.0272, -0.2419, 0.0022, 0.0006, 0.5753]$$

With the camera properly calibrated, we can remove distortion from the images, which allows for more accurate feature extraction and pose estimation. The corrected image points can then be used with the Perspective-n-Point algorithm to compute the pose of the drone in relation to the windows with higher precision.

C. *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

VI. Path Planning and Control

For the autonomous navigation of the quadrotor through Windows, we developed a path planning and control strategy that leverages the capabilities of our perception stack. Our approach focuses on utilizing the x, y, z coordinates extracted from the window's pose to position the drone optimally, ensuring the target window is within the Field of View (FOV) of the drone. This is crucial for the perception system to detect and segment the window accurately, even when multiple windows are present in the frame.

A. *Coordinate Extraction and Initial Positioning*

Our map reader module parses the environment file to extract the approximate x, y, z coordinates of the windows. With these coordinates, we calculate an initial waypoint positioned at a suitable offset—typically 1.5 meters from the window. This offset is crucial for ensuring that the window is detectable within the drone's FOV, considering potential inaccuracies in the map data and the drone's current position.

B. *Path Execution and Real-time Adjustment*

Once the drone is optimally positioned at the calculated offset, the perception stack evaluates the window's pose. If the window is determined to be in a navigable position, we utilize the DJITelloPy[2] Library to issue position commands to the drone. These commands are derived from the obtained waypoints, which are based on the pose estimation

matrix. With these instructions, the drone maneuvers through the window, executing a seamless transition from approach to passage.

C. Sequential Navigation

After successfully passing through one window, the drone's system immediately retrieves the position data for the next window from the map reader file. The process is then repeated: the drone maneuvers to the new offset position, the perception stack updates the pose estimation, and the drone navigates through the next window. This sequential approach is maintained until all windows have been traversed.

D. Challenges and Corrections

The main challenges we faced included ensuring that the drone's camera was correctly aligned to detect the window despite any camera tilt and managing the drone's odometry to maintain accurate tracking. These were addressed by adjusting the drone's altitude as needed and by real-time odometry tracking to correct any deviations from the planned trajectory.

In summary, the path planning and control system is intricately linked with the perception stack, allowing the drone to autonomously navigate through a series of windows. By continuously updating the drone's position in relation to the windows' locations and adjusting its path accordingly, we achieved a seamless and efficient navigation process.

VII. Results

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

A. *Test Image 1*

Fig. 8: Raw Image of the Tilted window

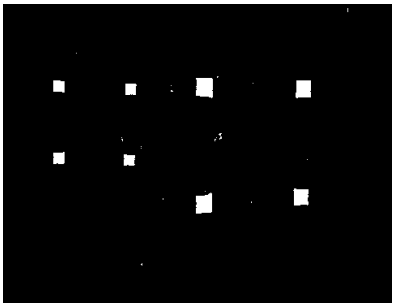

Fig. 9: Output from the DNN

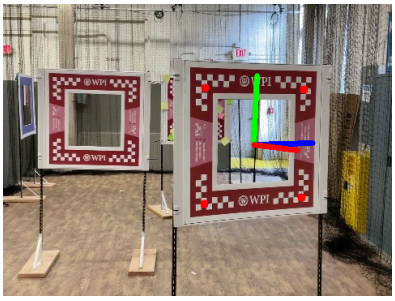

Fig. 10: Output after Pose Estimation

B. *Test Image 2*

Fig. 11: Raw Image of the Occluded Window

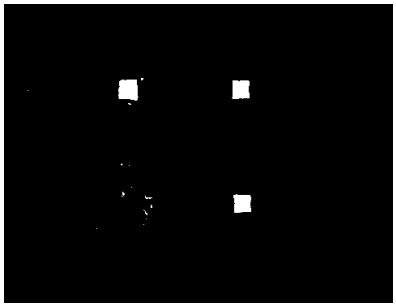

Fig. 12: Output from the DNN

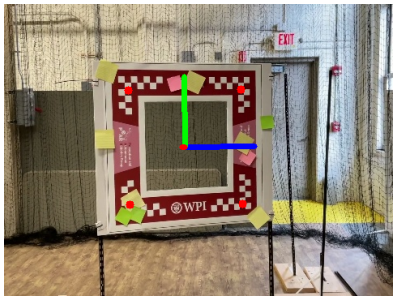

Fig. 13: Output after Pose Estimation

VIII. 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.

IX. Conclusion

X. Acknowledgment

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

References

[1] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," *CoRR*, vol. abs/2005.12813, 2020. [Online]. Available: https://arxiv.org/abs/2005.12813

[2] W. Giernacki, J. Rao, S. Sladic, A. Bondyra, M. Retinger, and T. Espinoza-Fraire, "Dji tello quadrotor as a platform for research and education in mobile robotics and control engineering," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 735–744.

[3] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org

[4] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[5] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597