P5: The Final Race!
Team Nimbus Navigators

Chaitanya Sriram Gaddipati[1]
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: cgaddipati@wpi.edu

Ankit Talele[1]
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: amtalele@wpi.edu

Shiva Surya Lolla[1]
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: slolla@wpi.edu

***Abstract*—This project focuses on designing a perception and navigation stack for the DJI Tello EDU quadcopter to autonomously navigate through three stages of an obstacle course in a drone race. For more details of the race track and the phases refer here (link). The first stage comprises two square-shaped windows with unique textures we passed through in project 3 (link). The second phase is comprised of an uneven shaped gap which we passed through in project 4 (link). The final stage is unique to this project and is comprised of a dynamic window with a rotating hand (Like a clock). We tackle this problem using color thresholding (due to unique colors) and passed through the window using angle estimates of the hand. Using this methodology, we pass through all three stages to finish the drone race successfully.**

I. Phase 1 - Race Window Detection

Refer to fig 3 to see the corners and 3D center axes detected on the race windows 1 and 2 on the race track.

A. Design Decisions

We wanted to use a neural network to get the mask of the front most window and then infer the corners of it and perform PnP to get the 3D pose of the window center. We selected YOLOv8 over other models for its superior real-time inference capabilities, essential for smooth operation on the NVIDIA Jetson Orin Nano.

B. Training and Augmentation

We used a sim2real transfer approach to train the yolov8 network to identify the front most race window on an image. For this we created a synthetic dataset from blender environment by arranging the windows randomly and get the image and corresponding label mask. For more details refer to project 3 report (link). Once we got some synthetic data we performed domain randomization by warping and changing the background for diverse data, better generalization and a robust detection network. In total we had around 15000 synthetic images and labels in the dataset.

C. Corner Inference

For window detection, we refined the network's masks using erosion and dilation, followed by contour detection and polygon approximation for accurate corner identification. This method was essential for overcoming challenges with the initial masks.

D. Calibration

Calibration involved using checkerboard images to determine the drone camera's intrinsic matrix and distortion coefficients, critical for accurate pose estimation.

E. Pose Estimation using PnP

We used solvePnP in OpenCV for pose estimation, requiring 3D object points, 2D image projections, and camera calibration data. The window's center was strategically positioned as the coordinate system's origin for effective pose extraction and we have the window length and breadth to get the known four 3D points.

F. Planning and Control

Our approach involved navigating through multiple windows in a mapped environment. We used a direct waypoint approach by going to a initial waypoint to see the window properly and get the center pose from here. This 3D pose is given to pass through the window.
The fig 1 shows a sample result of running the network to get masks and corners of the front most window.

Fig. 1. A sample of the masks detected by the network and corners inferred is shown here.

II. Phase 2 - Unknown Shape Gap

We use optical flows for detecting the gap and post process the flows to identify gap contour and center for the drone to pass through. Refer to fig 4.

A. Optical Flow Estimation using RAFT

The fundamental assumption of optical flow is that the brightness of any object point in an image remains constant over time. Mathematically, this is represented by the brightness constancy constraint equation:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{1}$$

where $I(x, y, t)$ is the intensity of the pixel at position (x, y) at time t, and dx, dy are the displacements of the pixel in the x and y directions between times t and $t + dt$.
In the context of drone navigation, optical flow aids in understanding the movement of the drone relative to its surroundings. By analyzing the optical flow of frames captured during the drone's flight, it is possible to infer motion patterns and make navigational decisions. The procedure involves:

1) Capturing consecutive frames during the drone's movement.
2) Applying computer vision techniques to these frames to calculate optical flow, which represents the motion between them.
3) Post-processing the optical flow data to extract meaningful information like the movement direction and magnitude.

In this specific application, the drone captures two frames while performing a vertical movement (up and down). The relative motion induced by this movement enhances the quality of the optical flow output. This output is then processed to generate contours, which help in identifying and locating gaps or openings (like holes) in the environment when fitted with convex hulls. The center of these gaps is calculated from the optical flow data and contours. The drone then aligns its image center with the gap center within a tolerance using a conversion factor we got for 2D pixel and 3D point correspondences. Once the centers are aligned the drone goes through the gap.

B. Use of RAFT Pre-trained Optical Flow Network

The RAFT (Recurrent All-Pairs Field Transforms) pre-trained network, known for its precision and efficiency in optical flow computation, was chosen for its compatibility with CUDA processing, enabling sub-second optical flow generation. This rapid capability is vital for real-time applications like drone navigation, allowing drones to process visual data and make immediate navigational decisions without delay. RAFT's effectiveness was confirmed through tests using various gap shapes and textures in Blender, and even in challenging scenarios with identical foreground and background textures. Although the Intersection over Union (IoU) was slightly lower in these scenarios, the results were still satisfactory. The network, along with post-processing steps, was successfully implemented on the drone in real-time.

C. Post-processing

In our post-processing, adaptive thresholding was applied to the optical flow image to accentuate key features under varying lighting conditions, outperforming global methods like Otsu's. This highlighted critical elements, especially gap edges. We then used the Canny algorithm for edge detection, known for its precision in identifying strong edges. The detected edges underwent dilation and a morphological 'closing' operation to create a continuous, clear representation of potential gaps. Contour detection followed, identifying and prioritizing the largest contours, likely indicating the biggest gaps. The centroid of the largest contour was calculated using image moments, crucial for guiding the quadcopters positioning to navigate through these gaps.

D. Visual Servoing

Our visual servoing process for drone navigation involves:

1) **Calculate Distance Between Centers:**
   - Identify the center of the gap in the image (i.e. in pixels).
   - Compute the Euclidean distance to the image center.
   - This indicates the drone's alignment with the target on a 2D plane.
2) **Check Distance Against Tolerance:**
   - If within a predefined tolerance, the drone proceeds.
   - Otherwise, it aligns with the gap's center.
3) **Horizontal and Vertical Adjustments:**
   - Calculate horizontal and vertical differences to the gap center.
   - Convert these into movement commands for alignment.
4) **Movement Execution:**
   - Execute adjustments to navigate through the gap.

Observations from experiments:

- Vertical adjustments are unnecessary if the drone's height matches the gap.
- The front camera's downward tilt allows more lenient tolerance.

III. Phase 3 - Dynamic window

In this stage, we navigate through a "cyan" colored square window with a pink clock-like hand rotating in the middle as shown in fig 2. The idea here is to pass through the window such that the drone does not hit the window or the rotating hand.

A. Strategy

As in the case of the third project (P3), we cannot pass through the center of the window because then the quadcopter would collide with the window. Therefore, our strategy was to pass through the bottom half of the window once we were

Fig. 2. Dynamic window

certain enough that the hand would not hit the quadcopter.

One decision we had to make was whether to use a deep learning model or a classical approach to achieve our goal. We could use our earlier trained Yolov8 network (from project 3) to solve this problem by getting the center position relative to the drone and using that position to get the coordinates of the center of the lower half of the window using the window dimensions to pass through the same. However, when we looked at the window colors we thought we could get away with simple HSV color thresholding due to the distinct colors of the window's hand and the window. This method is beneficial due to the limited computational overhead of HSV thresholding compared to neural network inference.

B. Algorithm

Refer to fig 5 for the masks and axes obtained using the below described steps.

1) HSV Thresholding and Contour Detection: The algorithm commences with the conversion of the input image from BGR (Blue, Green, Red) to the HSV color space. This transformation facilitates the identification of specific color ranges corresponding to the window and the hand. We defined specific HSV ranges for cyan (the window color) and pink (the hand color).

Following the color space conversion, the image undergoes thresholding to isolate the desired colors. This process results in binary images where the pixels of interest are white against a black background. To refine these binary images, morphological operations such as dilation and erosion are applied.

2) Contour Analysis: Post-thresholding, the algorithm proceeds to contour detection. Contours are identified in the thresholded images, and the largest contour, presumed to be the rotating hand, is selected for further analysis. We calculate the centroid of the square window and the extreme points of the rotating hand.

The primary challenge lies in determining the optimal time for the drone to pass through the lower half of the window. To address this, we assess the orientation of the hand relative to a fixed reference line, typically aligned with the noon position. By computing the angle between the hand and this reference line, we can predict the hand's movement and time of the drone's passage to avoid collision.

In our case, the drone passes through the window if it detects that the angle of the hand is greater than 170 degrees. To send the drone through the bottom half we use PnP algorithm to get the 3D coordinates of the window center from the corners and reduce the height to go through the bottom half.

IV. Conclusion

In this project we were able to demonstrate that a combination of robust deep learning neural nets and classical thresholding approaches could be used to tackle the obstacle course for the autonomous drone race. We discussed the startegies used to pass each of the phases in the course and how we combine them.
Please use the following link to look at a successful demo run on the track for a random arrangement of obstacles: link.
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 was able to clear the track in different arrangements consistently across multiple runs. Further we could explore better strategies to reduce the time taken to cover the track in future.

References

[1] Teed, Zachary, and Jia Deng. "Raft: Recurrent all-pairs field transforms for optical flow." Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16. Springer International Publishing, 2020.

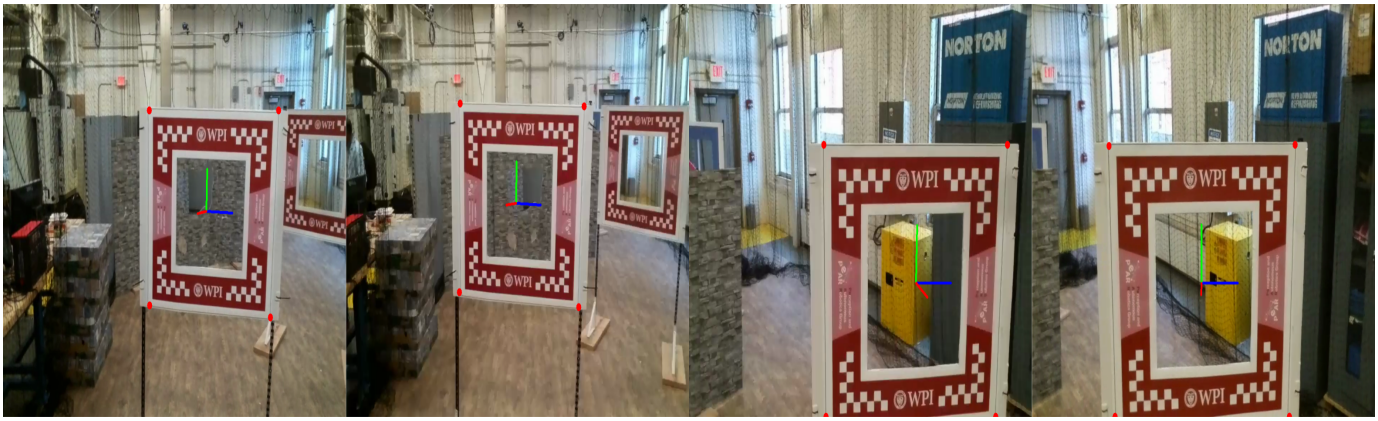

Fig. 3. Left to Right: Column 1: Drone detects the corners and center axes of the race window 1 through PnP algorithm.; Column 2: Drone performs a correction to ensure the window is at the center of its frame.; Column 3: Similarly drone detects race window 2; Column 4: Drone corrects its position to make window 2 at the center of the frame before going through it.

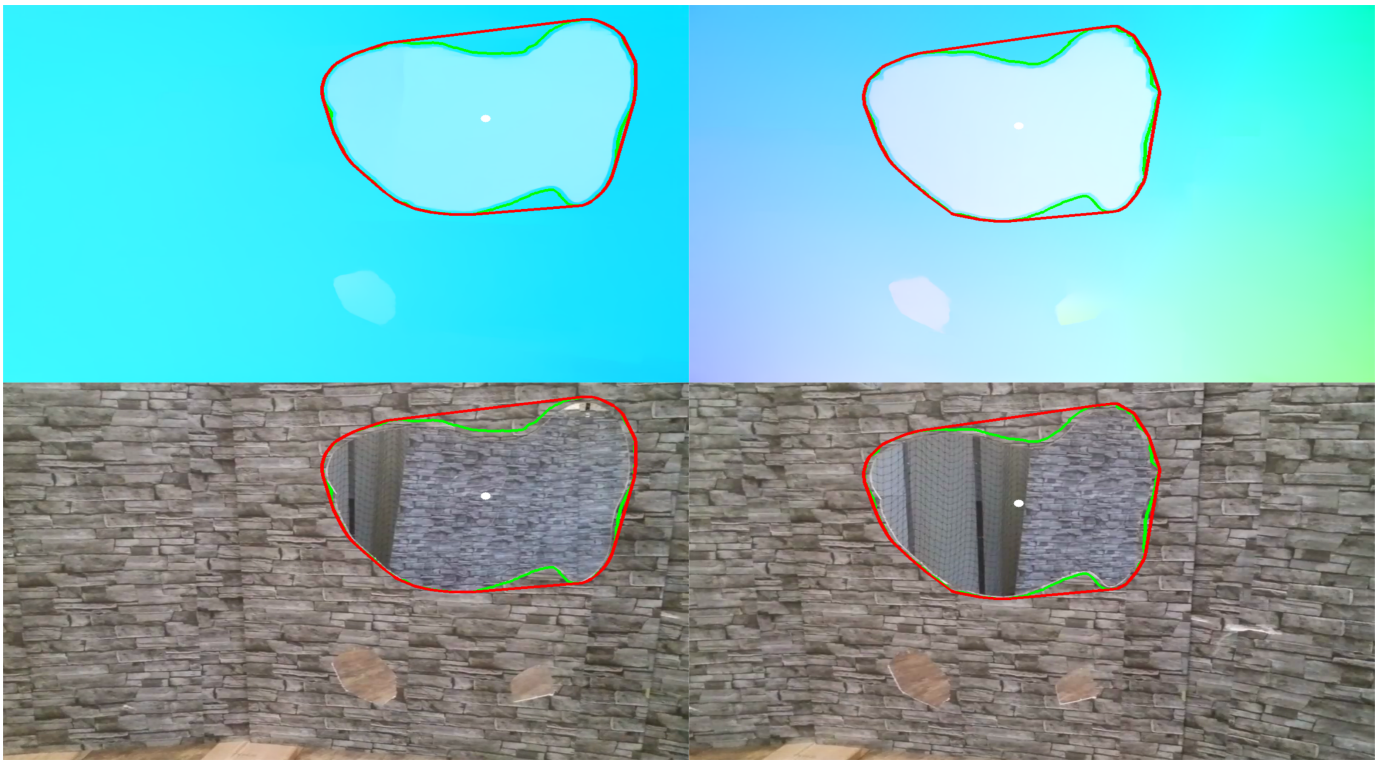

Fig. 4. Left to Right: Column 1: Drone detects the flow and identifies the gap center and contour. But the drone is not aligned with gap center.; Column 2: Drone performs a correction and now aligns its image center with the gap center and is ready to go through the gap.

Fig. 5. Left to Right: Column 1: Drone camera frame of the dynamic window; Column 2: HSV thresholded mask of hand and window with corners identified. In this image the angle of hand is measured w.r.t. vertical cyan hand in counter clock-wise direction. So our current angle as written is 197.67 degrees. Column 3: 3D coordinate and axes of window center obtained from PnP solution.