The Final Drone Race

Ankush Singh Bhardwaj
abhardwaj@wpi.edu

Sri Lakshmi Hasitha Bachimanchi
sbachimanchi@wpi.edu

Anuj Pradeep Pai Raikar
apairaikar@wpi.edu

***Abstract*—This project focuses on guiding a DJI Tello Edu quadrotor through a complex obstacle course, incorporating algorithms from previous projects (1 through 4), navigating the drone through different windows. Equipped with sensors like RGB and grayscale cameras, IMU, and altimeter, the quadrotor gathers essential data using the DJITelloPy package. The obstacle course involves three stages, including navigating racing windows, an unknown-shaped window, and a dynamic window with a rotating clock-like hand. The primary goal is for the drone to successfully navigate all windows in each stage, showcasing the adaptability of the DJI Tello Edu in dynamic and unpredictable environments.**

I. Environment

The test track is organized into three distinct stages, each presenting unique challenges for the DJI Tello Edu quadrotor. In the first stage, the quadrotor takes off from the heli-pad/origin and maneuvers through a set of two drone racing windows with checkerboard patterns on the corners, similar to those in project 3. Moving to the second stage, an unknown-shaped window is introduced, similar to project 4. Here, the quadrotor must navigate through this unfamiliar window as quickly as possible, again avoiding collisions with the background. The third and the last stage introduces a dynamic window with a rotating cyan square and a pink clock-like hand in the middle. The rotation speed of the hand is fixed but unknown. Navigating through this dynamic window poses a challenge of timing and precision, with the quadrotor required to fly through the window swiftly and without collisions. These varied stages collectively form a comprehensive obstacle course, testing the quadrotor's capabilities in speed, adaptability, and precise maneuvering (refer to Fig. 1).

II. Implementation

A. *Stage 1 - Racing Windows*

In this initial stage, the objective was to navigate the DJI Tello Edu quadrotor through a challenging course featuring a rectangular board with distinctive features, of logos and a unique checkerboard pattern.

1) Semantic Segmentation - Neural Network: We opted for a PyTorch-based U-Net to perform semantic segmentation, with a focus on identifying the unique checkerboard pattern located at the corners of the windows (refer to Fig. 2). Training the neural network was a meticulous process, starting from scratch on the WPI Turing clusters. This specialized approach ensured the network's adaptability to the intricate task at hand.

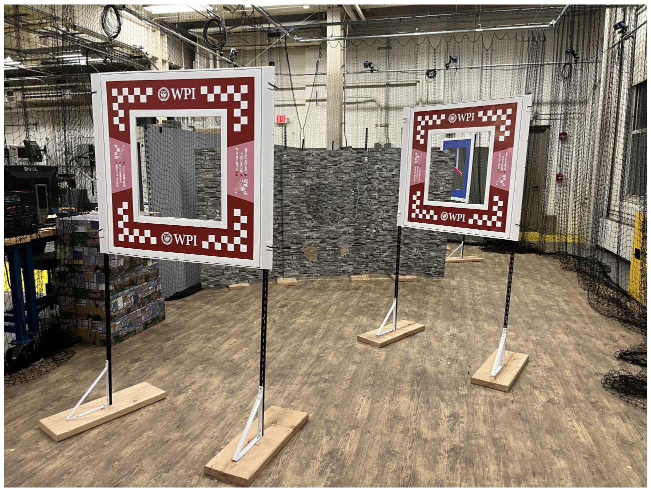
Fig. 1. Test Track

To generate a robust dataset for training, we utilized Blender, creating a diverse set of scenarios encompassing various camera angles, lighting conditions, and window configurations. Augmentations involved alpha matting, and the dataset was further enriched by compositing with real background images, adding an extra layer of complexity to the training data. The creation of binary masks for training involved storing 2D pixel coordinates of window corners in JSON files, mapped from their respective 3D coordinates. These binary masks served as ground truth data for the network, providing labeled information crucial for accurate segmentation.

2) Camera Calibration - Ensuring Precision: To ensure accurate projection of the 3D world onto the 2D image plane, the DJI Tello Edu's camera underwent calibration. Leveraging Matlab's Calibration toolbox and a printed checkerboard pattern, we estimated the camera's intrinsic parameters, including focal lengths, principal point, and distortion parameters. This calibration process laid the foundation for subsequent accurate 3D pose estimation.

3) Corner Detection - Integrating Classical Computer Vision: Post-segmentation, classical computer vision approaches were implemented for precise corner detection. Utilizing cv2.findcontour and bounding box techniques, we determined the corners of the segmented window. This integration of neural network segmentation and classical CV methods enhanced the accuracy of corner detection, a crucial

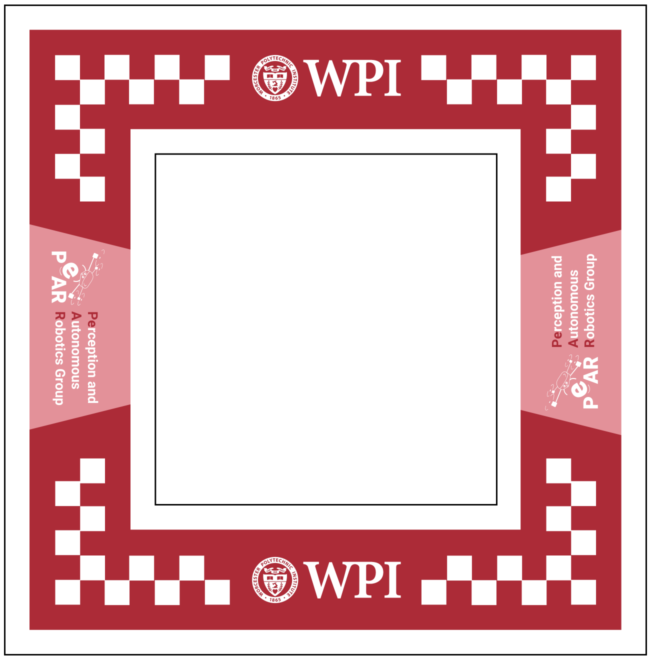

Fig. 2. Stage 1 Window

step in subsequent 3D pose estimation.

4) 3D Pose Estimation - Integrating Neural and CV Approaches: The final stage involved estimating the 3D pose of the window. Applying the cv2.solvepnp function, we utilized the calibrated camera parameters and pixel coordinates obtained through classical CV approaches. This integration of neural network segmentation and classical computer vision methods proved to be an effective strategy, providing the quadrotor with precise information about the window's position in 3D space.

B. Stage 2 - Unknown Window

Stage 2 presents an obstacle course with textured boards featuring irregularly shaped gaps of different sizes. The challenge involves accurate background-foreground segmentation and navigating through these gaps without collisions (refer to Fig. 3).

1) Perception Stack - Optical Flow: Our approach involves taking pairs of images while the drone moves along a wall with gaps. By doing this, we make sure the drone captures all the irregular gaps in its camera view. After the initial image, we command the drone to quickly move horizontally and capture a second image. These images become important for the next steps. We rely on optical flow, a concept in computer vision that helps us understand how pixels move between frames. It tracks pixel-level motion patterns between consecutive frames. We used SPyNet algorithm to calculate this flow, enabling the drone to grasp how it moves relative to its surroundings quickly. SPyNet is a spatial pyramid network trained on extensive datasets for obtaining dense optical flow offering detailed insights into the environment's motion field. SPyNet is trained to do this and worked well for our drone images.

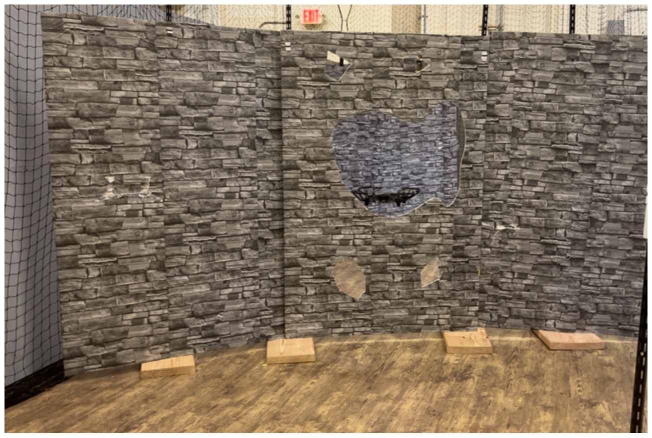

Fig. 3. Stage 2 Window

2) Background-Foreground Segmentation: The optical flow gives us a .flo optical flow analysis file showing how things move. By finding the minimum optical flow within gap boundaries and creating a grayscale map, we can separate the gaps from the wall. This map is turned into a binary mask, helping us find the boundaries of the gaps. We use this to find the biggest gap and figure out its center.

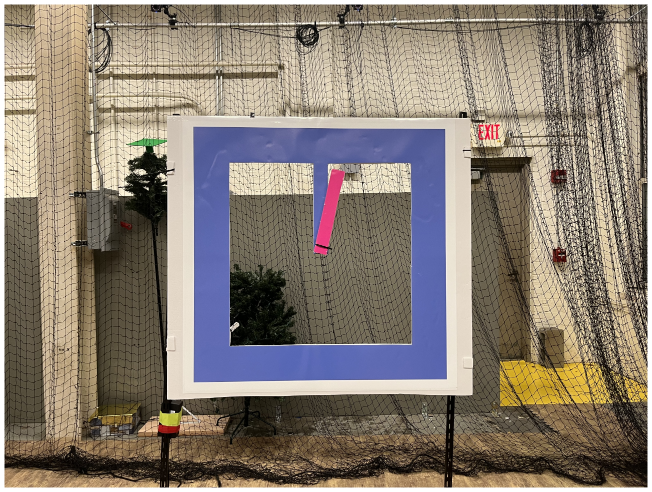

Fig. 4. Stage 3 Window

3) Safe Waypoint Detection and Navigation: Identifying the biggest gap is crucial for safe navigation. We use contours and edge detection to make sure we get it right. Once we know where the biggest gap is, we align its center with the image center as a navigation waypoint. This alignment guides the drone safely through the gap as it moves forward.

C. *Stage 3 - Dynamic Window*

In the third stage of our obstacle course, the drone encounters a dynamic window with a rotating clock-like hand. To address this challenge, we incorporated color thresholding, a fundamental technique for object detection. This approach enables the drone to identify specific colors within an image by defining a color range, proving essential for detecting objects based on their unique color characteristics (refer to Fig. 4).

1) Color Thresholding: We employed color thresholding as our foundational technique using hsv, allowing the drone to distinguish between the blue outer frame of the window and the rotating pink hand. By defining color ranges for both elements, the drone effectively identifies and navigates through the predefined windows. To precisely outline the detected objects within the window, we integrated contour detection into our color thresholding approach. We utilized the OpenCV library to find contours corresponding to the blue window and pink hand. This involved processing the color threshold masks to identify contours for each element independently.

2) Gap Detection: After detecting window and the rotating hand, we utilized the Canny edge detector and Hough Transform to identify lines within the masked image. The calculated angles of these lines with respect to the horizontal axis provide crucial information about the orientation of objects within the window. By overlaying these lines onto the original image, the drone gains insights into the spatial configuration of the dynamic window.

The integration of color thresholding with edge and hough transform empowers the drone to not only detect objects within the window but also discern the orientation of obstacles using angle calculation. This capability is pivotal for successful navigation through the dynamic and unpredictable environments encountered in our obstacle course. The resulting images, including the combined mask, edges, and lines drawn on the original image, serve as valuable visual aids in understanding the drone's decision-making process during navigation (refer to Fig. 5, 6).

III. Testing

To assess the viability of our implementations across all stages, thorough testing was conducted on the DJI Tello Edu drone, enhanced by the Jetson Nano platform. The drone's performance was rigorously evaluated in diverse scenarios, starting with Stage 1—Racing Windows, where precise navigation through designated windows was measured for speed and accuracy. In Stage 2—Unknown Window, the drone adeptly maneuvered through irregular gaps, showcasing the effectiveness of the optical flow-based navigation method. In Stage 3—Dynamic Window, achieved with color thresholding and angle calculations. Comprehensive testing encompassed overall navigation and integration, confirming the success of combining computer vision, deep learning, and drone control algorithms. The results underscore the robustness of our implementations, demonstrating adaptive and responsive navigation through complex obstacle courses on the DJI Tello Edu platform with Jetson Nano integration (Refer to subsequent figures).

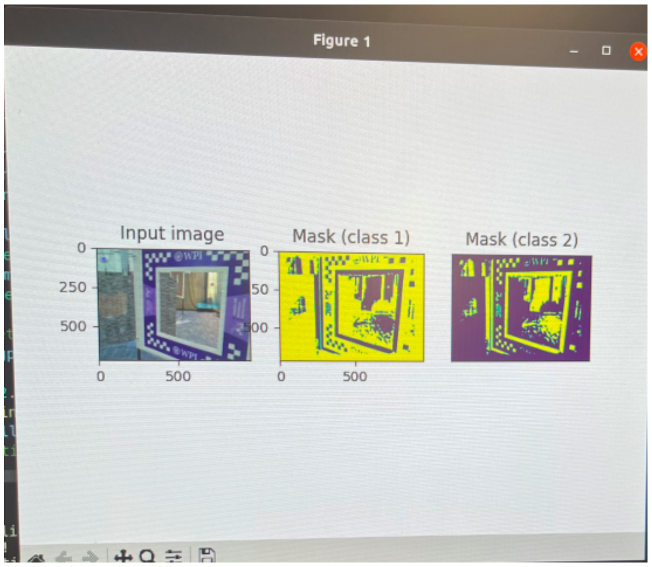

Fig. 5. Stage 1

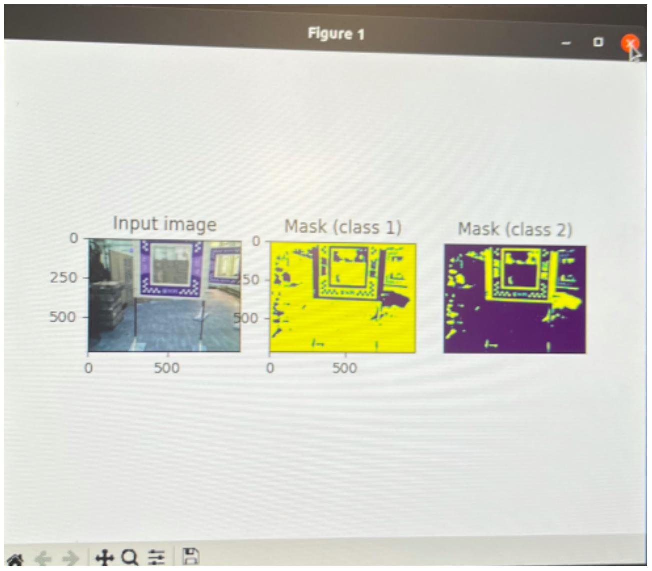

Fig. 6. Stage 1

IV. References

1 Principles of Robot Motion: Theory, Algorithms, and Implementations" by Howie Choset, Kevin M. Lynch, et al.
2 https://docs.px4.io/main/en/flight_stack/controller_diagrams.html
3 https://github.com/anuragranj/spynet

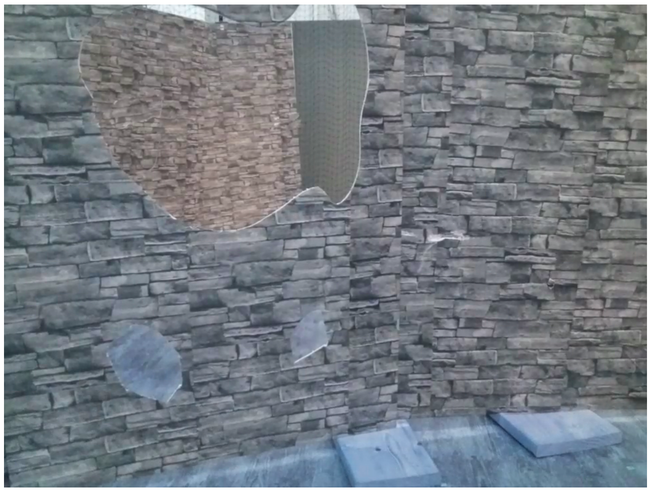
Fig. 7. Stage 2

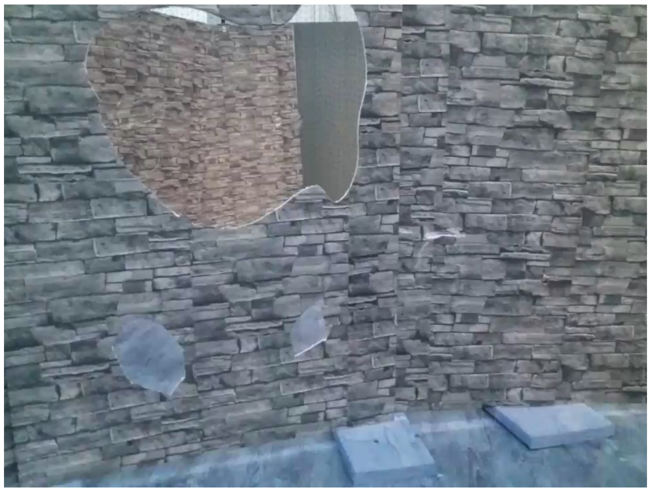
Fig. 8. Stage 2

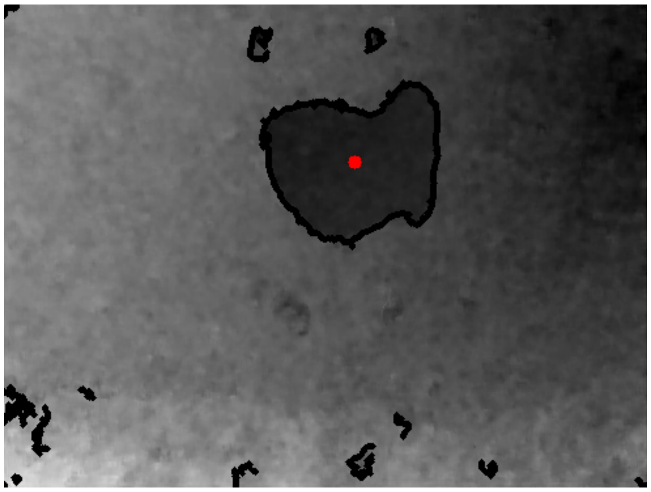
Fig. 9. Stage 2

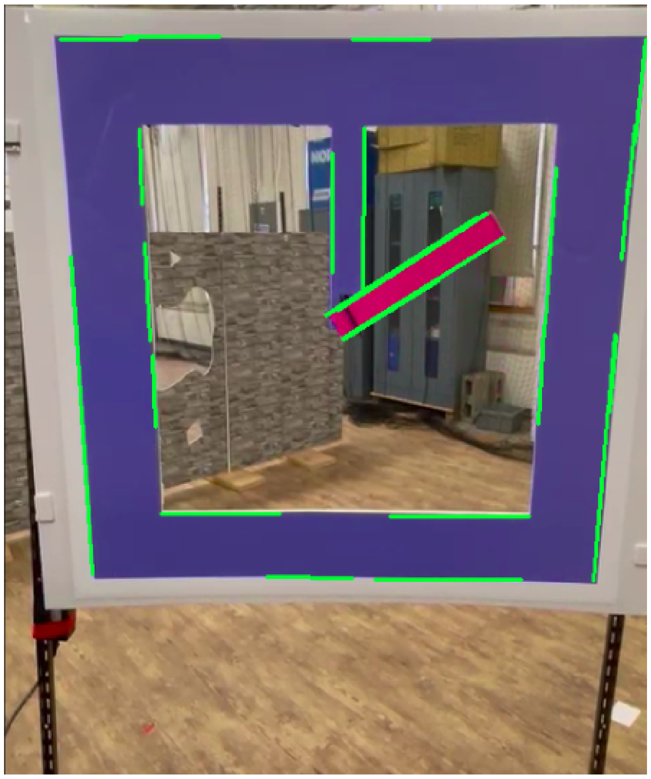
Fig. 10. Stage 3

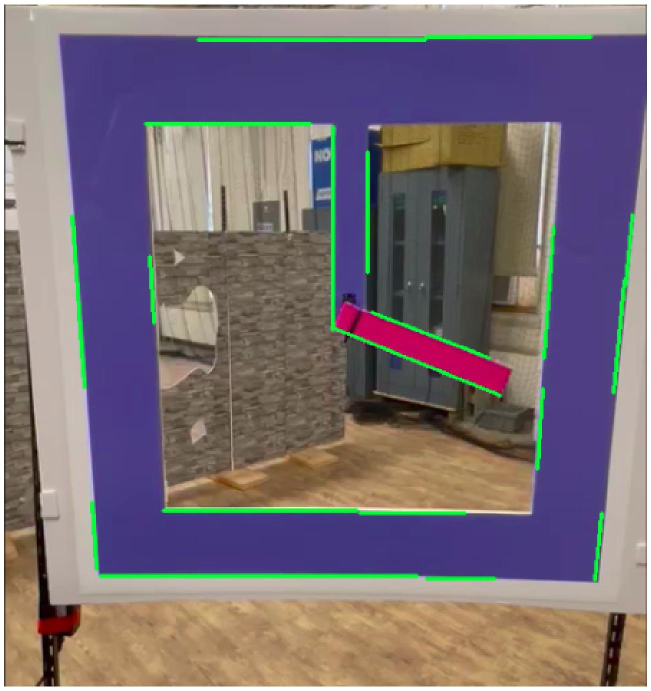
Fig. 11. Stage 3

4 https://github.com/damiafuentes/DJITelloPy/tree/master/djitellopy

5 https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello SDK 2.0 User Guide.pdf

6 https://www.deeplearningbook.org/

7 https://learnopencv.com/