Team Apache Stealth: The Final Race !!

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 addresses the challenge of enabling a small drone to autonomously navigate through an different kinds of windows in a diverse environment. Given the computational and weight limitations inherent to small drone platforms, the project focuses on developing innovative, resource-efficient algorithms suitable for the NVIDIA Jetson Orin Nano. The key components include a versatile perception stack for detecting and tracking window gaps, and a dynamic planning and control stack capable of generating safe navigation paths. The project emphasizes the use of simulated environments for preliminary testing, employing a variety of window shapes and textures to ensure robustness. The final evaluation involves a live demonstration where the drones navigate through unpredictably placed windows, with an emphasis on accuracy and speed of navigation. The project's success will be measured by the drone's ability to safely and swiftly identify and traverse the largest available gap without reliance on cloud computing resources, showcasing the potential of compact, intelligent autonomous systems in complex real-world scenarios.**

I. Introduction

There total 3 stages with different kinds of windows to pass through. we don't have the prior information of position and orientation. we are porivided a DJI tello with a front facing RGB color camera, a down facing grayscale camera, an IMU along with the altimeter on-board. Only utilizing these sensor we need to pass through the window.

A. Stage 1

In the first stage, we are presented with two checker-patterned racing windows. These windows are nearly square, featuring a checkerboard design along their edges. Each checkerboard pattern consists of a 7x6 grid. The distance from the checkerboard pattern to any edge of the window is equal to the size of one square in the checkerboard, highlighted in yellow in the referenced figure. Our task is to use neural networks to detect these checker-patterned windows, determine their orientation and pose, and then navigate through them successfully.

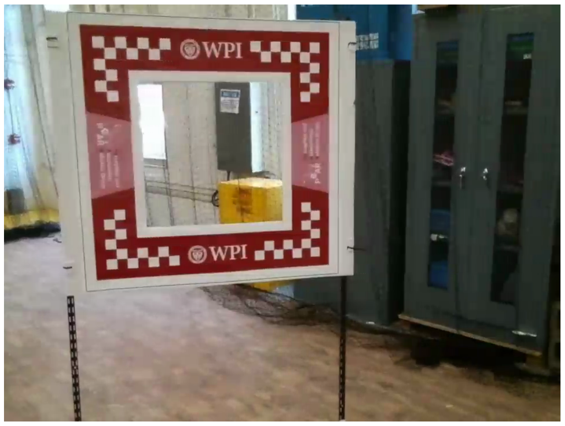

Fig. 1: Environment Image in Blender

B. Stage 2

The environment features a window of irregular shape, constructed from foamcore with various textures applied. The surrounding wall is also textured, and the background may have textures that are either identical or different from those on the window. While the colors of the window and the background might be similar, their patterns will not be exactly the same. The window, essentially flat, may contain several holes or gaps. In such cases, the objective is to identify and navigate through the largest gap. This largest opening, if there are multiple, will be sufficiently sized to ensure the safe passage of the quadrotor drone.

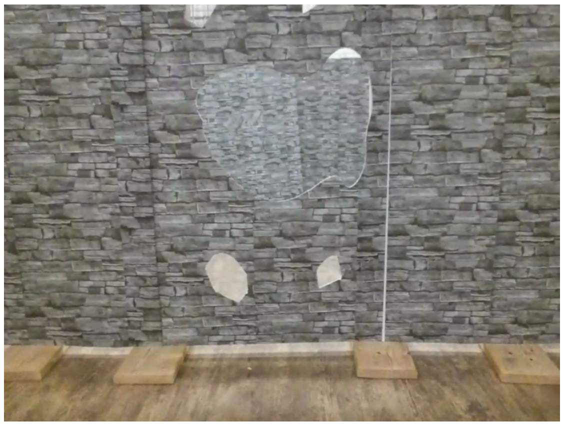
Fig. 2: Hole the wall

C. Stage 3

In this stage, our mission involved navigating through a dynamic window characterized by a cyan square frame, within which a pink clock-like hand was rotating at a fixed, yet unknown, speed. Our primary goal was to pass through this moving window swiftly and safely, avoiding any collisions. We had prior knowledge of the window's dimensions.

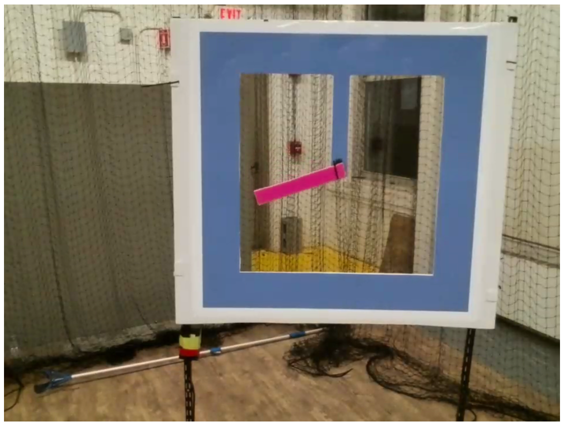
Fig. 3: Hole the wall

Link To Videos: **Click Here**

II. Hardware Specifications

The DJI Tello Edu drone, equipped with a 5-megapixel camera (2592x1936 photo resolution, 960x720 video streaming), weighs 82.6 grams and has a 13-minute flight capability with a 1.1 Ah/3.8 V battery. It operates within a 100-meter range and up to 10-meter altitude, using the DJITelloPy Edu Python library for programming. Complementing it is the NVIDIA Jetson Orin Nano, a compact AI and edge computing module, essential for processing the drone's real-time image processing and autonomous navigation tasks, ensuring efficient and precise visual data handling in complex environments.

III. STAGE 1: CHECKERS WINDOW

At this point, our objective is to identify the checker window within the environment and ascertain its orientation, allowing us to navigate through it successfully.

*1) **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.

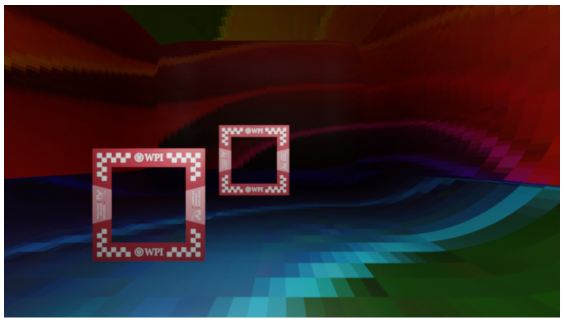

Fig. 4: Simulated Environment in Blender

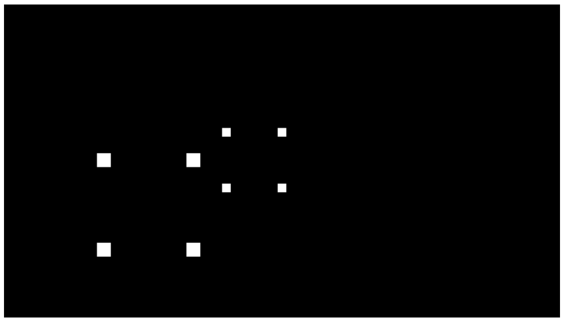

Fig. 5: Corresponding Label Image of the of the Simulated Environment

*2) **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.

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.

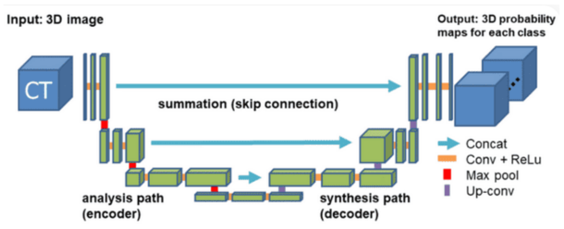

Fig. 6: U-NET Architecture

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]).

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.

*3) **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.

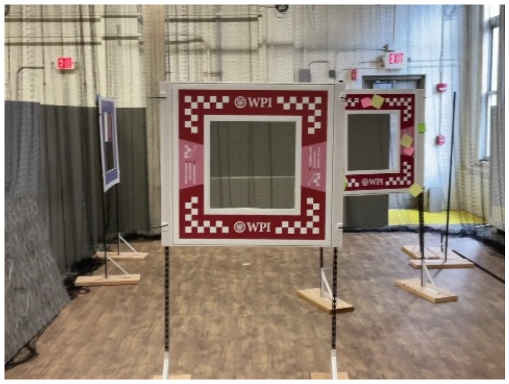

Fig. 7: Image going into the Network

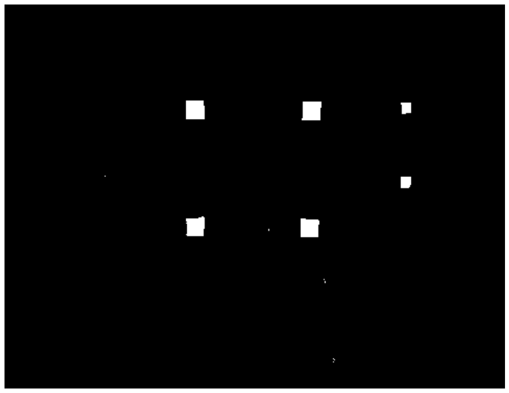

Fig. 8: Image coming out of the Network

*4) **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.

*5) **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

IV. Stage 2: Unknown Gap Window

The environment has an arbitrary shaped window(s) which can be 'seen' from the origin location. The window is made of foamcore with texture stuck on it. we do not know the texture prior but we will ensure that the windows are not textureless or they dont fully blend into the background texture. Also, the colors on the window and the background can be similar but patterns will not be exactly the same. The board can be assumed to be nearly planar and can have multiple holes/gaps, in which case we have to choose the largest one to fly through. The largest gap (if multiple gaps are present) will be large enough for the quadrotor to fly through it safely.

*1) **Sensors and Data Acquisition**:* The data acquisition for this project relies on the monocular camera equipped on the DJI Tello Edu drone, as detailed in the Hardware Specifications section. The primary challenge in data acquisition was positioning the drone so that the camera's field of view encompasses the potential gaps in the wall. This positioning is critical for ensuring that the subsequent image processing and gap detection algorithms have the necessary visual data to identify viable paths for navigation.

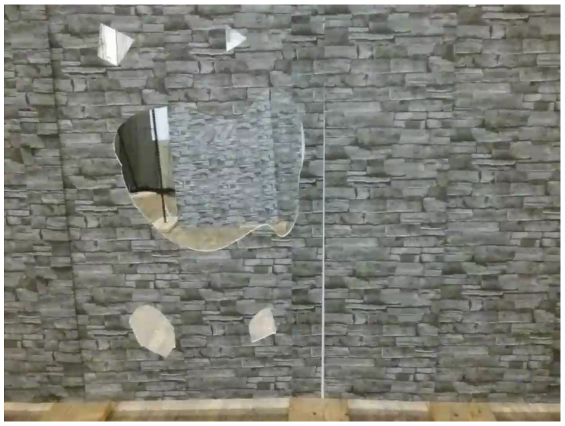
Fig. 9: Positioning the Drone such that the Gap is in FOV of Drone's Camera

*2) **Image Processing**:* The cornerstone of our image processing approach is the inference of optical flow using the SPyNet network. Optical flow, a concept in computer vision and image processing, describes the pattern of apparent motion of objects, surfaces, and edges in a visual scene, caused by the relative movement between an observer and the scene. SPyNet's processing time ranges from 0.2 to 0.5 seconds per inference, striking a balance between speed and accuracy. In the realm of computational considerations, SPyNet was selected over other candidates like FlowNet 2.0 and PWC-Net for its optimal balance between rapid inference time and adequate accuracy. Classical methods of optical flow determination, though expedient, lacked the robustness and precision provided by contemporary deep learning-based approaches. We also investigated other deep-learning models for optical flow, such as FlowNet 2.0 and PWC-Net. However, these models are computationally demanding, with over 100 million parameters, making them resource-intensive. Hence, We chose SpyNet for our needs, and to boost its precision, we conducted several optical flow evaluations. Through this process, we pinpointed and chose the region most likely to contain a gap by analyzing the gap probability map associated with each optical flow map. This compensates for the potential shakiness or distortion in the drone's captured images. This methodology ensures that the final optical flow output is both reliable and representative of the actual scene dynamics, facilitating accurate gap identification. The conceptual foundation for this approach is derived from Dr. Nitin Sanket's GapFlyt paper, which addresses a similar challenge in drone navigation.

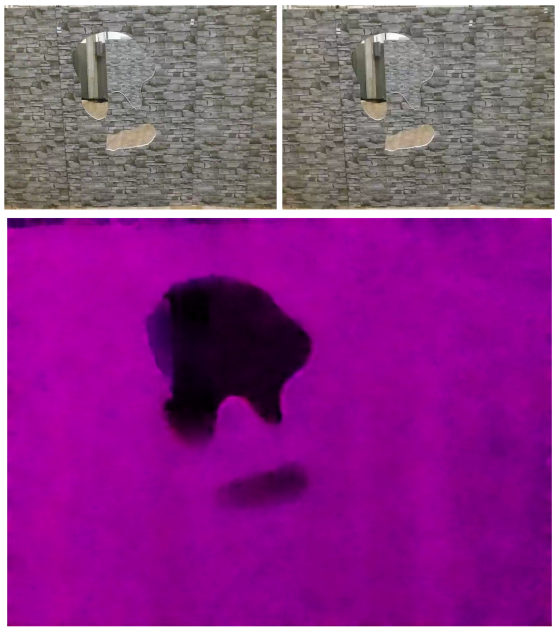
Fig. 10: [Top left] - **Image1**, [Top Right] - **Image2**, [Bottom] - **Optical Flow of Image1 and Image2**

To generate the probability map for each optical flow, we applied the Otsu thresholding method. This technique effectively highlights the regions most likely to represent gaps. Following this, we overlaid the images, employing binary operations on each pixel to accurately identify the gaps. This approach proved to be highly effective, consistently yielding reliable results even when dealing with suboptimal/bad optical flow maps.

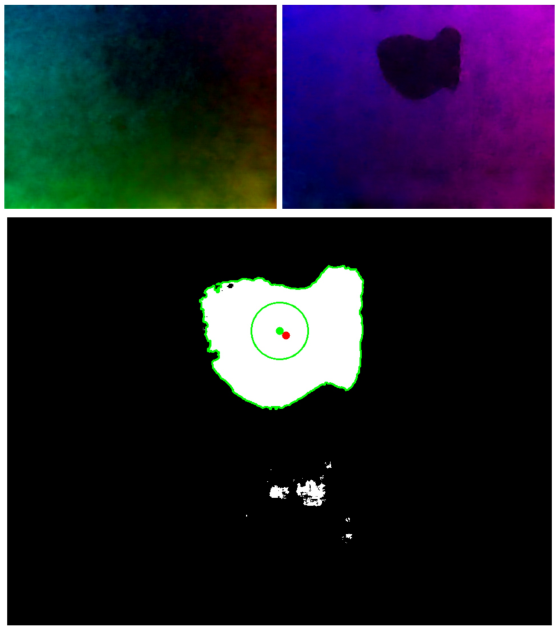
Fig. 11: [Top left] - **Optical Flow 1**, [Top Right] - **Optical Flow 2**, [Bottom] - **Binary Mask Image obtained after Thresholding with Position Markers**

Figure 5 illustrates that the left optical flow map lacked accuracy, primarily due to image shakiness and distortion from the drone's camera. Similarly, the right side image also hinted at a potential gap near the bottom. However, when these maps were superimposed, the resulting composite enabled us to obtain a precise binary mask. This mask effectively highlighted the areas with the highest probability of being gaps, as demonstrated in the figure. In the binary image, the green dot represents the center of the image, which has been adjusted upwards by 150 pixels for 960 X 720 images (determined based on an experimental basis) to account for the camera's tilt. Meanwhile, the red dot indicates the center of the contour.

A. *Softwares and Frameworks*

The project employed Python and PyTorch for programming and deep learning tasks, respectively, with OpenCV for image processing. DJITelloPy was used for drone control, ensuring a cohesive and efficient system for real-time autonomous drone navigation.

B. Path Planning and Control

1) Initial Positioning and Movement: Considering the wall's specified distance of 1.8 to 3 meters from the origin, with a possible tilt angle of -20 to +20 degrees, the drone's initial strategy involves a predetermined forward movement. This initial maneuver is designed to compensate for potential takeoff errors or hardware inconsistencies, ensuring that the drone starts from a consistent and reliable position relative to the wall.

2) Gap Detection and Alignment: Utilizing the Perception Stack, the drone computes the optical flow from its camera feed and post-processes this data to identify the center of the largest gap in the wall. As detailed in the Perception Stack section, the processed output is a binary image with a red dot marking the estimated gap center and a green dot at the image center, surrounded by a threshold-indicating green circle. This setup is integral to the subsequent visual servoing algorithm.

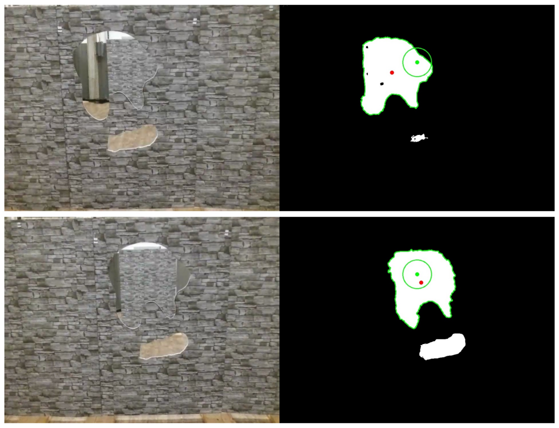

Fig. 12: Visual Servoing in Action. [Top] - **Drone not aligned with Gap Center.** [Bottom] - **Drone Aligned with Gap Center**

3) Visual Servoing Implementation: Visual servoing is a critical component of the drone's navigation strategy. It operates by ensuring that the red dot (gap center in the image) is aligned within the green circle (the threshold around the image center). When the alignment is achieved, indicating that the drone is positioned correctly relative to the gap, a forward command is issued to the drone.
The visual servoing mechanism is executed using the position control feature in the DJITelloPy library. The process involves calculating the distance between the red and green dots on the x and y axes of the image plane. As the perception stack does not provide depth information, an experimental approach was taken to estimate a workable depth between the drone and the wall, considering the known distance range. This estimation was used to compute a scaling factor, transforming the pixel distances into metric units. The drone then moves according to these calculated distances, repeating the process until the desired alignment is achieved.

4) Depth Estimation and Scaling Factor: The depth estimation process is crucial for the success of the visual servoing strategy. Without direct depth information from the perception stack, the team conducted multiple trials to identify a practical distance between the drone and the wall. This experimental distance was then used to derive a scaling factor, crucial for converting the distances on the image plane to real-world metrics. This scaling factor plays a pivotal role in ensuring that the drone's movements are precise and aligned with the real-world dimensions of the environment.

In summary, the planning and control strategy of the drone involves an initial forward movement for position normalization, followed by gap detection and alignment using a sophisticated visual servoing approach. This approach leverages experimental depth estimation and scaling to translate image plane measurements into real-world navigational commands, ensuring precise and effective movement toward the target gap.

C. Important Frames from the Live Feed of Demo

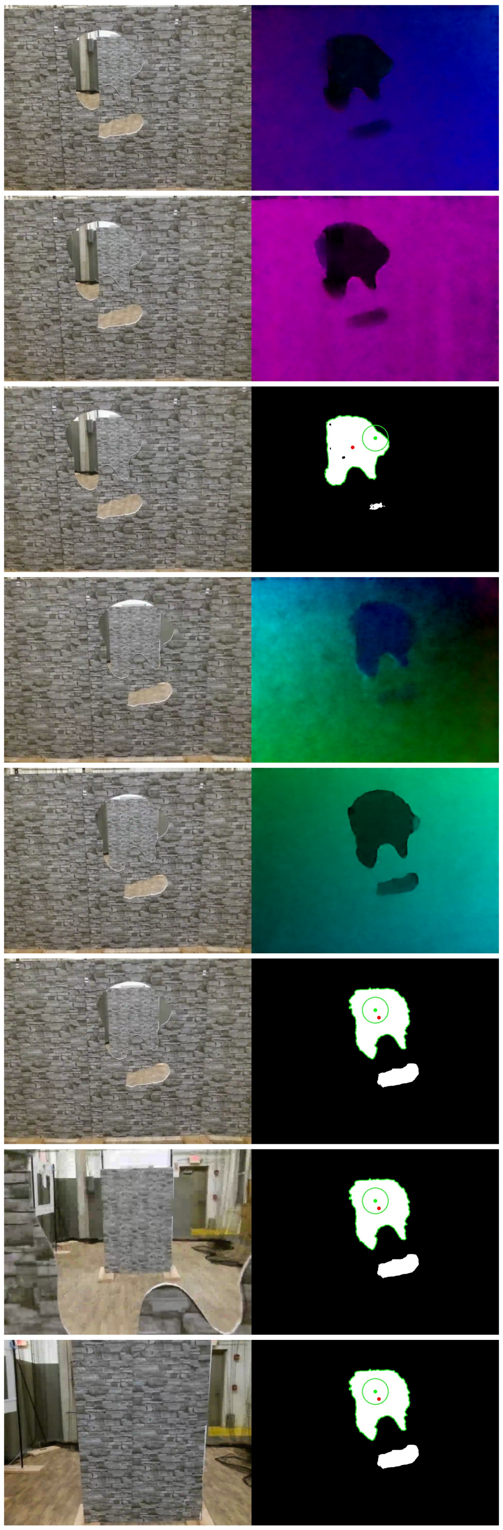
Fig. 13: Snippets from the Test Run

V. Stage 3: Dynamic Window

In this stage, our mission involved navigating through a dynamic window characterized by a cyan square frame, within which a pink clock-like hand was rotating at a fixed, yet unknown, speed. Our primary goal was to pass through this moving window swiftly and safely, avoiding any collisions. We had prior knowledge of the window's dimensions.

To isolate specific colors in an image, we used an HSV (Hue, Saturation, Value) color segmentation approach. We segmented a cyan window by applying HSV thresholds for blue (Hue: 70-148, Saturation: 65-255, Value: 51-255) and a pink hand by using thresholds for pink (Hue: 120-179, Saturation: 90-255, Value: 0-255). To continuously track the hand's position, we overlaid a clock-like reference on the image. This graphical overlay provided real-time information on the hand's angle relative to a central line passing through the window, aiding in our navigation in this dynamic setting.

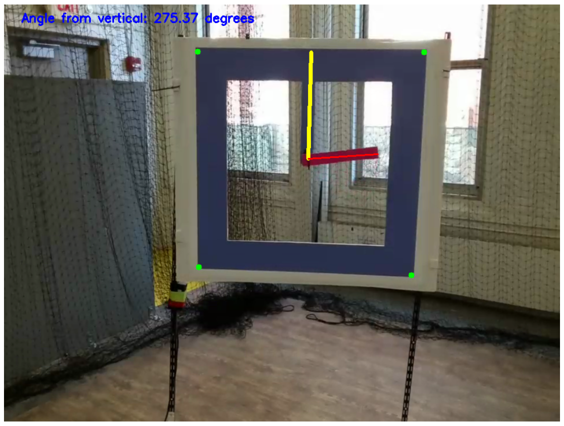

Fig. 14: Orientation of hand (angle calculated anti-clockwise)

1) Pose Estimation: To determine the pose of the window we utilized 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.

2) Planning: To accurately time the drone's passage through the dynamic window, we considered the approximate distance of the drone from the window. We issued a command for the drone to pass through when the angle of the hand was between 135 to 225 degrees from the left. This specific angle range was determined to be the safest zone, allowing the drone to traverse without colliding with the moving hand in the window.

VI. Conclusions

This project successfully demonstrated the capability of a small drone, specifically the DJI Tello Edu, to autonomously navigate through different types of windows in a controlled environment. Employing the NVIDIA Jetson Orin Nano, we developed a robust perception stack that utilizes Neural nets, optical flow, classical computer vision appraches for gap detection and a visual servoing approach for precise navigation. Our approach effectively overcomes the challenges posed by limited computational resources and the constraints of using a monocular camera system. The live demonstrations and tests confirmed the viability of our system in accurately identifying and traversing through the largest gaps in various window shapes, highlighting the potential of compact drones in complex navigation tasks.

VII. Acknowledgment

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

References

[1] RBE595-Hands-On Autonomous Robotics Course Web-site**Link**

[2] DJITelloPy **Link**

[3] Optical Flow Estimation using a Spatial Pyramid Network **Link**

[4] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller and Y. Aloimonos, "GapFlyt: Active Vision Based Minimalist Structure-Less Gap Detection For Quadrotor Flight," in IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 2799-2806, Oct. 2018, doi: 10.1109/LRA.2018.2843445. **Link**