P5: The Final Race

Mayank Bansal
Robotics Engineering
Worcester Polytechnic Institute
Email: mbansal1@wpi.edu

Siyuan 'Oliver' Huang
Robotics Engineering
Worcester Polytechnic Institute
Email: shuang4@wpi.edu

Miheer Diwan
Robotics Engineering
Worcester Polytechnic Institute
Email: msdiwan@wpi.edu

***Abstract*—This project, segmented into three distinct phases, explores the development of an autonomous drone navigation system. In the first phase, we focus on navigating through multiple windows, drawing inspiration from the Alpha Pilot competition. We employ YOLOv8 for gate detection, trained exclusively on simulated data, coupled with Perspective-n-Point (PnP) for determining the gate's pose relative to the drone. This phase culminates in the successful integration of controls, planning, and hardware with the DJI TelloEDU drone. The second phase addresses the challenge of flying through an unknown, arbitrarily shaped gap. Here, we utilize the SPyNet optical flow detection algorithm, achieving nearly 100 percent accuracy in gate navigation. The final phase tackles navigating through a dynamic window with a central, rotating dial. A simple yet effective method of color thresholding and line fitting is employed to calculate the dial's angle, guiding the drone's timing to fly through the window. Finally we present our results and learnings throughout the duration of this course.**

I. Phase 1

A. Environment

The map consists of multiple gates placed at 3D poses known apriori from the environment file. The window locations are given in the format:

```
# boundary xmin ymin zmin xmax ymax zmax
# window x y z xdelta ydelta zdelta
qw qx qy qz xangdelta yangdelta zangdelta
boundary 0 0 0 45 35 6
window 1 1 1 0.2 0.2 0.2 0.52 0.85 0 0 5 5 5
```

- x, y, z represents the approximate center of the window in meters.
- $xdelta$, $ydelta$, $zdelta$ represents the variation in meters that is possible from x, y, z values.
- qw, qx, qy, qz represents the approximate orientation of the window as a quaternion.
- $xangdelta$, $yangdelta$, $zangdelta$ represents the ZYX Euler angle variation in degrees that is possible from the approximate orientation given.

B. Perception Stack

The first phase of this project dealt with the development of a robust Perception stack for detecting or segmenting windows in an unknown environment. For this, we decided to train a custom Instance segmentation model using YOLOv8.
Since the windows in the environment have different poses in the world, a simple object detection network would not have been able to provide us with accurate results. Additionally, the bounding boxes generated by such a network do not take into account the orientation of the windows or sometimes have overlaps when multiple windows are present in the same frame. Using instance segmentation, we were accurately able to detect the windows and generate segmentation masks for each window.

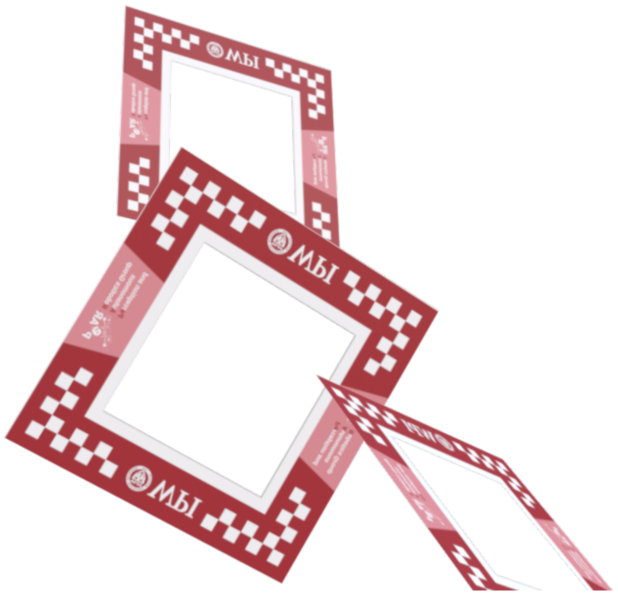
Fig. 1. Multiple gates used for dataset generation

1) Dataset Generation Using Blender: Having a good and unbiased dataset is key for obtaining great results with a deep neural network. To make this task more complicated, we only had access to simulated data. We created our dataset using only rendered images from Blender. Blender is a free and open-source 3D computer graphics software toolset. The images consisted of multiple windows spawned in random orientations. We also changed the camera pose and lighting conditions in each image to add more randomness to our dataset. Our original dataset consisted of 5000 images created using Blender. Since our dataset only consisted of simulation data, there may have been some intrinsic bias. To address

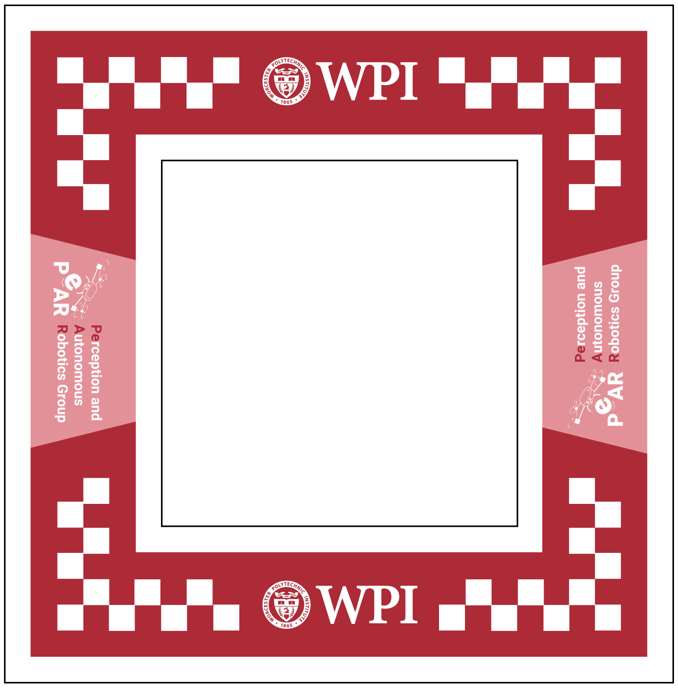
WPI
WPI
Perception and Autonomous Robotics Group
PeAR
Perception and Autonomous Robotics Group
PeAR

Fig. 2. Gate

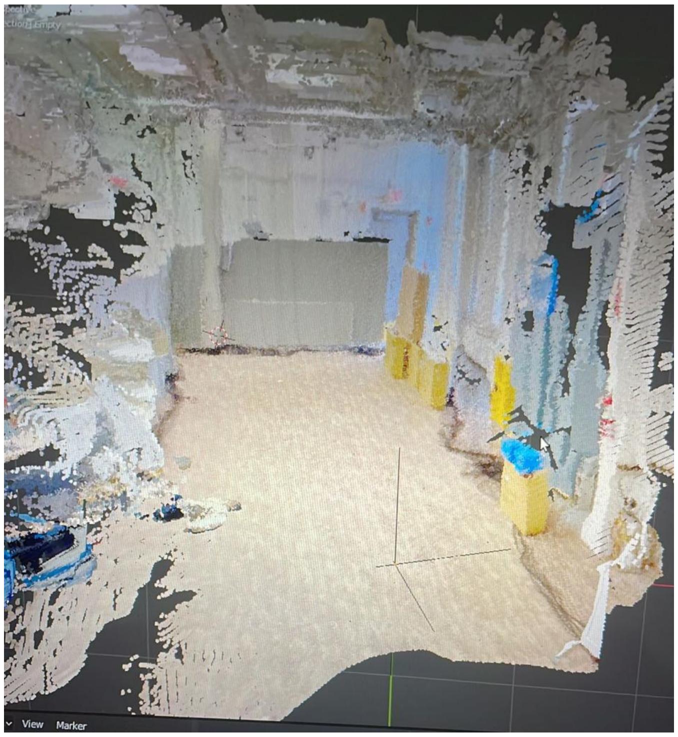

Fig. 3. GAUSSIAN SPLAT OF ENVIRONMENT

this problem, we used two approaches:

1) *3D Gaussian Splatting:* The first approach to solving the problem of intrinsic biases in popular literature is to use hyper-realistic simulated data. To this extent, we made use of Gaussian Splatting to recreate a 3D color point cloud of the actual environment and imported it into Blender. We spawned our window frames on this background and We used this GitHub repository to perform this task: 'Gaussian-Splatting-Windows'. While this approach worked very well for our test case, it could not be generalized across other backgrounds. So, we decided to proceed with the second approach as it made our network more robust.
2) *Domain Randomization:* Domain randomization is a technique for transferring Deep Neural Networks from Simulation to the Real World and helps us bridge the gap between simulation and reality. We took images from the FlyingChairs dataset and used them as backgrounds for our rendered images. Thus, we expanded our original dataset of 5000 images to 10000 images by adding new backgrounds to them.

2) Transformations and Augmentations: We used transformations and augmentations to expand our dataset from 10000 images to **24000 images**:

1) Auto-orientation
2) Grayscale
3) Brightness: Between -30
4) Blur: Up to 2.5px
5) Noise: Up to 10
6) Cutout: 15 boxes with 5

3) Instance Segmentation Using YOLOv8: We used the Roboflow API and Ultralytics YOLOv8 to train our custom model for window segmentation. YOLOv8 is a state-of-the-art object detection and image segmentation model created by Ultralytics in January 2023 and using the PyTorch framework. We trained our instance segmentation model using the pre-trained weights from the YOLOv8n-seg. YOLOv8n-seg is the lightest YOLOv8 model, has the fastest inference time, and was trained on the COCO dataset. The model was trained for 100 epochs

Dataset Division:

- Train Set: 21000 images
- Test Set: 1000 images
- Validation Set: 2000 images

We trained multiple models with different outputs and this helped us gauge what works best for our task. We started by training the model with just the four corners of the window as an input. While this model was good, the segmentation masks were not reliable in some cases where there was overlap or orientation changes.

The next model we trained only had the 4 corners of the window with the checkerboard pattern as the ground truth. This model was extremely accurate and robust. However, predicting which four corners belonged to the same window proved to be difficult. So, we changed our model again.

Our final model combined the previous two approaches and predicted the segmentation masks of two classes — the entire

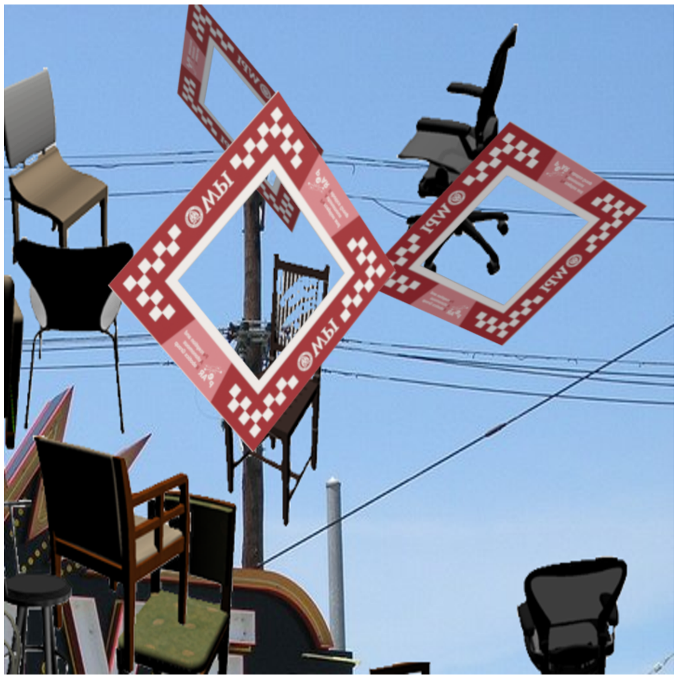
Fig. 4. Dataset with background image

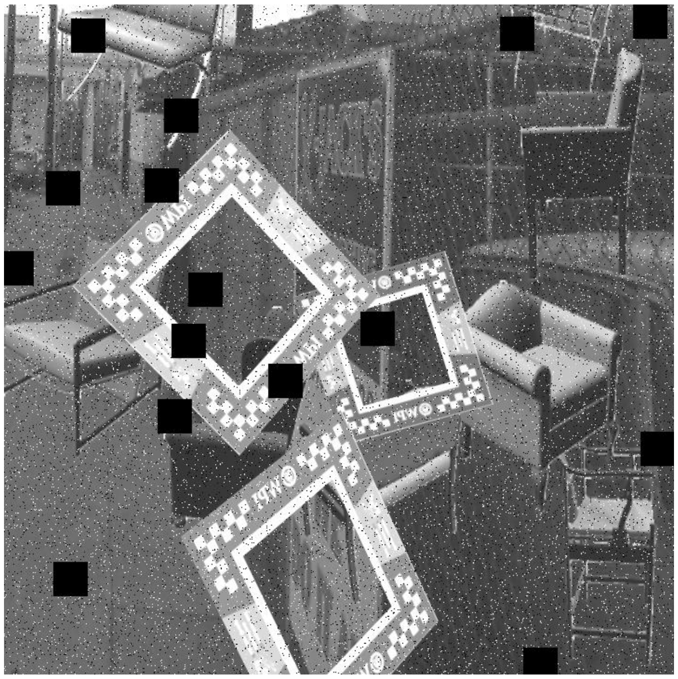
Fig. 5. Augmented training image

window and the four corners. This helped us develop a robust model which was able to predict the window masks accurately. Extracting the window and corner masks with some post-processing was a lot easier this way too. the next section talks about the post-processing techniques we used.

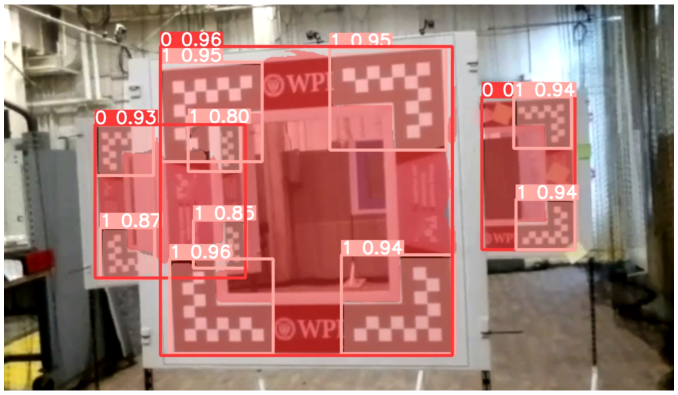

Fig. 6. Segmentation result of the network

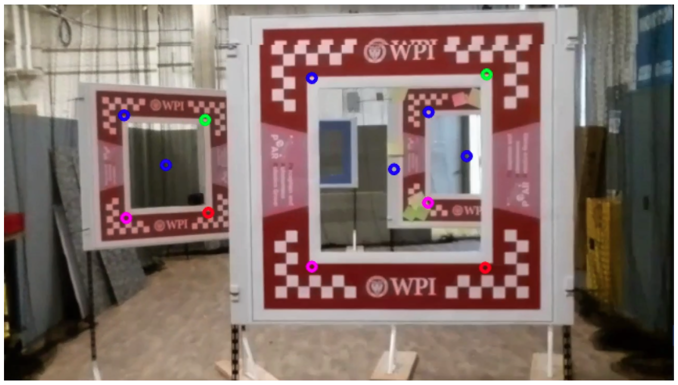
Fig. 7. Corners obtained after post-processing

4) Post-Processing Techniques: In our post-processing section, we are fitting a circle to each segment in order to determine which corners belong to which gate. After getting circles on each segment, we allot each corner to gate based on ratio of the distance of the center of the corner centre to the gate centre and the radius of the gate circle. If this ratio is between 0.6-0.8, this corner is assigned to that gate. This process is repeated for all the corners and gates.

5) Camera Calibration: Camera Calibration was done with the help of MATLAB and a checkerboard pattern to correct the distortions in the images captured by the DJI Tello EDU drone and predict the camera intrinsics.

6) PnP: Pnp (Perspective-n-Point) is the algorithm used to estimate the pose of the closest window with respect to the camera frame. We use the cv2.solvePnP() function in which the inputs are image co-ordinates of the corners of the closest window from top-left corner in anti-clockwise manner, the 3D world frame co-ordinates of the corresponding window corners in the same order, the intrinsic camera matrix of the camera on the Tello obtained from calibration process and the distortion co-efficients(assumed to be zero in our case). The output is the pose of the world frame with respect to the camera frame. This pose estimation can now be used to fly the drone past the window.

II. Phase 2

A. Environment

The environment has an arbitrarily shaped window(s) that can be 'seen' from the origin location. The window is made of foam core with texture stuck on it. The texture is not known prior to the flight but the windows are not textureless. Also, the colors on the window and the background could be similar but the patterns were not exactly the same. The board was nearly planar and had multiple holes/gaps. In this case, the goal was to fly through the largest gap. Furthermore, there were offsets of about +20 and -20 degrees in the yaw angle and a similar offset in the X-axis.

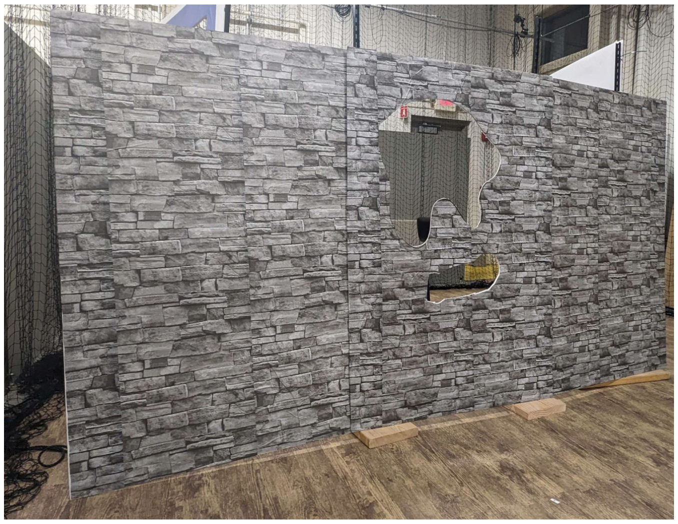

Fig. 8. Environment image

B. Gap Detection

1) Optical Flow: Optical flow is a technique used to understand image motion. It is usually applied to a series of images that have a small time step between them, for example, video frames. There are two types of optical flow - Sparse and Dense. Sparse optical flow follows the movement of only a few pixels while dense optical flow follows the movement of all the pixels in the image. To find the location of the gap, dense optical flow is more useful than sparse optical flow. The idea is that the pixels belonging to the background texture moves slower compared to the foreground texture pixels when drone is moved. This disparity in pixel flow (optical flow) will enable us to find the gap.

Initially, we tried using classical methods like Farneback algorithm to find the optical flow vector, but this method is pretty slow and the resultant vector field is not of a high resolution which makes it difficult to segment out the gap effectively. We then shifted to deep learning methods which are usually more accurate than classical methods.

We used the PyTorch implementation of Anurag Ranjan's SPyNet (Spatial Pyramid Network for Optical Flow). It is a light-weight network with fast inference speeds (We were getting each inference in around 0.5 seconds).

Link to Github Repository: pytorch-spynet

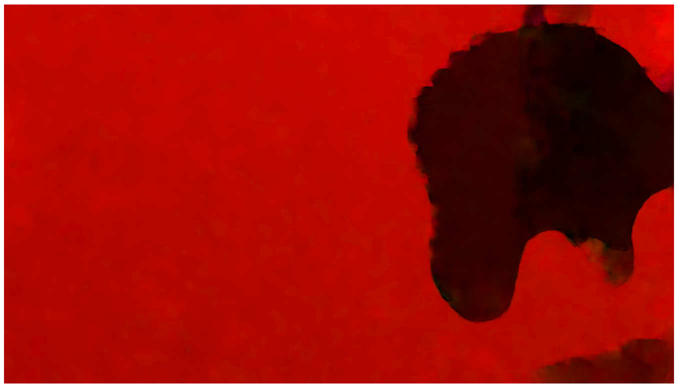

Fig. 9. Colorized Optical Flow

2) Methodology: The drone takes off and then moves a bit forward so that the full board is in view. We take two images in 0.5 seconds succession and then we then calculate the optical flow using SPyNet from these two images. Figure 9 shows the colored optical flow from the two images. The color(hue) is based on the angle of the optical flow vectors and the saturation is based on the magnitude of these vectors. Figure 10 shows the normalized magnitude of these vectors. Now we need to detect the largest contour and segment that out to find the largest gap. Initially we used Canny edge detection and thresholding on the normalized optical flow image to find the largest contour but this method was not very robust. We ended up using otsu + binary thresholding which decides the threshold value dynamically for the magnitude image and segments the darker parts based on this threshold value. Then, the contours are detected on this thresholded image and the largest contour is found along with its centroid. Figure 11 illustrates this step.

The next step is visual servoing. The primary logic here was very simple. If the centroid of the largest contour was away

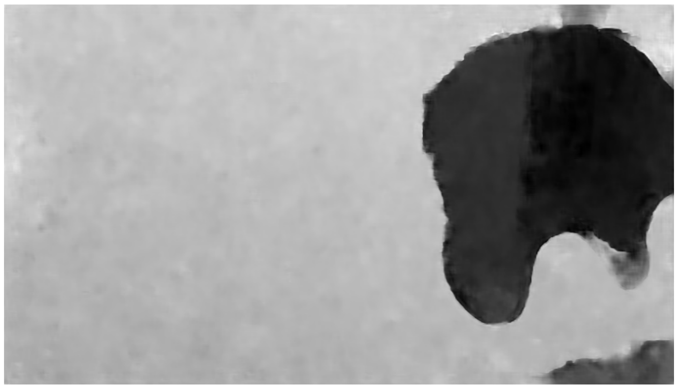

Fig. 10. Normalized Optical Flow Magnitude

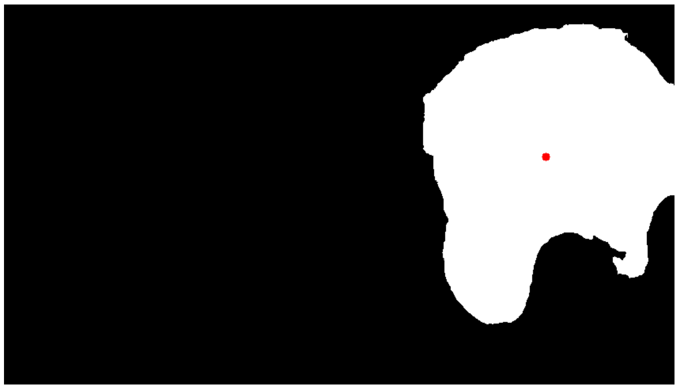

Fig. 11. Largest Contour with its Centre

from the image center, we commanded the drone to fly in that direction. For example, if the centroid is to the right of the image center, we command the drone to move right for a very small distance. This was repeated till the centroid was within a set range of the image center and helped us align the drone with the window. After centering, we fly through the window.

III. Phase 3

A. Environment

The third phase of our project presented the challenge of piloting a drone through a dynamic window, which featured a centrally rotating pink dial, akin to a clock hand. This dial, characterized by its constant rotation speed, served as a moving barrier that the drone had to adeptly navigate. A visual representation of this window and its rotating element is provided in Figure 12.

B. Angle Estimation of the Pink Dial

We address the challenge of estimating the angle of the rotating pink dial within a dynamic window, which is critical for drone navigation. Utilizing the OpenCV library, we implemented a color thresholding technique to isolate the pink dial. This process involves extracting the dial's contour and fitting a line to this shape. Intersection points between the line and the contour are determined, which are integral to computing the rotation vector.

This vector extends from the rotation center, established by solving the line equations at every tenth frame, to one of the intersection points. The angle of the dial is then deduced from this vector using the arctan function. The resultant angle is positive when the dial is in the upper half and negative when in the lower half. This angular information is pivotal for the subsequent navigation of the drone.

C. Drone Navigation Through the Dynamic Window

Following the angle estimation, the strategy to navigate the drone through the window is implemented. The drone is programmed to initiate flight through the lower half of the

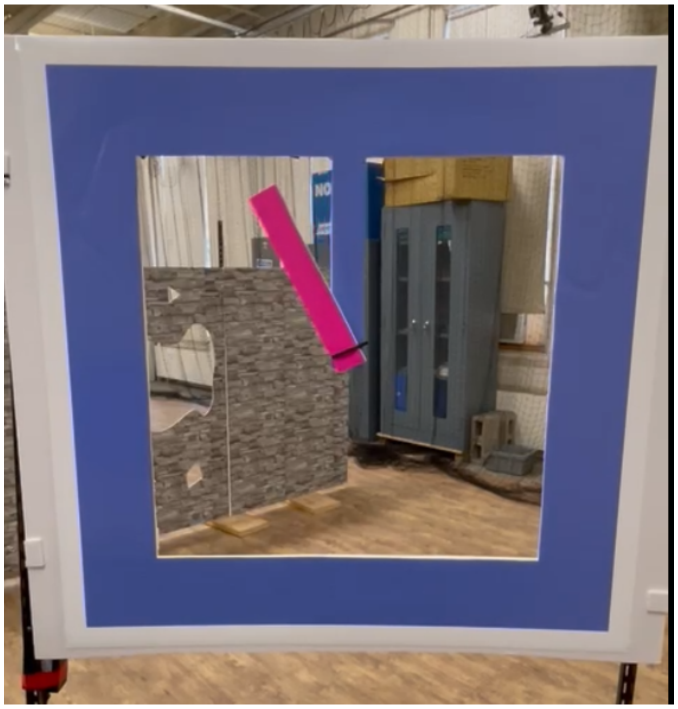

Fig. 12. Dynamic window

window as soon as the dial transitions from a negative to a positive angle. This implies that the dial has moved to the upper half, opening a passage for the drone. Fig. 13 shows the condition when drone is commanded to go through and Fig. 14 shows the condition when the drone cannot go through. Fig. 15 shows the flowchart of the full process the drone undergoes from phase 1 to phase 3.

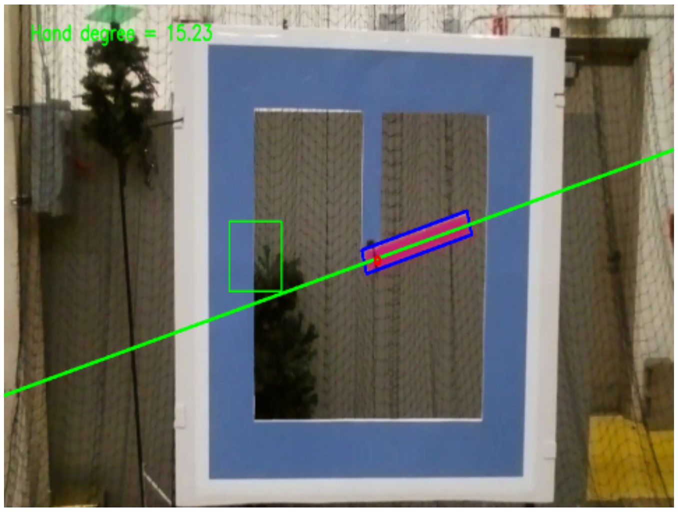

Fig. 13. Dynamic window - Go

We employ a synchronization mechanism between the angle detection and the drone's flight control system. By continu-

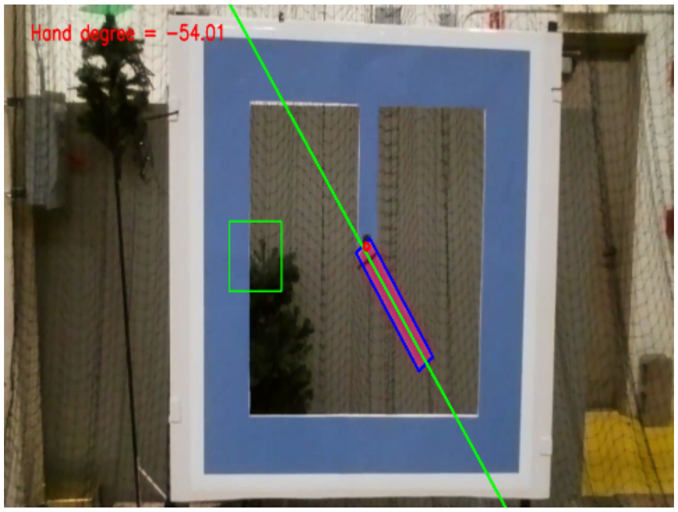

Fig. 14. Dynamic window - Do not go

ously monitoring the angle of the dial, the drone's onboard system can precisely time its flight to ensure a safe passage through the dynamic window. This method leverages the real-time processing capabilities of our system to achieve seamless and autonomous navigation.

IV. Results

The video recording of the run can be seen from the following link:

Video Submission Link

V. Conclusion

This project, segmented into three distinct phases, explores the development of an autonomous drone navigation system. In the first phase, we focus on navigating through multiple windows, drawing inspiration from the Alpha Pilot competition. We employ YOLOv8 for gate detection, trained exclusively on simulated data, coupled with Perspective-n-Point (PnP) for determining the gate's pose relative to the drone. This phase culminates in the successful integration of controls, planning, and hardware with the DJI TelloEDU drone.

The second phase addresses the challenge of flying through an unknown, arbitrarily shaped gap. Here, we utilize the SPyNet optical flow detection algorithm, achieving nearly 100

The final phase tackles navigating through a dynamic window with a central, rotating dial. A simple yet effective method of color thresholding and line fitting is employed to calculate the dial's angle, guiding the drone's timing to fly through the window.

Each phase of this project not only demonstrates technical proficiency but also contributes significantly to the field of autonomous drone navigation.

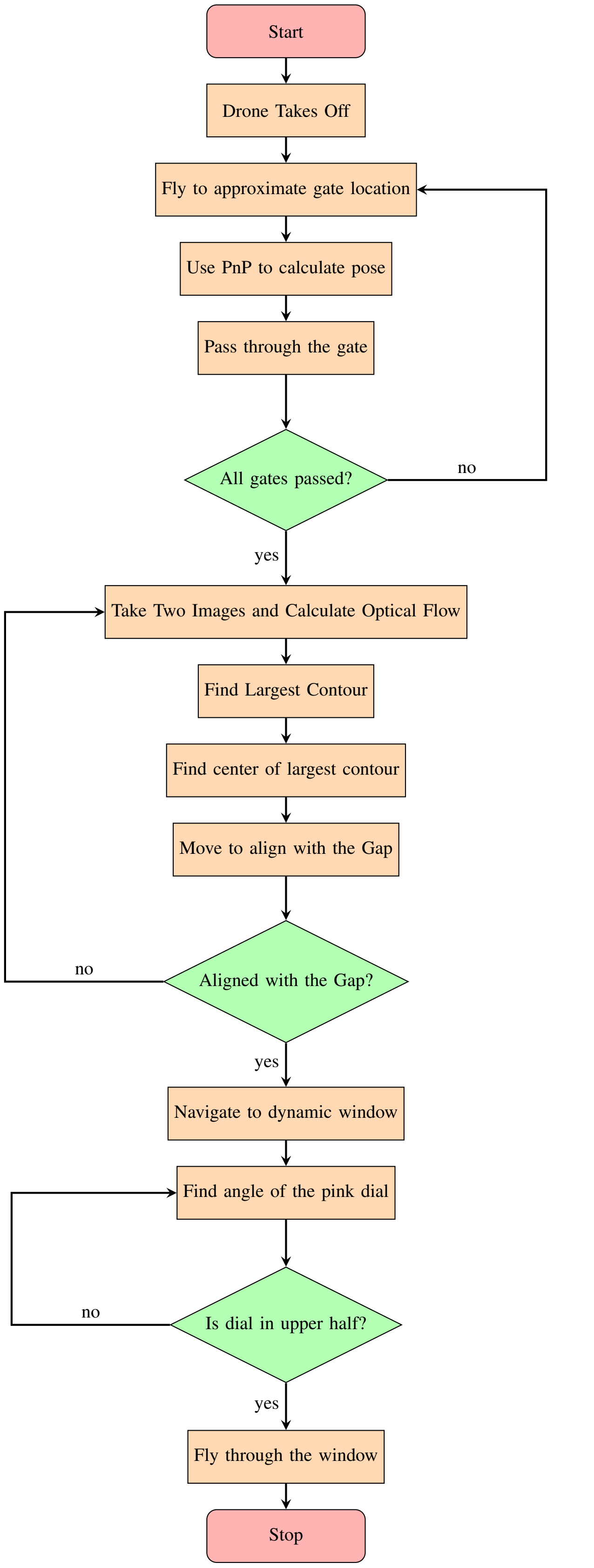
Start
Drone Takes Off
Fly to approximate gate location
Use PnP to calculate pose
Pass through the gate
All gates passed?
no
yes
Take Two Images and Calculate Optical Flow
Find Largest Contour
Find center of largest contour
Move to align with the Gap
Aligned with the Gap?
no
yes
Navigate to dynamic window
Find angle of the pink dial
Is dial in upper half?
no
yes
Fly through the window
Stop

Fig. 15. Flowchart of the drone navigation process through gates and dynamic windows.