P3: Mini Drone Race: (Using 2 late days)

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 aims to develop an autonomy stack to navigate through multiple windows inspired by the Alpha Pilot competition. The first phase of the project deals with the development of a robust Perception stack which has been trained using only simulation data and the second phase of this project deals with implementing the Controls, Planning, and hardware integration of our algorithm with the DJI TelloEDU drone.**

I. 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.

II. 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.

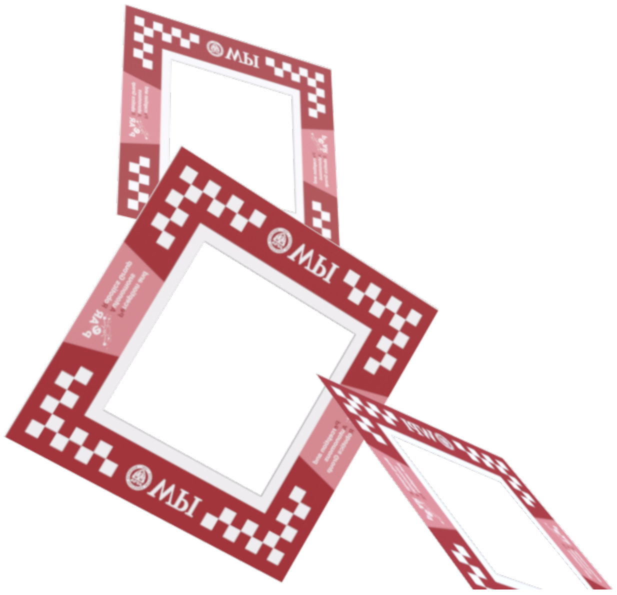


Fig. 1. Multiple gates used for dataset generation

A. 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 this problem, we used two approaches:

1) *3D Gaussian Splatting:* The first approach to solving the problem of intrinsic biases in popular literature is

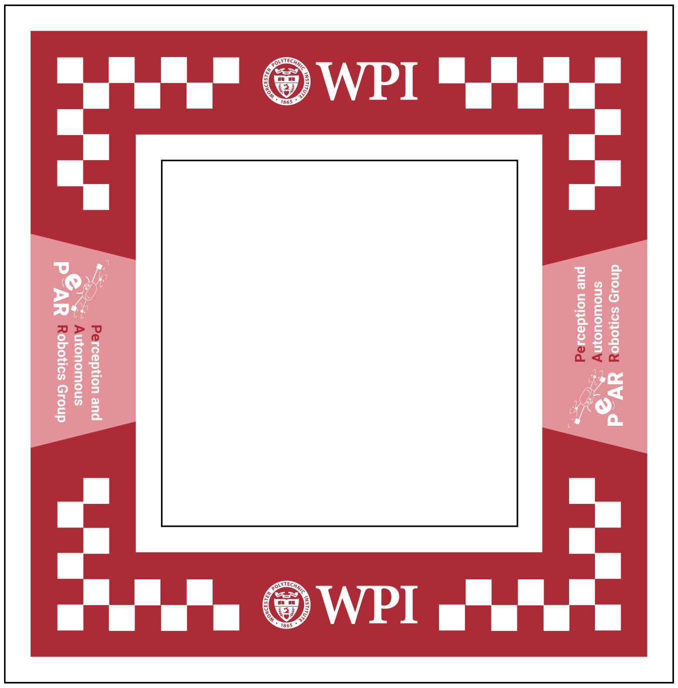


Fig. 2. Gate

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.

B. 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

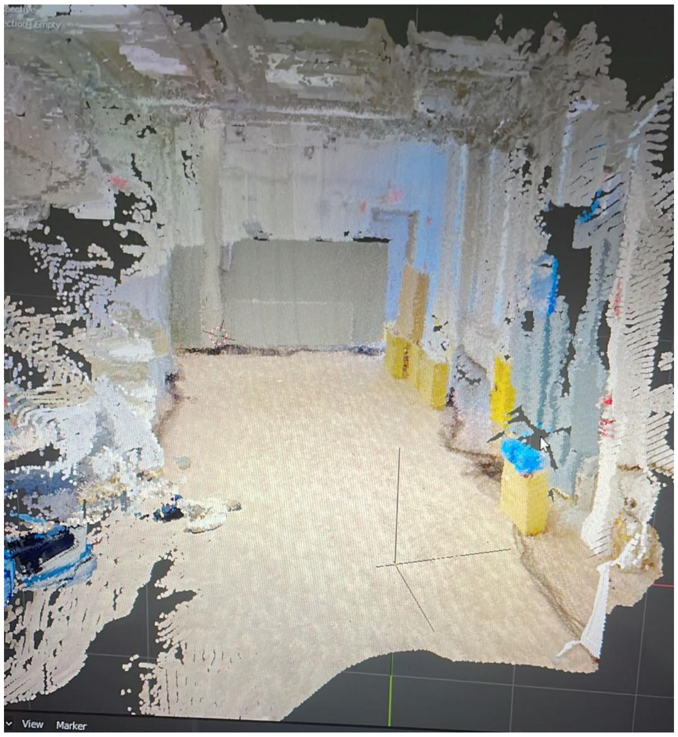


Fig. 3. GAUSSIAN SPLAT OF ENVIRONMENT

C. 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 window and the four corners. This helped us develop a robust

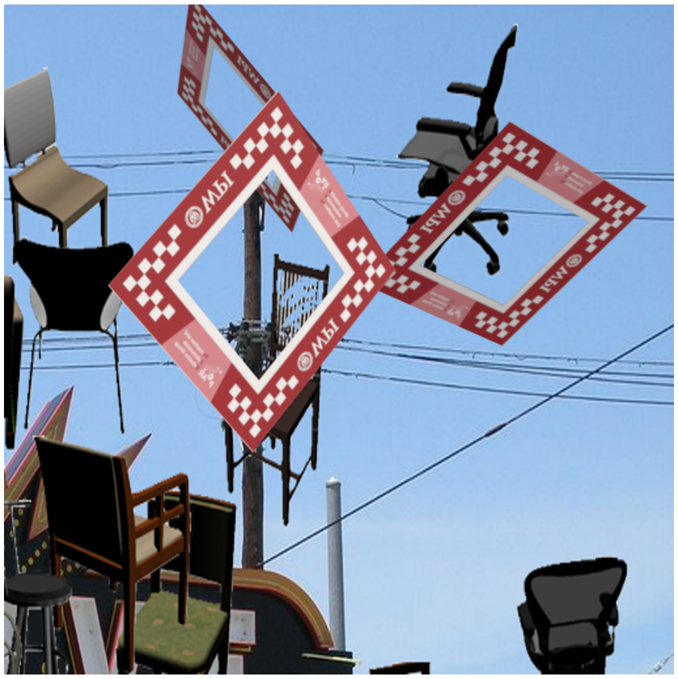


Fig. 4. Dataset with background image

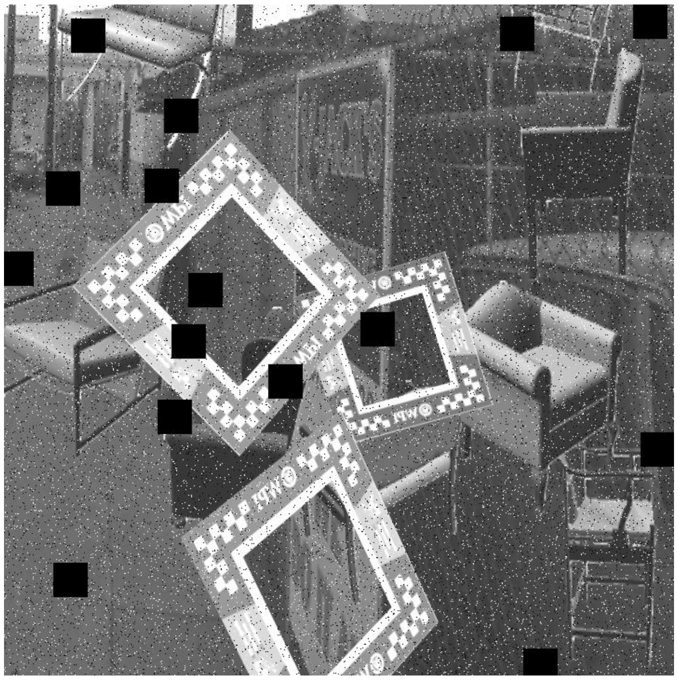


Fig. 5. Augmented training image

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.

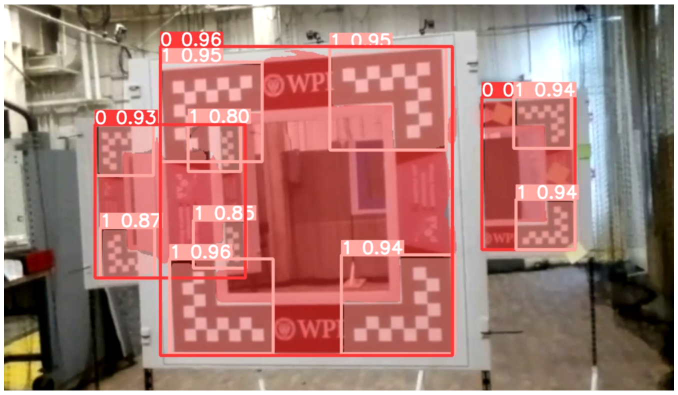


Fig. 6. Segmentation result of the network

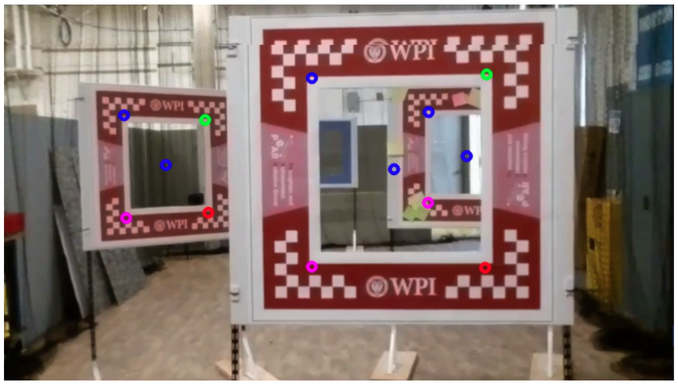


Fig. 7. Corners obtained after post-processing

D. 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.

E. Camera Calibration

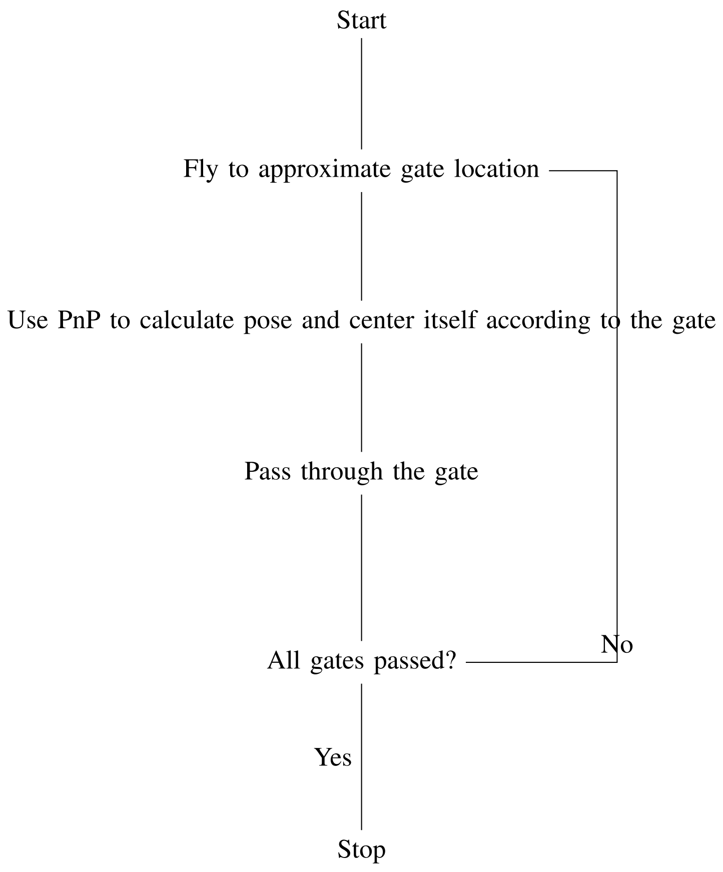
Camera Calibration was done with the help of MATLAB and a checkerboard pattern to correct the distortions in the images captured by the DJI Tello EDU drone and predict the camera intrinsics.

F. 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.

III. Deployment on DJI Tello

The perception stack developed in P3a is now deployed on the drone to make it pass through multiple windows. The basic heirarchy of our process is shown below:



Start
Fly to approximate gate location
Use PnP to calculate pose and center itself according to the gate
Pass through the gate
All gates passed?
No
Yes
Stop

IV. Result

Video Submission Link

V. Conclusion

The perception stack is really robust to noise in the input image and the corner calculations are really good. The drone is successful at crossing the gate. The only problem we faced as a team was to fly the drone accurately in every run. This is a problem we acknowledge and we will try to fix it in the next run.