P3A: Mini Drone Race - The Perception Saga!

Dushyant Patil
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
dpatil1@wpi.edu

Keshubh Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
kssharma@wpi.edu

***Abstract*—This project presents an approach to build a perception stack for DJI Tello drone for flying through objects of known shape and size. We use semantic segmentation to detect a window to pass through. We use DJI Tello's camera with Jetson Nano Orin for real time inference of window in field of view and estimates its position in Tello frame of reference. We have also listed our findings as a quick start guide for using pDJI Tello Edu drone for pose estimation of simplistic target objects.**

I. Problem Statement

The aim of this project is to estimate pose of a window of known shape and size. For this we will be using the semantic segmentation followed by a set of post-processing algorithm to get an estimate of window pose in drone frame. The estimated pose will later be used to fly the drone through the window.

II. World map

The data provided to us was the text format which consists of information about boundaries of 'world' and approximate pose of windows.

1) **boundary**: This parameter is defined as $\begin{bmatrix} x_{min} & y_{min} & z_{min} & x_{max} & y_{max} & z_{max} \end{bmatrix}$ where, $x_{min}, y_{min}, z_{min}$ defines the lower left point & $x_{max}, y_{max}, z_{max}$ defines the upper right point of the rectangular environment.
2) **window**: This parameter is defined as:
$$\begin{bmatrix} x & y & z & x_{delta} & y_{delta} & z_{delta} & \\ qw & qx & qy & qz & X_{delta} & Y_{delta} & Z_{delta} \end{bmatrix}$$
where, x y z represents the approximate center of the window in meters. qw qx qy qz represents the approximate orientation of the window as a quaternion. While the remaining parameters are variances in the pose in a Gaussian distribution.

We load this data in blender as cube objects using `primitive_cube_add` function of bpy module. The Washburn laboratory has exactly same setup which replicated the map file given.

III. Data Generation

A. Synthetic Dataset Creation

To train the segmentation mask, we created a synthetic dataset using blender rendering capabilities. We moved the position of camera randomly in a set of parallelopiped such that we could cover different possible camera/ drone configurations. In blender, we applied the constraint on the camera so that is is centered towards a target point (A point close to the window center of one of the 3 windows). We also varied the position of light in 6 configurations to impose different lighting conditions. We followed the steps given by Professor Nitin about setting the blender environment so that we get different segmentation masks using different pass indices and different compositing. By combining all above mentioned steps, we generated about 2500 images with segmentation masks. The image below shows a basic example of single window mask creation with different occlusions.

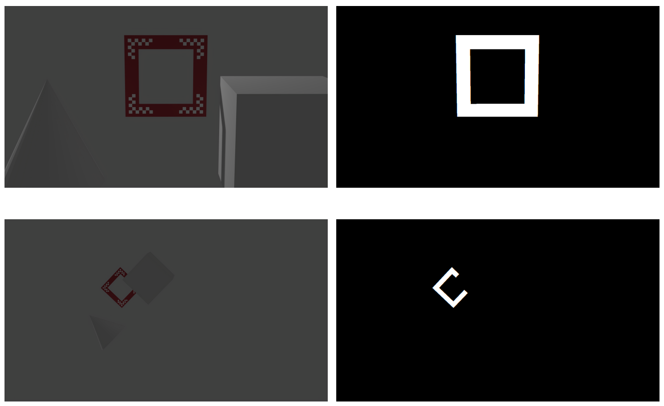
Fig. 1: RGB Images with segmentation masks

B. Sim2Real Transfer

To account for simulation to real life transfer for UNet predictions, we added the washburn laboratory background to out blender environment in material properties of different background cubes. Fig 2 shows the example of such renderings.

C. Use of Homography on Single Windows

To expedite the speed of dataset generation, we applied known homography to the corners of the window and applied similar homography transformation to its mask. We also embedded this transformed window mask pair with different background images. All above steps amount to around 5K images of resolution 720x960.

IV. Semantic/ Instance Segmentation using DL

Deep learning is effective for segmentation due to its ability to automatically learn intricate patterns and features

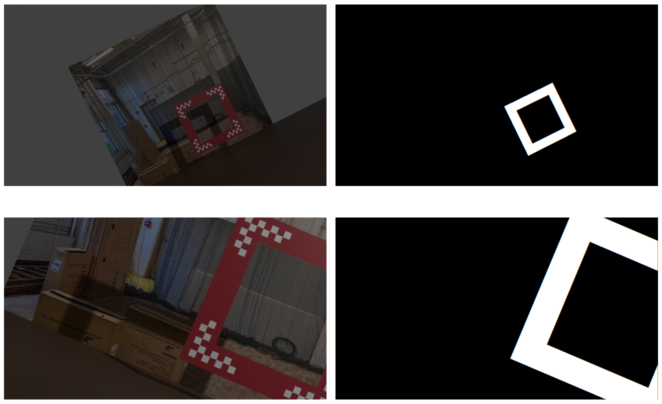
Fig. 2: RGB Images with segmentation masks and Washburn lab background

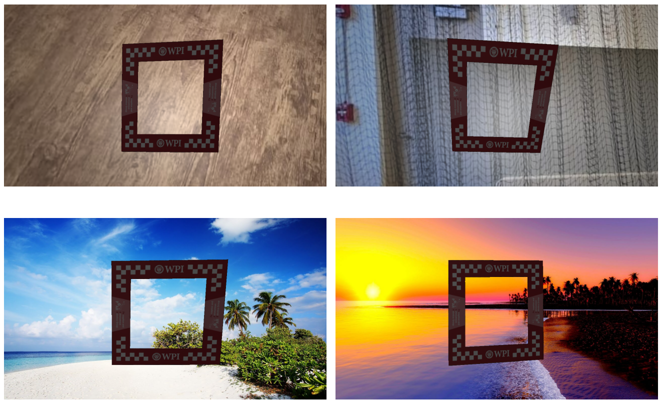

Fig. 3: RGB Images with applied homography and different backgrounds

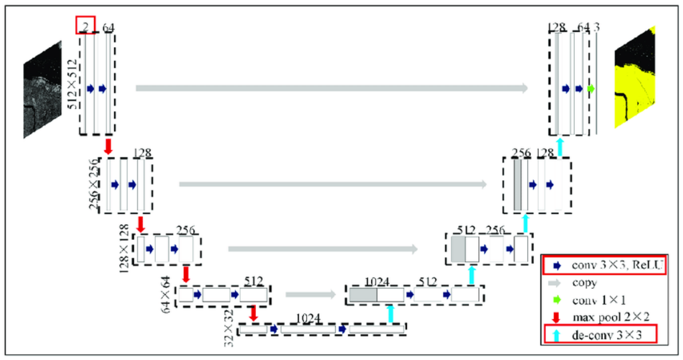

Fig. 4: Unet Architecture

from data. Convolutional Neural Networks (CNNs) excel at capturing spatial dependencies in images, making them ideal for tasks like object or image segmentation, where precise localization and intricate detail extraction are crucial.
For our project we decided to use Unet which is a deep learning architecture commonly used for image segmentation tasks, particularly in medical image analysis and computer vision.The network consists of an encoder and a decoder. The encoder captures features from the input image through convolutional layers, reducing spatial dimensions. The decoder then upscales and refines these features to generate pixel-wise segmentation masks. Notably, UNet employs skip connections, which connect corresponding layers in the encoder and decoder. This helps in preserving fine details during the upsampling process, enhancing segmentation accuracy.

For our implementation, as shown in Fig. 4, we take the input image as 512x512 with 3 channels (Red, Green, Blue) and pass it through a double convolution layer that keeps the same dimension but increases the channels to 64. We then perform max pooling to reduce the size to half. This combination of double convolution and max pooling is repeated 4 more times which leads to encoded tensor of size 32x32 with 1024 channels. After this we de-convolute this tensor which increases the size to 64x64 and reduces the channels to 512. We then copy the previous encoded tensor of matching size and concatenate it to the de-convoluted tensor and then pass it through a double convolution. We repeat this de-convolution, concatenation & double convolution 4 more times to regain the original dimension of 512x512 with 64 channels. We then pass this tensor through a fully connected layer to reduce the number of channels to 1 to get the segmentation mask.// The described model was trained on the previously described simulated dataset for 900 epochs with a learning rate of 0.0003 to achieve desired robustness and reliability. This trained model is then used to perform inference using the Jetson Orin Nano on the images captured by the DJI Tello Edu Quad-rotor.// Fig. 5 shows the captured frame from DJI Tello and Fig. 6 shows the segmentation mask on the captured image. We can observe that our implementation gives proper segmentation mask even if multiple windows are present.

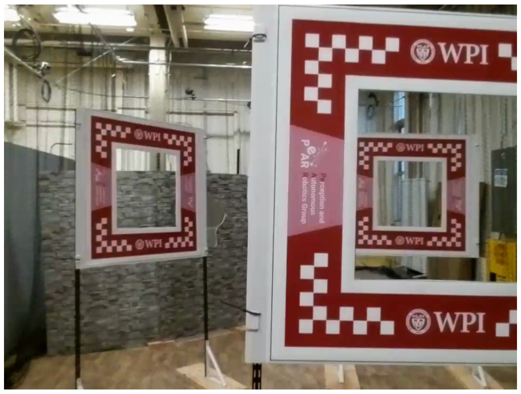

Fig. 5: Captured frame from DJI Tello

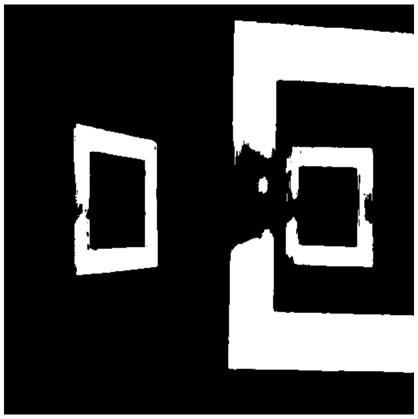

Fig. 6: Inference from trained Unet model

V. Corner Detection

Using the UNet inferred masks, we estimate closest window and its corners. Using these corners, we estimate the pose of the window with respect to the Tello camera.

A. Closest window estimation

We used largest area criteria to estimate closest window among the predicted masks. We used contour detection with the help of opencv function cv2.findContours. We sorted these contours in the ascending order of area occupied by contours in pixels. We use the last contour as our closest window. As these contours are not perfect rectangles or squares, we apply further post processing to estimate corners of windows. We tried two different approaches which gave us certain accuracy at certain estimation speed. Below image show the inference and largest area detection using this method:

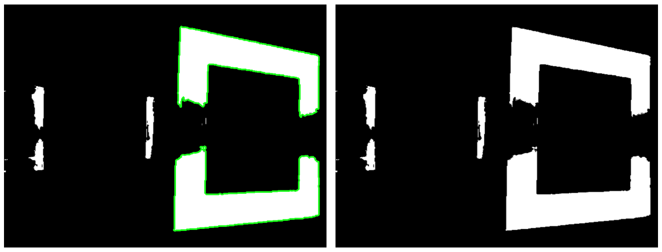

Fig. 7: RGB Images with applied homography and different backgrounds

Fig. 8: Inference with closest window estimation

B. Delaunay Triangulation Based Estimation

To approximate the exact corners of the closest window, we apply erosion-dilation on the predicte mask followed by Canny edge detector. On the Canny edges, we apply probabilistic Hough transforms to get set of lines. We find intersection of all lines making an angle of 45 degrees or more with each other. Through all the intersection points, we apply Delaunay triangulation. On these Delaunay triangles, we find edges which are part of only one triangle. These edges represent outer edges of the estimated window shape. Using the points on these outer edges, we approximately fit a quadrilateral and the corners of these corners give us the corners of window which are pretty accurate. But this method takes a lot of time (around 1 s per frame). Thus we tried an alternative which would give similar accuracy in less time. Below images 9 show this algorithm which shows all the intermediate result of image processing:

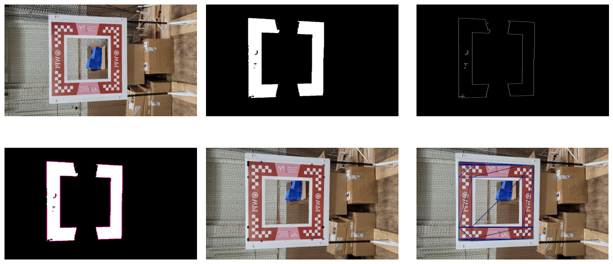

Fig. 9: Delaunay Triangulation pipeline

C. Rectangle Fitting

On the closest estimated window, we apply dilation-erosion. On this image, we approximate a convex hull. On this convex hull, we find the maximum and minimum x and y coordinates. Using these coordinates, we fir a rectangle covering enclosing the predicted mask. On the edges of the rectangle we try to find the points which are closest to the vertices but lie withing the white region of our predicted mask. This gives a good estimation of corners of the window. This method gives us real time corner detection with good accuracy. Below images **??** show the rectangle fitting pipeline to get bounding box of closest window:

VI. Camera Calibration

To perform pose or depth estimation, we need to determine the instrinsic parameters of the camera of the Tello drone. We use a checkerboard of size 10x7 where each square was of size 23 mm to perform the calibration. As the image quality changes significantly depending on whether you are recording a video or taking still photographs. As we need to perform inference on frames captured during the flight, we performed calibration on images recorded during a video recording. We selected 22 frames from the video such that the image sharpness was good enough and the camera angles varied a lot. We used the Matlab CameraCalibrate tool to get intrinsic parameters of the camera. We used a self implemented code

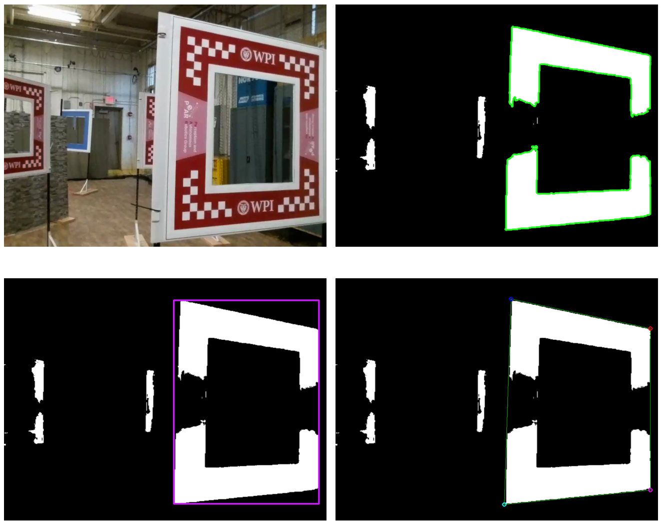

Fig. 10: Rectangle Fitting Corner Estimation

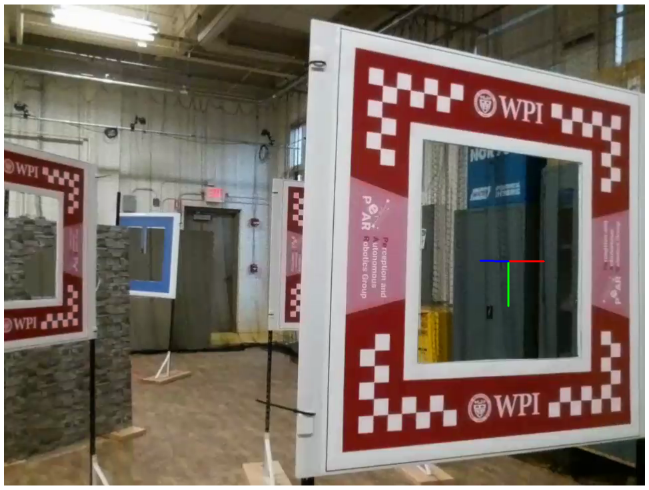

Fig. 11: "Reprojecting the Window center in drone camera frame

of Zhang's calibration (From CV Fall22 course) to check if the matlab camera matrix and our code's camera matrix gave similar parameters. We found that to be the case and our camera matrix was as follows:

$$K = \begin{bmatrix} 911.4 & 0 & 467.4 \\ 0 & 911.2 & 357.3 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients = [0.0011, -0.031, 0, 0, 0]. We use these parameters to estimate window pose in camera frame.

VII. Pose Estimation

We use Perspective N-Points method to estimate the pose of the window. We use cv2.solvepnp function to estimate the pose of closest window in camera frame. We reproject the coordinate frame on the image as shown in image 11 below:

References

[1] A Quaternion-based Unscented Kalman Filter for Orientation Tracking

[2] Class Notes by Prof. Nitin Sanket

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015. Available: https://arxiv.org/pdf/1505.04597.pdf

[4] "Unet from Scratch —— Unet Tutorial —— Developers hutt," www.youtube.com.

[5] "Implementing original U-Net from scratch using PyTorch," www.youtube.com.

[6] "PyTorch Image Segmentation Tutorial with U-NET: everything from scratch baby," www.youtube.com.