P5: The Final Race

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 navigate through a race track consisting of a window of known size and shape, followed by a window of unknown shape and size followed by a dynamic window in the end. We use perception and navigation stacks developed in previous 2 projects (Segmentation- PnP for known shape and optical flow for unknown shape window) for first 2 stages. For dynamic window we use segmentation to detect fixed frame and dynamic part of the frame and decide a set of navigation commands to pass through this dynamic windows. We use DJI Tello's camera with Jetson Nano Orin for real time inference of windows in field of view and estimate its relative center in Tello frame. We use threading to fly, record and infer to execute parallel perception and navigation.**

I. Problem Statement

The aim of this project is to fly through a race track using on-board sensing and Jetson Orin compute. The racetrack consists of different types of obstacles as shown in 1. We use a combination of classical and deep learning based computer vision for perception and use DJI Tello API for navigation.

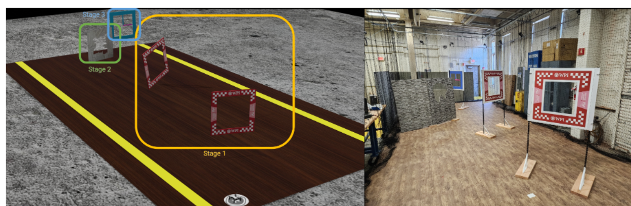


Fig. 1: Simulated and Real Race Track

II. Stage1 : Passing through Knwon Shaped Window

We follow the Project 3 perception and navigation stack to pass throug hfirst 2 stages which consists of 2 windows of known shape ad size. We use UNet model for semantic segmentation to detect the windows and estimate their depth. Deep learning is effective for segmentation due to its ability to automatically learn intricate patterns and features 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.

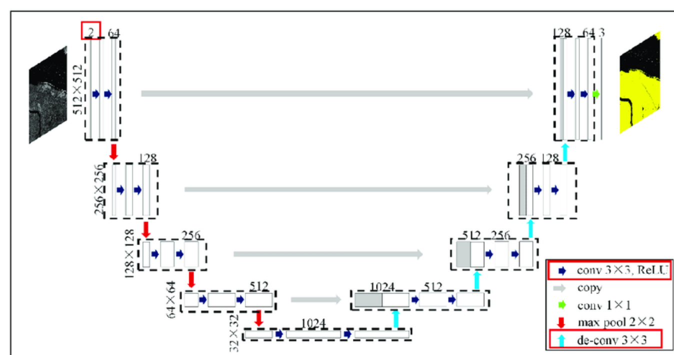


Fig. 2: Unet Architecture

For our implementation, as shown in Fig. 2, 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. 9 shows the captured frame from DJI Tello and Fig. 10 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. 3: Captured frame from DJI Tello

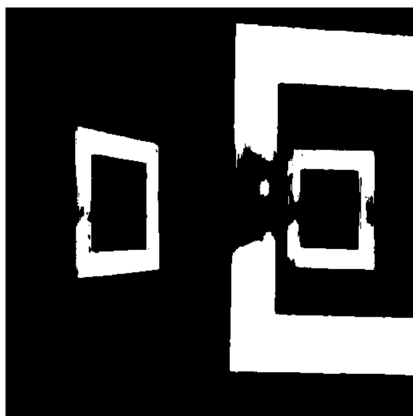


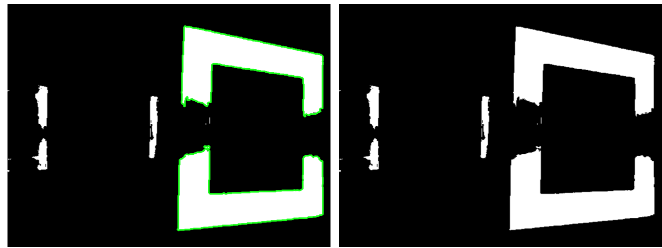
Fig. 4: Inference from trained Unet model

III. 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:



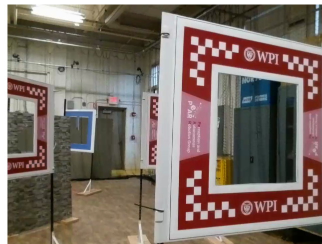


Fig. 5: RGB Images with applied homography and different backgrounds

Fig. 6: Inference with closest window estimation

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 15 show the rectangle fitting pipeline to get bounding box of closest window: .

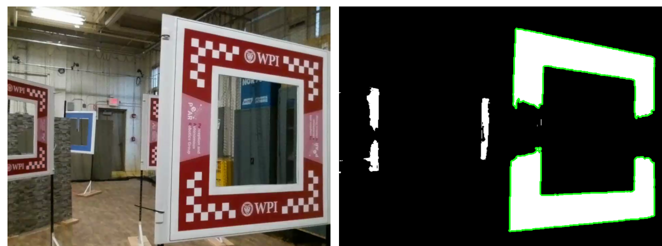


Fig. 7: Rectangle Fitting Corner Estimation

On this detected window, we apply perspective N-point (PnP) to determine the pose of the windows with respect to the camera. Using the PnP distances, we first perform horizontal and vertical adjustments to the drone. Following this, we send the drone forward through the window. The submitted RunVideo.mp4 showcases a successfull attempt using this algorithm.

IV. Stage2: Optical Flow using DL

Optical flow is a computer vision technique that quantifies the motion of objects in a sequence of images or frames. It involves tracking the apparent movement of pixels between consecutive frames, providing a dense vector field representing the velocity of each pixel. Optical flow is crucial for tasks like object tracking, motion analysis, and video understanding. By calculating the displacement of pixels over time, it enables machines to perceive and comprehend dynamic visual scenes, finding applications in robotics, autonomous vehicles, and video processing.

Deep learning methods enhance optical flow by automatically learning intricate patterns and representations from data, improving accuracy and robustness. Traditional methods often struggle with complex scenarios, while deep learning models, such as convolutional neural networks, excel at capturing nuanced motion patterns. They adapt well to diverse scenes, making them more effective in real-world applications, such as object tracking and autonomous navigation, where the ability to handle varying motion complexities is crucial.

For this project we decided to use LiteFlowNet which is a compact and efficient optical flow estimation model designed for real-time applications, particularly on resource-constrained devices. Optical flow refers to the apparent motion of objects between consecutive frames in a sequence of images or video frames. It is a crucial computer vision task with applications in video analysis, object tracking, and motion understanding.

LiteFlowNet is derived from FlowNet, a deep learning architecture for optical flow estimation, but it is optimized for lightweight and real-time performance. The ”Lite” in LiteFlowNet signifies its focus on reducing computational complexity while maintaining competitive accuracy.

The architecture of LiteFlowNet consists of multiple lightweight convolutional layers that capture spatial dependencies in the input frames. It employs a correlation layer to efficiently compute feature correspondences between frames, enabling it to estimate pixel-level motion information. The use of densely connected layers helps capture intricate motion patterns in the input data.

One notable feature of LiteFlowNet is its pyramid processing, where the model considers multiple scales of information. This multi-scale approach is beneficial for handling different motion scales present in diverse scenes.

LiteFlowNet's design prioritizes computational efficiency, making it suitable for deployment on devices with limited computational resources, such as mobile phones or embedded systems. The model strikes a balance between accuracy and speed, making it well-suited for real-time applications like robotics, augmented reality, and video processing on edge devices.

In summary, LiteFlowNet is a lightweight optical flow estimation model designed for real-time applications, offering a balance between accuracy and computational efficiency. Its architecture incorporates features like dense connections and pyramid processing to capture complex motion patterns across multiple scales in input frames. This makes LiteFlowNet particularly useful for resource-constrained devices where real-time optical flow estimation is essential.

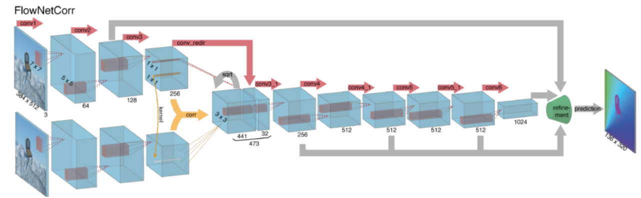


Fig. 8: FlowNet Architecture

In our application, we executed a sweeping motion using the DJI Tello drone while capturing images. This sweeping motion is employed to comprehensively map the entire frontal area of the drone, ensuring that we capture the widest possible perspective. The sequence of images captured during this motion is processed using the pre-trained LiteFlowNet. The temporal sequence of images is processed in pairs, starting from the beginning of the recording, to derive optical flow information between each pair.

To facilitate optical flow inference, the image pairs are resized to 1024x436, aligning with the dimensions the model was initially trained on. The model then subjects the stacked images to a series of convolution layers, enabling multiplicative patch comparisons between the two feature maps. The resulting feature maps are concatenated at the output, facilitating the computation of optical flow. Subsequently, the output undergoes a series of upconvolution operations to restore the feature maps to the original image size, completing the optical flow estimation process.

Fig. 9 shows the captured frames from DJI Tello and Fig. 10 shows the segmentation mask on the captured image. We can observe that our implementation gives great optical flow description.

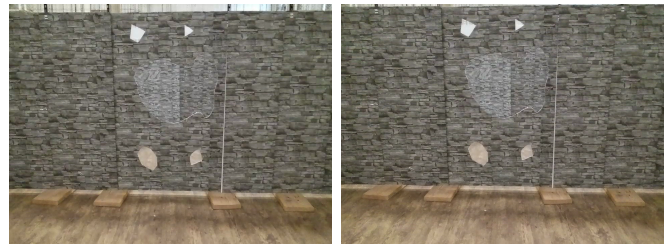


Fig. 9: Input images to Liteflownet

Fig. 10: Inference from pre-trained LiteFlowNet model

V. Gap center prediction

Using the the LiteFlowNet inferred masks, we estimate largest window. Using this window's contours, we estimate the center of the window with respect to the Tello camera. We also assume that the gap is big enough and not oriented much so that the drone does not need to perform yaw movements. We use binary inverted threshold to segment the foreground and background. We tried with a few values of threshold and came to an observation that the threshold of 100 gives good results when the drone moves at 20 cm/s. If we change the speed of the drone for visual servoing, we will need to tune the threshold again. The image 11 shows the thresholding output on the flow visualization image 10.

Fig. 11: Thresholding for Binary Inverted mask

We then try to estimate the largest window in the foreground to decide which window to pass through.

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.

B. Rectangle Fitting

On the estimated largest window, we apply dilation-erosion. On this image, we approximate a convex hull to distringuish between an objects / open space on the side of the foreground from the hole in the foreground (gap). On this convex hull, we find the maximum and minimum x and y coordinates. Using these coordinates, we fit a rectangle covering enclosing the predicted mask. This gives a good estimation of center of the window and its bounding box. This method gives us center detection with good accuracy. Below images 15 show the rectangle fitting pipeline to get bounding box of closest window:

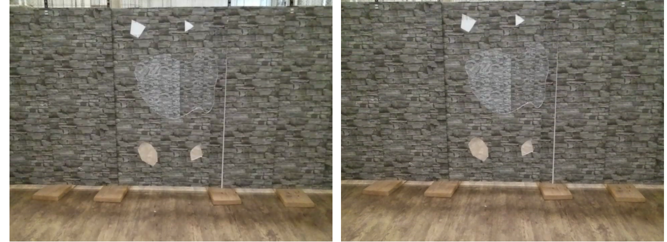


Fig. 12: Input images to Liteflownet

Fig. 13: Optical Flow

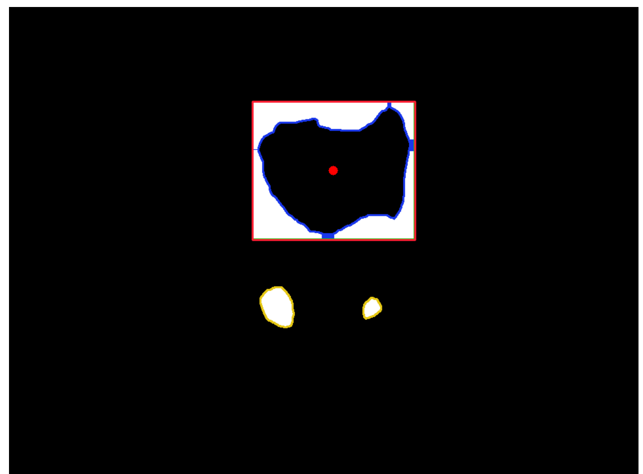


Fig. 14: Center Estimation and bounding box

VI. Visual Servoing and Navigation

We use a delayed visual servoing as the liteflownet takes some time (around 0.5 seconds per pair of images). Using the world map details as explained in the World Map section above, we initially move the drone in X and Y directions (approximately) parallel to the foreground. We parallely record images at a regular time interval and estimate their approximate pose in 3D for each image. Once we complete the initial movement, we then run the inference on the images recorded and estimate the center of largest gap as explained in above sections. From the center pixel coordinates, we estimate if the gap center and drone center have aligned. For all the images where drone center and gap center had aligned, we find the mean of 3D poses associated with all the images. Then we give the drone a command to go to that pose using position control. After this we send the drone forward. The image below shows one image where drone center and gap center are aligned.

Fig. 15: Center Estimation and bounding box

VII. Stage3: Flying through dynamic window

The third stage consisted of a dynamic window similar to a clock with the blue frame and pink hand as shown in fig116. As the window shown below has very distinct color schemes, we decided to apply color segmentation to detect fixed frame and rotating part of this dynamic window.

As the colors of the fixed frame and rotating hand have a very distinct color which was also different from the background color, we decided to use color segmentation for detecting the pose of the window. We used a simple thresholding in HSV colorspace. We were able to get good masks for the fixed and dynamic frame as shown in image 17 18.

To find the angle the hand makes with the frame, we estimate the rotating hand center by using the frame center and use a bounding box to find endpoints of the rotating hand. To tune the HSV threshold values, we used a trackbar using opencv GUI for trackbar as shown in image19

The navigation through the third window consists of 2 stages:

1) Approach Phase - In the approach phase we reach a certain distance i.e. 160 cm from the window 's approximate center position. We try to determine the angle made by the frame with global Coordinate system using PnP on the frame mask. Using the angle, we try to find the next waypoint at 160 cm from the window and navigate the drone there. Once the drone reaches that point, we rotate it at a given angle determined previously.
2) Fly-Through Phase - Following this position we constantly track the angle of the hand with the horizontal. We also find the distance the window is at (which should be close to 160 cm) using PnP on the frame mask. Once the hand reaches in between the angle bounds of $110^o - 135^o$, we give a move forward command using the distance found using PnP. We found out that if we

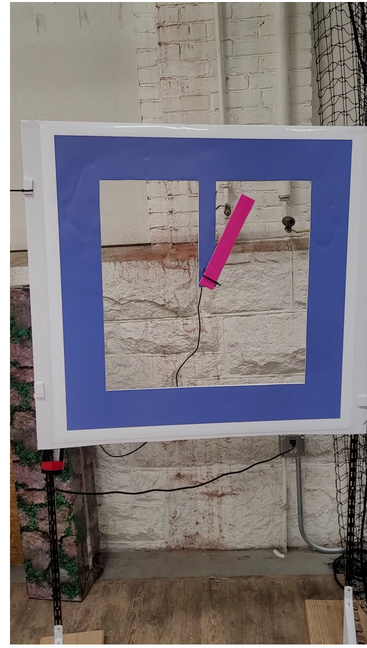


Fig. 16: Dynamic Window

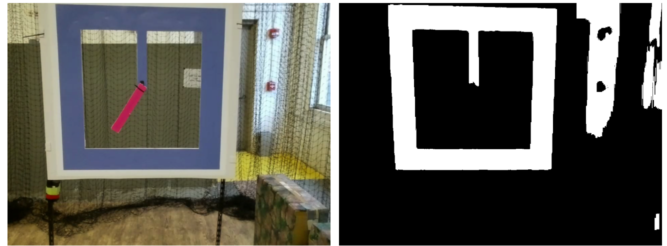


Fig. 17: Input images to Frame Mask

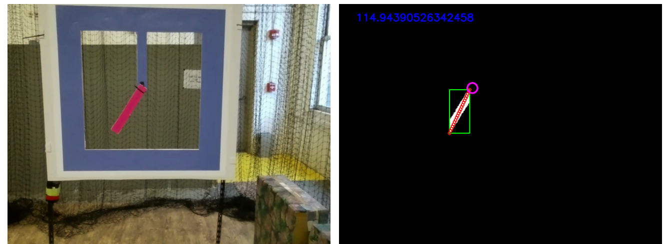


Fig. 18: Input images to Rotating Hand Mask

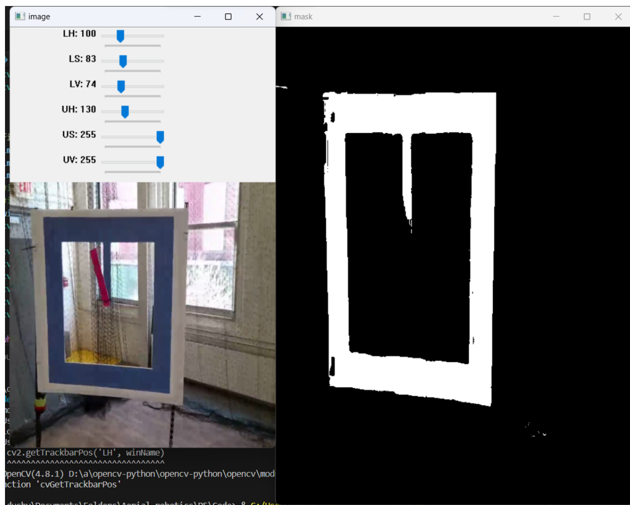


Fig. 19: Color Threshold Tuning using Trackbar

give a high speed so that the drone could cross this distance within 6 seconds we can cross the window withing one motion command. The results can be seen in the submitted video run file.

VIII. Problems Faced

- The comparatively slow inference time of LiteFlowNet made real time inference on image pairs unikely. We had to execute a recording routine and the infer them.
- The slow inference time also presented the challenge of Tello auto landing after 20 seconds. To overcome this we created a standby co-routine in which the drone moves up and down until the inference is complete.
- Another problem we faced was our drone wasn't properly working with motion commands. It would either randomly skip the commands altogether or execute an inaccurate action. After changing the drone for a new one and using the Jetson's PCIe network card limited this problem to an extent. We suspect that simultaneously recording frames and giving motion commands on parallel threads puts strain on the UDP connection and it confuses the commands. Further analysis is necessary for confirming this and how to mitigate this issue.

References

[1] A Quaternion-based Unscented Kalman Filter for Orientation Tracking
[2] Class Notes by Prof. Nitin Sanket
[3] T.-W. Hui, X. Tang, and C. Loy, "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation."
[4] S. Sniklaus, "Sniklaus/Pytorch-liteflownet: A reimplementation of LiteFlowNet in pytorch that matches the official Caffe version,"