P4: Navigating The Unknown!

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 unknown shape and size. We use optical flow 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 estimate its relative center in Tello frame. We use threading to fly, record and infer in parallel which causes some issues for UDP communication which we have mentioned about at the end of the report.**

I. Problem Statement

The aim of this project is to estimate pose of a window of unknown shape and size. For this we will be using the optical flow related deep learning models 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 approximate area of the location of the unknown window. 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. We use 4 different textures and window shapes provided in the starter code file of the P4 assignment to select the network of choice, develop the perception and navigation codebase, and test our approach in simulated environment (blender) before testing the code in the real world (Washburn laboratory).

III. 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.

In our application, we executed a sweeping motion using

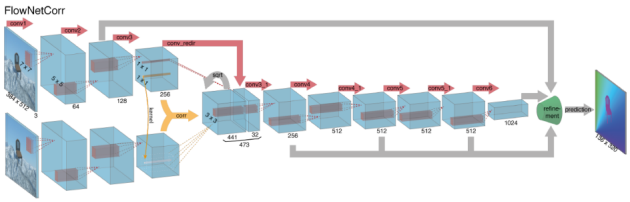

Fig. 1: FlowNet Architecture

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. 2 shows the captured frames from DJI Tello and Fig. 3 shows the segmentation mask on the captured image. We can observe that our implementation gives great optical flow description.

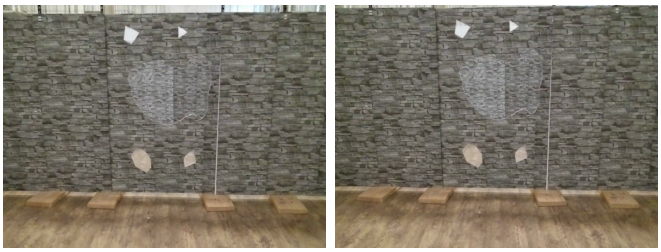

Fig. 2: Input images to Liteflownet

IV. 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 4 shows the thresholding output on the flow visualization image 3.

Fig. 3: Inference from pre-trained LiteFlowNet model

Fig. 4: 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 8 show the rectangle fitting pipeline to get bounding box of closest window:

V. 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

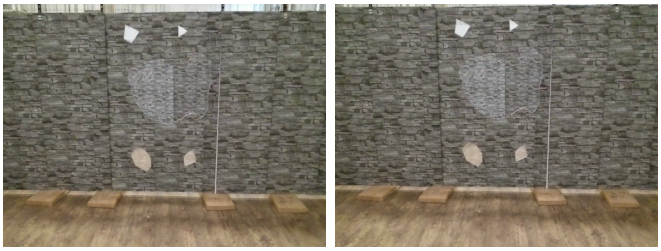

Fig. 5: Input images to Liteflownet

Fig. 6: Optical Flow

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

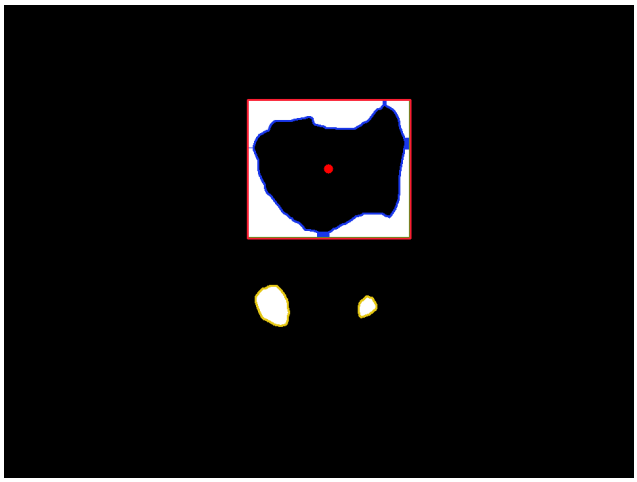

Fig. 7: Center Estimation and bounding box

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. 8: Center Estimation and bounding box

VI. 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,"