P4:Navigating The Unknown!

Shakthibala Sivagami Balamurugan
Robotics Engineering
Worcester Polytechnic Institute
Email: sbalamurugan@wpi.edu

Aditya Patwardhan
Robotics Engineering
Worcester Polytechnic Institute
Email: apatwardhan@wpi.edu

Harsh Shah
Robotics Engineering
Worcester Polytechnic Institute
Email: hshah2@wpi.edu

***Abstract*—The aim of the project is to demonstrate a vision-based approach for autonomous drone navigation through obstacles of a unknown shape and size. We developed a perception and navigation stack for autonomous navigation in Vizflyt using optical flow and drone movement. We used RAFT for optical flow estimation and classical computer vision techniques for gap detection of unknown shapes, location and size. We use this for visual servoing and navigation of the drone through the gap.**

I. Introduction

The capability of navigating a complex environments is challenge for autonomous drones. We aim to enable the drone to autonomously identify and navigate through gaps of unknown shape, size and location. The process is initiated with the drone flying to position where the gap is visible in its frame. We detect the gap using optical flow from Recurrent All-Pairs Field Transforms (RAFT) network. We maneuver the drone horizontally and vertically after a gap has been detected. This movement cause the foreground and background (gap) to have different magnitudes of optical flow. We use this to get the regions of gap in the textured wall. In the next phase we process this flow and use canny edge detection to get the gaps outlines. We then find the largest contour and calculate its centroid using moments. We now have the center of the unknown gap, we need to center the drone such that its image center is aligned with the center of the gap. By using Visual Servoing we are able to accomplish centering the drone with the center of the gap. We then go through the gap, achieving the goal of autonomous navigation through unknown gaps.

Fig. 1. FPV view of drone in vizflyt

II. Implementation

Detecting the gap requires optical flow which requires movement. We explore different methods for optical flow

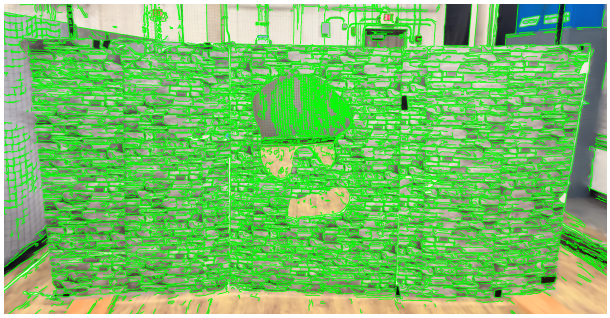

Fig. 2. Canny edge detection for preliminary window finding

A. Optical Flow using Farneback method

We first implemented Farneback method for optical flow using opencv library. This method was used to differentiate foreground and background using thresholding on the flow output. While this method was fast it did not provide optical flow. Even with post-processing with magnitude filter, canny edge detection and contouring we were not able to get good results for gap centroid.

B. Window Finder

Our drone camera field of vision is wide due to which the textured wall is a small part of the image at start. This results in a lot of the background having similar properties as the gap in the textured wall. This led to improper detection which was difficult to solve through post processing. We tried to segment the textured window using classical cv methods like canny and contouring. Due to the texture of the wall this led to subpar window boundary detections and the brick texture overloaded the contouring methods. We also realized that not having a consistent crop would break the optical flow, so we did not use this approach.

C. Optical Flow using Dense Inverse Search

While implementing the Dense Inverse Search (DIS) Algorithm to compute optical flow, which is also a function of the opencv library, the results were not that promising. Even though the model was generating better results than Farneback, these results were not yet near usable for gap centroid processing.

D. Optical Flow using RAFT

The Recurrent All-Pairs Field Transforms (RAFT) Algorithm for optical flow is known to achieve high accuracy with method essentially combining the correlations with a recurring refining mechanism. While testing the available images with the pretrained raft models, crisp images were generated that detected the window gap and the centroid with great precision. While testing the models on different pretrained RAFT models, we were able to determine that different raft models were actually better for detection in images and in videos. The model trained on the *Things3D* dataset was detecting better for still images, while the model trained on the *Sintel* dataset was detecting the gap window with a very good accuracy.

III. Post-Processing

Following the optical flow estimation, we implemented a magnitude-based segmentation pipeline to isolate the target gap. We operated on the premise that during forward flight, the gap (representing the distant background) exhibits significantly lower optical flow magnitude compared to the window boundaries (foreground).

First, we applied an inverse binary threshold to the flow magnitude matrix. By setting a magnitude threshold of $\tau_{mag} = 2.0$, we generated a binary mask where pixels exhibiting flow below this value were set to high (255), effectively highlighting the static or distant regions representing the gap.

To ensure spatial coherence and reduce noise artifacts, we employed a two-stage morphological filtering process. A morphological *close* operation was first applied to fuse fragmented regions and fill small holes within the detected gap area. This was immediately followed by a morphological *open* operation to eliminate small, stochastic noise speckles from the surrounding regions, resulting in a clean binary segmentation of the low-flow areas.

From this processed mask, we extracted contours to delineate the boundaries of potential gaps. We filtered these candidates by area, selecting the largest contour as the primary target. Finally, we computed the first-order image moments of this largest contour to calculate its centroid (C_x, C_y). This coordinate pair serves as the visual servoing setpoint, allowing the quadrotor to align itself with the center of the gap for safe traversal.

IV. Visual Servoing

Our visual servoing strategy is a position-based approach that iteratively minimizes the offset between the detected gap center and the camera's optical center. This process utilizes a Proportional (P) control mechanism to generate translational movements that align the drone with the target in the lateral (Y) and vertical (Z) world axes.

A. Error Calculation and Coordinate Mapping

The first step is to quantify the error between the detected gap centroid $(\mathrm{c}_x, \mathrm{c}_y)$ and the camera's image center $(\mathrm{img\_c}_x, \mathrm{img\_c}_y)$. The differences are calculated in pixel space as d_x (horizontal error) and d_y (vertical error).

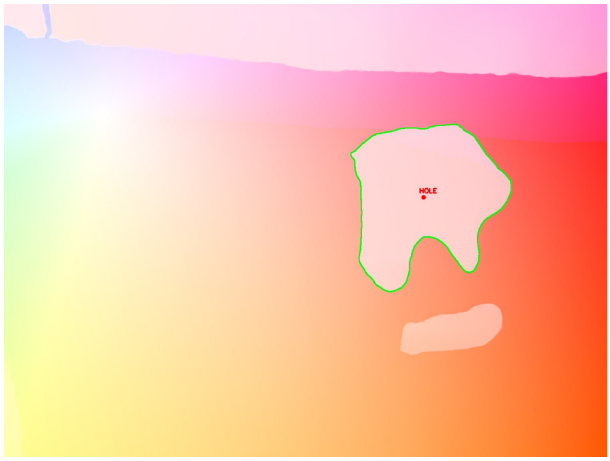

Fig. 3. RAFT with gap centroid

A critical step is the mapping of the image-space error to the world coordinate system, where the drone's lateral motion is controlled by the Y-axis and its vertical motion by the Z-axis.

- The horizontal pixel error (d_x) is directly mapped to the required **lateral (Y-axis)** movement (d_{y_world}).
- The vertical pixel error (d_y) is directly mapped to the required **vertical (Z-axis)** movement (d_{z_world}).

B. Proportional Control Command Generation

To translate the pixel error into a stable and proportional control command, we apply tuned scaling factors ($\mathrm{lateral\_scale}_y$ and $\mathrm{lateral\_scale}_z$). These constants define the gain of the proportional controller, determining the magnitude of the world adjustment relative to the observed pixel error. A negative sign is introduced to ensure that a positive pixel error (target to the right or below) results in a corrective movement in the negative world direction (left or down).

$$\mathrm{d}_{y\_\mathrm{world}} = -\mathrm{d}_x \times \mathrm{lateral\_scale}_y$$
$$\mathrm{d}_{z\_\mathrm{world}} = -\mathrm{d}_y \times \mathrm{lateral\_scale}_z$$

C. Target Waypoint Update and Execution

The calculated world offsets are used to define a new intermediate target position (Center_targetPose) relative to the drone's current world position (currentPose). Specifically, the d_{y_world} is used to update the Y-coordinate (lateral alignment). This new target waypoint is then executed using a fixed velocity of $0.31\,\mathrm{m/s}$. This iterative movement ensures that the drone continuously shifts its world position until the gap centroid is centered in the image frame, at which point the errors (d_x and d_y) approach zero.

V. Testing

RAFT was tested on images which were rendered from Blender. The Fig. 4 to Fig. 11 are the Blender simulation images along with their detection and centroid. Their respective IOU values are mentioned in the results.

VI. Assumptions

- It is assumed that Quadrotor is a point object.
- We have decoupled the camera orientation from the drone's body so that the Splat view always stays upright by sending only the yaw (heading) angle to the renderer while keeping roll and pitch fixed at zero. This prevents the video from flipping when the drone starts inverted, without affecting its dynamics or control behavior.

VII. Results

IOU values for testing

- texture 1: 99.36
- texture 2: 99.83
- texture 3: 99.85
- texture 4: 99.87

VIII. Conclusion

With the algorithm mentioned in section 4, We were able to pass through the unknown gap with the drone and the perception stack for drone with monocular camera is written successfully.

IX. Supplementary Material

The videos of the Assignment 4 are given along with both detections overlay and without overlay: https://bit.ly/3K6bIcm

References

.

[1] Farnebäck, G. (2003). Two-Frame Motion Estimation Based on Polynomial Expansion. In: Bigun, J., Gustavsson, T. (eds) Image Analysis. SCIA 2003. Lecture Notes in Computer Science, vol 2749. Springer, Berlin, Heidelberg.

[2] Kroeger, T., Timofte, R., Dai, D. and Van Gool, L., 2016, September. Fast optical flow using dense inverse search. In European conference on computer vision (pp. 471-488). Cham: Springer International Publishing.

[3] Teed, Zachary, and Jia Deng. "Raft: Recurrent all-pairs field transforms for optical flow." Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16. Springer International Publishing, 2020.

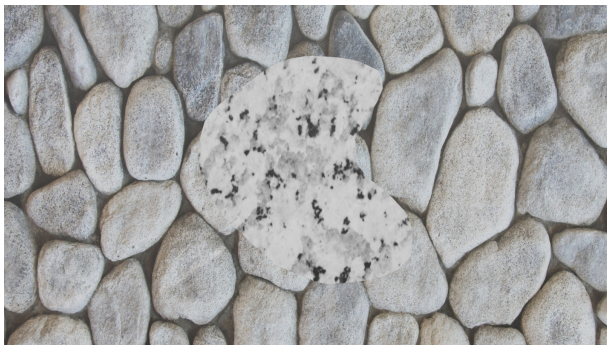

Fig. 4. Blender Texture 1

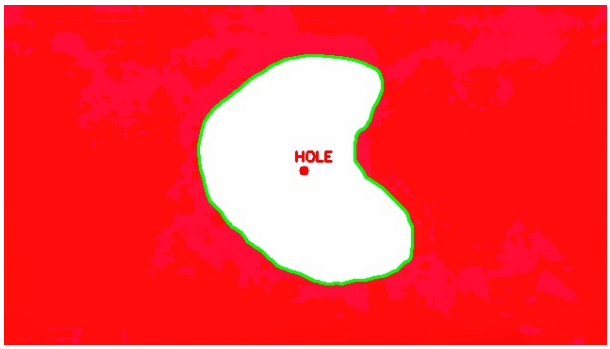

Fig. 5. RAFT for Texture 1

Fig. 6. Blender Texture 2

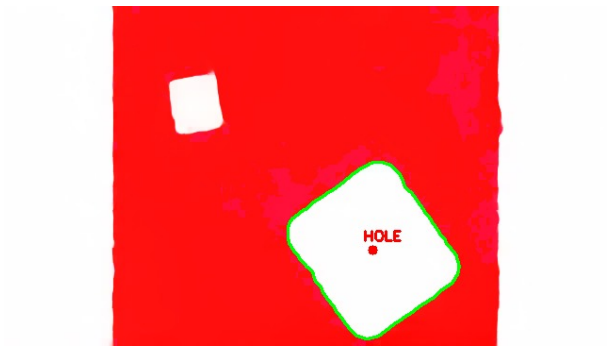

Fig. 7. RAFT for Texture 2

Fig. 8. Blender Texture 3

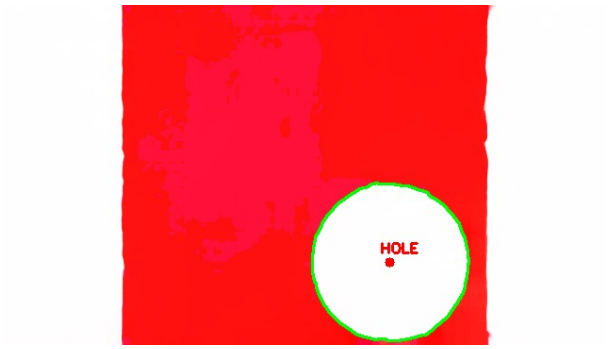

Fig. 9. RAFT for Texture 3

Fig. 10. Blender Texture 4

Fig. 11. RAFT for Texture 4