Project 4b: Avoid the wall and find the bridge

Team 6: Noob Quaternions

using 2 late days

Prateek Arora
Masters of Engineering in Robotics
University of Maryland, College Park
Email: pratique@terpmail.umd.edu

Abhinav Modi
Masters of Engineering in Robotics
University of Maryland, College Park
Email: abhi1625@umd.edu

I. Introduction

In this project, we have two tasks:

1) Find the bridge placed on the river and cross the bridge while avoiding the river
2) Detect the wall in front of your quadrotor and go through above or below the wall, depending on the height of the wall

II. Data Collection

1) Find the bridge: Since we used front camera for this task, the images from front camera were collected by hovering the quadrotor near one end of the river, at the height of 0.8 meters from the ground and 1.5 meters away from the river and then moving the quadrotor manually all the way towards the other end of the river.
2) Wall detection: For this task, the images from the front camera were recorded by moving the quadrotor in the plane parallel to the wall. The quadrotor was moved manually in a rectangular trajectory (in the plane parallel to the wall) starting at height of 1.0 meter from ground. The data was collected for different height of the wall.

III. Implementation

A. Avoiding the wall

The most important aspect in this part was to detect the wall and find the height at which it was placed. A simple answer to this problem is feature based segmentation but it was not so simple in this case. The features detected on the wall were so similar to the features in the background that no feature algorithm proved good enough to help detect the wall with the required degree of robustness. This led us to try the following different techniques which take into account the 3-dimensional nature of the scene to aid in the segmentation process.

*1) **Monocular Structure from Motion**:* The first thing we tried was using sequence of images to estimate the relative depth in the scene and generate a sparse depth map. The problem with this algorithm was its high dependence on feature matching in the two frames.

*2) **FastDepth: Monocular Depth Estimation using CNNs**:* Then we tried a deep learning approach to estimate depth of the scene for a monocular system. Most of the deep learning methods for complex tasks like depth estimation from image data are computationally very expensive to be run on an embedded system like the Intel Upboard. But recently MIT came up with a method called **FastDepth**[1] where they came up with a novel network architecture using depth-wise convolutions and hardware specific compilation of the code to reduce the latency in both the encoder and decoder segments of the network.
We deployed one of their pretrained model(on NYU Depth v2 dataset) but the scene at hand was geometrically very different from the ones in the dataset that was used for training. The output of the network for one test image is shown in fig(1)

Figure 1: Depth map generated using fast-depth approach using a pretrained network for the image taken from the monocular camera

Due to lack of time we could not generate enough data so as to train this network for our purpose. But this was a good exercise to help us learn about such approaches and maybe if we can generate enough data, such an approach can be used for future projects requiring such complex on-board computations.

*3) **Optical Flow Estimation using a Spatial Pyramid Network**:* Then we tried a data agnostic deep learning based approach. Estimating depth from optical flow using consecutive image frames. The only assumption was that rotational motion is absent and the drone moves only translation directions X,Y and Z which is quite reasonable for small angle approximations at low speeds.

The following dense optical flow image (fig(2)) was generated by the network given two consecutive frames taken from the onboard monocular camera as input.

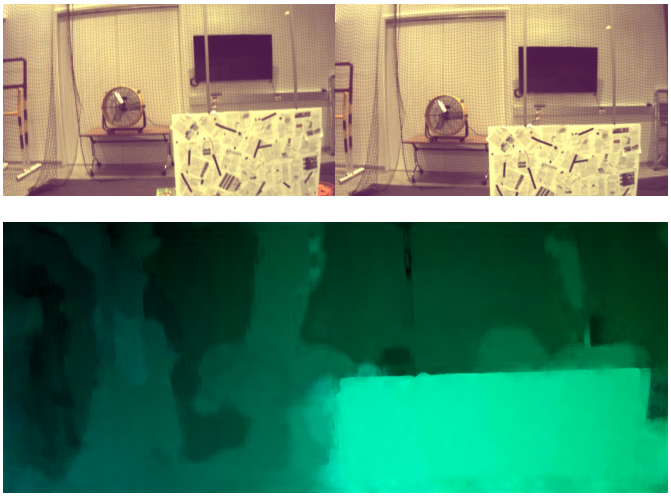

Figure 2: Flow estimation using SPN using the top two images as input

The flow estimation was very accurate and sharp giving perfect segmentation of the wall. But high accuracy comes with a high computation cost. As the project requires on-board communication of bebop and the micro-processor for autonomous flight, there was only so much processing power available for the neural net to run onboard. The inference time achieved for the network while running all the other required processes in parallel was approximately 43s per frame. Pruning and other network compression techniques could not be tried but due to lack of time.

4) Dense optical flow - Farneback: Finally, we ended up using the traditional dense flow estimation technique "Gunnar Farnebäck" approach of *Two-Frame Motion Estimation Based on Polynomial Expansion*. The flow estimation was not as accurate as compared to SPY-Net but it makes up for the loss in accuracy by its low latency and real-time execution. This was a major point for consideration. The dense flow estimated by the algorithm is shown in figure 3:

Using the Farnebäck approach we estimate flow and then segment the wall using simple k-means clustering where k=4

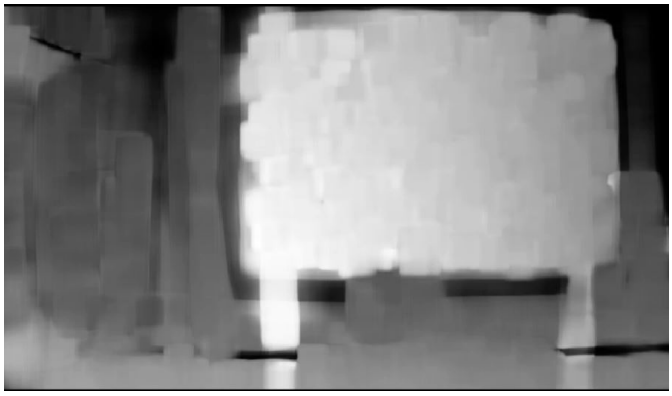

Figure 3: Flow estimation using Gunnar Farnebäck method

bins. To remove random clusters in the background a simple thresholding on median flow values was also performed. Once the segmentation is complete the following pipeline is followed for determining the location of the wall in 3D and then navigating across it:

1) The segmentation gives a binary mask of the wall in the image.
2) Simple Blob detection from OpenCV is used to find the center of this blob.
3) Once in hover after takeoff, the altitude is increased from 1 to 1.5m in intervals of 0.05m allowing motion in vertical direction to provide multiple views for flow calculation. Meanwhile, the drone aligns the center of the detected blob with the center-line of the image.
4) As the center of the detected blob aligns with the image center, the vertical error is calculated using

$$v_{error} = \frac{h_{image}}{2} - y_{(blob\_center)} \tag{1}$$

5) If the v_{error} is negative it means blob center lies in the lower half of the image indicating that the wall center is present below the horizon of the camera i.e., below the drone. Thus, it is safe to fly forward at a height of 2-2.25m, i.e., above the wall.
6) Vice-versa, v_{error} is positive it means it is safe to fly at a height of 0.8-1m i.e., below the wall.

Note: A threshold was empirically determined to handle cases where the wall center is just below or just above the image center-line. The threshold was chosen to be around 20 pixels to assure save passage avoiding near-miss scenarios. Links to recordings of the demo run and trajectory followed by the drone are present in the section V.

B. Find the bridge

Finding the bridge was a relatively easier task. For this task, we assume the the quadrotor takes off at the left end of the river and searches for the bridge by moving along the river until the other end of river. The process to bridge detection is elaborated in further sections:

1) Color thresholding:: For the current setup and environment the river was easily segmented using color thresholding because of the contrast blue color offered from the rest of the scene (as seen in figure 4). Owing to the fact that we could choose lighting of the scene, color thresholding was an apt choice as it is faster than Single Gaussian Model and Gaussian Mixture Model. After thresholding the image, the ROI was extracted and resized to the actual dimension of the image for further processing (as seen in figure 5).

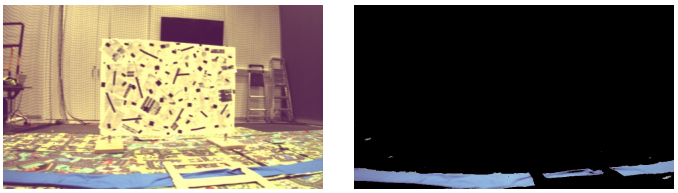

Figure 4: From left to right (a) Input image (b) Result of color thresholding

Figure 5: Output of ROI extraction and resizing

2) Image Processing and filtering:: The resulting image was converted to binary mask. The resultant mask contained a big blob corresponding to the river and noise corresponding to the small patches of blue color because of imperfect segmentation. A median filter with a kernel of 7x7 was used to remove the noise followed by a closing morphology operation. This gave a decent, noise-free segmentation of river as seen in figure 6(b).

3) Blob detection:: In order to detect the bridge, a simple approach of detecting discontinuity in the river mask was followed. To do so we detect blobs in the image with a constraints on size and inertia of the blob. The aim was to detect mainly two categories of blob i.e (1) the part of river (blue) visible in between the bridge (referred to as bridge-blob) and (2) the part of river on either side of the bridge (referred to as river-blob). These constraints were set to differentiate between the two categories of blob. The algorithm is designed to find the bridge-blob in the the first frame (figure 7(a)) and the align the image center to the bridge-blob center. In case multiple bridge-blobs are detected the quadrotor is commanded to move right. However, if the bridge-blob is not detected in the first frame, river-blob was detected. Owing to the constraints the we have chosen there can either be a single river-blob or two river-blobs in the image. If a single blob was detected (as seen in figure 7(b)) the quadrotor moves to the right along the river. However, if two river-blobs are detected we remove the two river-blobs in the image followed by detection of bridge-blob (as seen in figure 8). After the detection of river-blob, we again try to align the image center to river center.
Once the image center is aligned to the bridge-blob center we go forward until the blob is out of image frame.

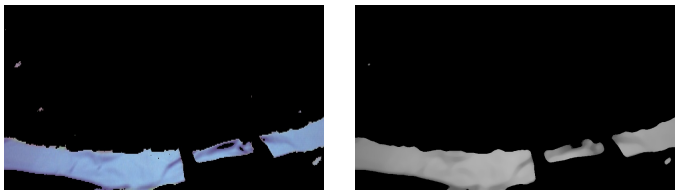

Figure 6: From left to right (a) Mask after thresholding (b) Noise reduction after applying median blur

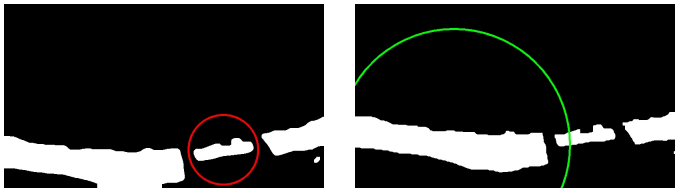

Figure 7: From left to right (a) Detection of Bridge-blob in the first frame (b) Detection of river-blob when bridge-blob is not detected in the first frame

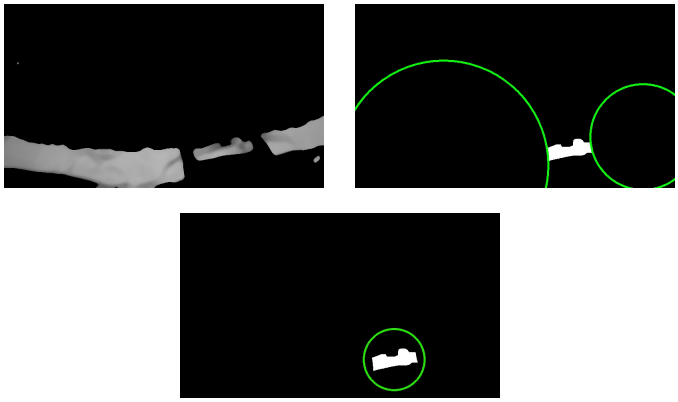

Figure 8: From left to right (a) Image with filtered mask (b) River-blobs on either sides of bride detected and removed (c) Bridge-blob detection after removing river blobs

IV. Discussion and Conclusion

A. Optical Flow Estimation: wall Detection

As explained in section III, multiple methods were tested for segmenting the wall from the background. The traditional methods provide reasonable results but deep learning approaches give far more accurate results. There is still a long way to go before deep learning approaches can be used on embedded systems but the future looks bright with the advent

of approaches like *fast-depth* showing promise in the field of neural net inference on computationally constrained systems. Flow based methods to estimate static scene geometry require the camera to atleast move from its initial position to another location to get an estimate of the flow, which was solved by initially increasing altitude from 1 to 1.5m.

B. River Detection

For our test case we have used color thresholding to separate river. Simple color thresholding in the river detection can be replaced by Gaussian Mixture Models for robust color segmentation of river.

V. Outputs and Results

Cross the bridge: The odometry plot begin with quadrotor mid air with the red axis pointing towards the river. The river is parallel to the plane formed with red and green axis.
Here are the links:

- Rviz plot task1.mp4
- Link to live demo

Avoid the wall: The odometry plot begin with quadrotor taking off from the origin and the window is placed to the left of the quadrotor (in front of green axis as seen from the first 2 seconds of the video)
Here are the links:

- Rviz plot task2.mp4
- Link to live demo

VI. References

1) FastDepth: Fast Monocular Depth Estimation on Embedded Systems(link)
2) Optical Flow Estimation using a Spatial Pyramid Network(link)
3) Two-Frame Motion Estimation Based on Polynomial Expansion(link)