Optical Flow For Vision-Aided Navigation

Elizabeth Boroson
Stanford University
lboroson@stanford.edu

1. Introduction

A current subject of interest in navigation is the use of vision as an aiding source. Most unmanned aerial vehicles (UAVs) have a camera onboard, so a navigation system with vision could use these existing sensors instead of requiring that new sensors be added, which would be an advantage when the space available for sensors is limited. Some vision-aided navigation algorithms that have been developed recently find displacement between images by matching SIFT or SURF features. However, this type of algorithm is usually computation-intensive, and most UAVs are small and have very strict weight and power limitations for the processors that can be used. An alternative would be to use optical flow to find displacement, which tends to have lower requirements for the processor. Unfortunately, the cameras available on UAVs usually have relatively low frame rates, and there is enough displacement between images that typical optical flow algorithms are not effective. For my project, I want to find and optimize an algorithm that can apply optical flow techniques to the type of data encountered in navigation.

2. Problem Statement

Several algorithms have been proposed to improve the performance of optical flow techniques over large displacements. For my project, I plan to implement several of these techniques in MATLAB and test their performance on images similar to those that would be encountered in navigation. I will then select one of these algorithms and further refine it by taking advantage of properties specific to navigation data. By refining the algorithm, I hope to be able to improve either its speed or its accuracy.

I will compare the algorithms to each other in both accuracy and computational performance. Additionally, I will implement a simple SIFT-matching algorithm for comparison with each of these new algorithms. I expect the displacement calculated by my optical flow algorithm to match that calculated by matching SIFT features, and I expect the computational performance to be better. I plan to compare the algorithms' accuracy in MATLAB. I will compare the performance by looking at the computational complexity of the algorithms. If I have time, I'd also like to implement both the SIFT-matching algorithm and the optical flow algorithm I develop in C to do a more accurate comparison.

I expect to be able to optimize the algorithm I select by using specific assumptions that are true for navigation data. In navigation, we can assume that all motion in the image is due to the motion of the camera. This would not apply to all situations (for instance, navigation in an urban environment), but in situations where weight and power are limited, like on a UAV, we are likely to be far away from anything else that is moving. However, we may encounter objects at different distances in the image, so I will not be able to assume that optical flow is continuous over the entire image.

I have collected four datasets, each consisting of several images from Google Earth over a specific region. Each set of images is recorded from an altitude that a UAV might fly at, and there is slight motion between each image. Three of the image sets are flat aerial images, taken from between 500 and 1000 meters. One set was recorded over Stanford, one over MIT, and one over a fairly flat and uninhabited region slightly northwest of Los Angeles. This region contains a dirt road, some trees, and some variation in terrain, but no man-made biuldings. These regions will provide an interesting comparison, since I have generally found that SIFT features can be detected and matched much more accurately on manmade sturctures than on natural structures and terrain. It will be interesting to see if this (or the opposite) is true of optical flow. Three consecutive images from the Stanford dataset are shown in Figure 1 as an example of the images I'll be using.

The fourth dataset was collected over downtown Los Angeles, from an altitude of about 800 meters. Unlike the others, these image contain 3-dimensional models of buildings that can be displayed in Google Earth. The roofs of these buildings are significantly higher than the surrounding streets, so these images are an example of navigation data in which different parts of the images are at different depths and have different displacement between images. I am very interested to see how the algorithms perform with these non-flat images.

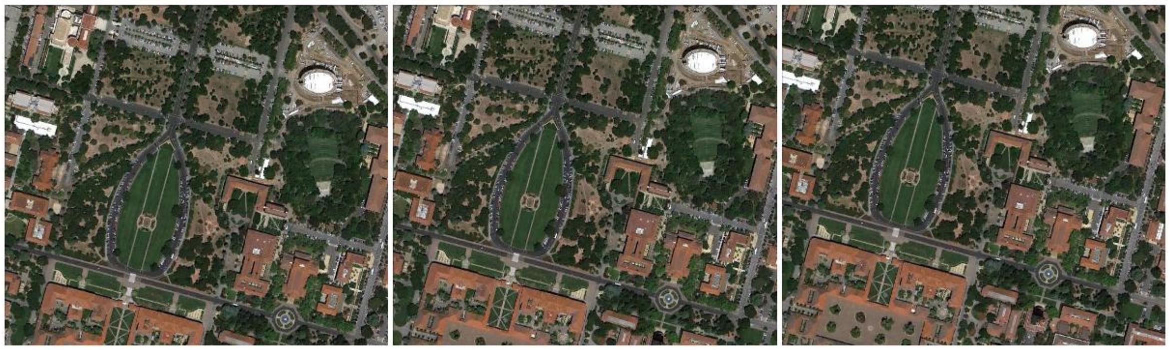

Figure 1. Three consecutive images from the Stanford dataset.

3. Technical Approach

I've selected three algorithms that have been proposed to improve the performance of optical flow techniques in images with large displacement. I am currently implementing each of these algorithms in MATLAB to compare their results. As a baseline for comparison, I am also extracting SIFT features from the images and matching them. This is the method that is commonly used in navigation to find displacement between images, so it makes a good comparison.

The first optical flow algorithm that I am implementing is a hierarchical standard optical flow calculation, similar to the one which was discussed in lecture. In this algorithm, the images are blurred and downsampled at several different scales using a Gaussian pyramid approach. At the largest scale, the displacement between the two images is calculated using the optical flow equation:

$$I_x \cdot u + I_y \cdot v + I_t = 0 \tag{1}$$

The displacement is calculated at every pixel using a window of pixels around that point. This will enforce a smoothness constraint, since it will prevent a pixel from having a drastically different displacement than its neighbors. A least-squares solution to the optical flow equation is used, as shown in Shi and Tomasi [5], so the solution at each point is:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{2}$$

where the sums are over all points in the window. Once the displacement is calculated at a particular scale, the displacements of pixels in the image are interpolated to the next smallest scale. One of the images is shifted to account for the aready-calculated displacement, then the calculation is performed again at this scale.

The second algorithm is based on the one in Brox *et al*. [3]. It is similar to the previous algorithm, but does not use the linearization of the optical flow equation given in Equation 1. That linearization makes the equation much easier to solve, but also limits it to regions where the gradient is roughly linear. Since the gradient can change rapidly, this only occurs for very small displacements between images, less than about one pixel. With the low frame rate of navigation images, the displacement is usually much larger than one pixel.

This algorithm follows that same technique of calculating the displacement at different scales, but it minimizes a more accurate expression for the image energy instead. The expression includes terms for the brightness constancy assumtion and the gradient constancy assumption:

$$\begin{aligned} E_{Data}(u,v) = \int_\Omega (&|I(\mathbf{x}+\mathbf{w}) - I(\mathbf{x})|^2 \\ &+ \gamma |\nabla I(\mathbf{x}+\mathbf{w}) - \nabla I(\mathbf{x})|^2) \mathbf{dx} \end{aligned} \tag{3}$$

with γ as a weight between the two constraints and $\mathbf{w} = (u, v, 1)^T$. It also includes a smoothness constraint:

$$E_{Smooth}(u,v) = \int_\Omega (|\nabla_3 u|^2 + |\nabla_3 v|^2) \mathbf{dx} \tag{4}$$

The total expression which is minimized is a weighted sum between these constraints.

The third algorithm that I will be implementing is described in Brox *et al.* [2]. It takes advantage of the image structure by segmenting the image into regions and calculating the optical flow separately for each region. A descriptor is calculated for each region and these descriptors are matched. For all regions with good matches, the region in the second image is shifted to the location of the matched region in the first image, and the optical flow from Equation 1 is used to refine the calculated displacement.

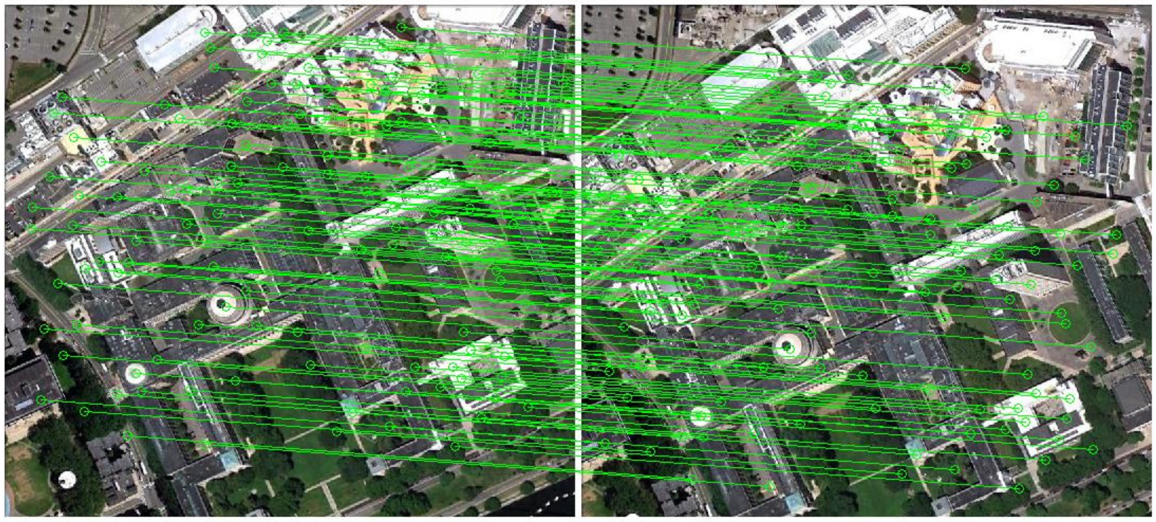

Figure 2. SIFT features matched between two images in the MIT dataset.

4. Preliminary Results

My first step on this project was the background research. I've completed that, and selected the three algorithms that I plan to test. Next, I collected the images that I will use for testing. I collected the images using Google Earth, and the images that I will be using are described in Section 2.

Next, I began to implement the algorithms. For the reference that I am using for comparison, I downloaded SIFT feature extraction code for MATLAB from David Lowe's website.[1] This code provides descriptors and keypoints for features found in each image. I follow the matching technique described in Lowe [4]. I match each feature in one image with a feature in the next image by finding the feature in the second image whose descriptor has the smallest Euclidean distance from the original feature. If this feature is significantly closer than the next closest feature, I consider it to be a match. Matched features between two consecutive images in the MIT dataset are shown in Figure 2.

I have also written most of the code for the three algorithms that I am testing. For the two hierarchical algorithms, I'm following the algorithms as they are described fairly closely. For the segmentation-based algorithm, I am using the image segmentation code described in Arbelaez *et al.* [1], which is available from their website.[2] This is the same segmentation technique that was used in Brox *et al.* [2], so I hope to achieve similar results.

I plan to compare the algorithms by estimating the displacement calculated by the algorithms at different points throughout the image, and comparing each of them to the reference SIFT-matching algorithm. For the SIFT-matching algorithm, I will find the estimated displacement by averaging the displacements of nearby matched features. For the hierarchical approaches, the displacement will be calculated at those pixels (after being interpolated from displacements calculated at larger scales). For the segmentation-based approach, I am assuming that regions move together, and the displacement of a point matches the displacement of the region containing it.

[1] http://www.cs.ubc.ca/ lowe/keypoints/

[2] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):898 –916, may 2011.

[2] T. Brox, C. Bregler, and J. Malik. Large displacement optical flow. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:41–48, 2009.

[3] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In T. Pajdla and J. Matas, editors, *Computer Vision - ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin / Heidelberg, 2004.

[4] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 10.1023/B:VISI.0000029664.99615.94.

[5] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593 –600, jun 1994.