Optical Flow For Vision-Aided Navigation

Elizabeth Boroson
Stanford University
lboroson@stanford.edu

Abstract

In this project, I looked at the use of optical flow algorithms to estimate displacements between images in navigation data. In navigation, computing power is often limited, so feature-matching algorithms are not ideal. However, the large displacements that are seen in this data require modifications to traditional optical flow algorithms. I reviewed several optical flow algorithms that are optimized for larger displacements. I implemented one of these, and compared its results with the results of a feature-matching algorithm. The optical flow algorithm performed fairly well on simulated images in which the only difference was the displacement, but did not perform as well on more realistic image sets. In order to use optical flow for navigation data, the algorithm used would need to be improved to be less susceptible to changes in illumination and image brightness.

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 studied algorithms that apply optical flow techniques to the type of data encountered in navigation.

2. Background

Most recently-developed vision-aiding algorithms for navigation rely on matching SIFT or SURF features between images to determine displacement between them. For example, Mourikis *et al*. [4] implemented a navigation algorithm in which matched features are used to determine constraints between two vehicle poses, and these constraints can be used to improve the pose estimates. Though these features can be matched between images with large displacements, they are not ideal for this purpose due to both the high computational cost of detecting and matching the features and also the poor localization of feature positions. In a UAV with a fairly low-resolution camera, one pixel in an image could correspond to several meters on the ground. A difference in a feature position of a fraction of a pixel could change the calculated displacement enough to impact the quality of the camera as an aiding source.

However, optical flow techniques can also be used to detect the displacement between two images. These techniques can be more accurate, and they can also be much less computation-intensive

than feature detection and matching. In particular, the algorithms lend themselves to being implemented in hardware, which would greatly decrease the weight and power requirements for the computer on board the UAV.

Despite all of these advantages, traditional optical flow techniques are not effective for the large displacements seen in navigation data. They use linearized equations which are only valid in the regions where the image gradient is approximately linear, for displacements of less that one pixel. For a UAV camera with a low frame rate, the displacement may be several pixels, well outside this region.

Several algorithms have been proposed to improve the performance of optical flow techniques over large displacements. Some of these use the technique of blurring and downsampling the image to be able to apply the optical flow equations at larger scales. By downsampling the image, the region in which the linearized optical flow equation is valid increases, so the same technique can be used even with larger displacements.

Another algorithm applies the optical flow equations without linearizing them, so the equations are valid outside of the linear region [2]. The resulting equations from this technique are much more complicated, but are valid for larger displacements. In the algorithm of this type that I studied, there is no closed-form solution to these equations, but they can be solved iteratively.

There are also other recent techniques that are not relevant to navigation data, but may still be interesting. One of these, which would be useful for images where different objects are moving at different rates, involves segmenting the image into regions and applying optical flow to these regions separately [1]. For each region in the image, a descriptor is calculated. Assuming that each region looks the same in both images (or has only been changed by an affine transformation), the regions are matched, giving an estimate of the motion for each region. In navigation, we can generally assume that all motion is due to the motion of the camera, so it is unusual for different regions within an image to move separately. While this technique is interesting, it would not be very useful for navigation.

For my project, I implemented one of these techniques in MATLAB and tested its performance on images similar to those that would be encountered in navigation. I also implemented a SIFT-matching algorithm for comparison with each of these new algorithms. I compared the algorithms in both accuracy of the estimate, or how close it was to truth, and consistency, or what the distribution of estimates was for all pixels. In order to use displacement calculated from optical flow as an aiding source, we would want it to have a Gaussian distribution with a fairly small standard deviation.

3. Approach

I've selected two of the algorithms that have been proposed to improve the performance of optical flow techniques in images with large displacement. I've implemented these algorithms in MATLAB, and have been able to test one of them on images similar to those that would be encountered in navigation. I've also written MATLAB code to extract and match SIFT features from the images in order to compare the optical flow algorithms with the current standard method.

The first optical flow algorithm that I have implemented 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 5×5 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 so-

lution at each point is:

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

where the sums are over all points in the window. Once the displacement is calculated at each point at a particular scale, the displacements of pixels in the image are interpolated back to the original image scale. The second image is shifted to account for the aready-calculated displacement, then the calculation is performed at the next smallest scale. Using this technique, the displacement is in the linear range at the largest scale. As it moves to progressively smaller scales, the images are shifted to be closer together, so the displacement remains in the linear range even as this range gets smaller. In the last iteration, at the original image scale, it is only solving for subpixel motion.

The second algorithm is based on the one in Brox *et al*. [2]. It is similar to the previous algorithm, but does not use the linearization of the optical flow equation given in Equation 1. Instead, it applies several constraints which must be minimized at the correct displacement, and describes a method to iteratively minimize these equations.

The three constraints which are used in this technique are:

1. A brightness constancy assumption. This constraint assumes that a given point has the same intensity in all images. This is the constraint that, when linearized, gives the optical flow equation given in Equation 1. The energy which must be minimized for this constraint is:

$$E_{\text{Brightness}}(u,v) = \int_\Omega (|I(\mathbf{x}+\mathbf{w}) - I(\mathbf{x})|^2)\mathbf{dx} \quad (3)$$

2. A gradient constancy assumption. This constraint assumes that a given point has the same gradient in all images. The brightness constancy assumption is very sensitive to slight changes in the overall image brightness. This constraint would not be nearly as sensitive to small changes in brightness, and would improve the performance of the algorithm in situations where, for instance, there was a slight change in illumination. The energy which must be minimized for this constraint is:

$$E_{\text{Grad}}(u,v) = \int_\Omega (|\nabla I(\mathbf{x}+\mathbf{w}) - \nabla I(\mathbf{x})|^2)\mathbf{dx} \quad (4)$$

3. A smoothness constraint. This ensures that pixels near each other do not have drastically different displacement. In the previous method, this constraint was enforced by using a window of nearby pixels to calculate the displacement at any given point. Here, an energy describing the total variation of the flow field must also be minimized:

$$E_{\text{Smooth}}(u,v) = \int_\Omega (|\nabla_3 u|^2 + |\nabla_3 v|^2)\mathbf{dx} \quad (5)$$

The total energy which is minimized in this technique is a weighted sum of the three constraints:

$$E_{\text{Total}} = E_{\text{Brightness}} + \gamma E_{\text{Grad}} + \alpha E_{\text{Smooth}} \quad (6)$$

The weights can be adjusted based on the type of data. For example, if we know that the entire image should have the same displacement, we can weight the smoothness constraint heavily. If we know that there will be variation in illumination, we can weight the gradient constancy assumption much more than the brightness constancy assumption.

Brox *et al*. also describe an iterative technique to find the displacement that minimizes this energy. The technique has two loops. The outer loop iteratively finds the total displacement, $\mathbf{w} = (u,v)^T$. Updates to $\mathbf{w}$ are calculated by the inner loop, which finds the incremental displacement, $\mathbf{dw} = (du,dv)^T$. On each iteration $k+1$ of the outer loop, $\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{dw}$. On each iteration $l+1$ of the inner loop, $\mathbf{dw}_{l+1}$ is the solution to the system of equations:

$$A \cdot \mathbf{dw}_{l+1} = \mathbf{b} \quad (7)$$

with

$$
\begin{aligned}
A_{11} =&(\psi^{k,l}_{\text{Data}}) \cdot (I^k_x I^k_x + \gamma(I^k_{xx} I^k_{xx} + I^k_{xy} I^k_{xy})) - \alpha((\psi_{\text{S}})_x + 2\psi_{\text{S}} + (\psi_{\text{S}})_y) \\
A_{12} =&(\psi^{k,l}_{\text{Data}}) \cdot (I^k_x I^k_y + \gamma(I^k_{xx} I^k_{xy} + I^k_{xy} I^k_{yy})) \\
A_{21} =&(\psi^{k,l}_{\text{Data}}) \cdot (I^k_x I^k_y + \gamma(I^k_{xx} I^k_{xy} + I^k_{xy} I^k_{yy})) \\
A_{22} =&(\psi^{k,l}_{\text{Data}}) \cdot (I^k_y I^k_y + \gamma(I^k_{yy} I^k_{yy} + I^k_{xy} I^k_{xy})) - \alpha((\psi_{\text{S}})_x + 2\psi_{\text{S}} + (\psi_{\text{S}})_y)
\end{aligned} \tag{8}
$$

$$
\begin{aligned}
\mathbf{b}_1 =&\alpha * ((\psi_{\text{S}})_x (u^k(x,y) - u^k(x-1,y) - du^k(x-1,y)) \\
&+ \psi_{\text{S}}(u^k(x,y) - 2u^k(x-1,y) + u^k(x-2,y) - 2du^k(x-1,y) + du^k(x-2,y)) \\
&+ (\psi_{\text{S}})_y (u^k(x,y) - u^k(x,y-1) - du^k(x,y-1)) \\
&+ \psi_{\text{S}}(u^k(x,y) - 2u^k(x,y-1) + u^k(x,y-2) - 2du(x,y-1) + du(x,y-2))) \\
&- (\psi^{k,l}_{\text{Data}}) \cdot (I^k_x I^k_z + \gamma(I^k_{xx} I^k_{xz} + I^k_{xy} I^k_{yz})) \\
\mathbf{b}_2 =&\alpha * ((\psi_{\text{S}})_x (v^k(x,y) - v^k(x-1,y) - dv^k(x-1,y)) \\
&+ \psi_{\text{S}}(v^k(x,y) - 2v^k(x-1,y) + v^k(x-2,y) - 2dv^k(x-1,y) + dv^k(x-2,y)) \\
&+ (\psi_{\text{S}})_y (v^k(x,y) - v^k(x,y-1) - dv^k(x,y-1)) \\
&+ \psi_{\text{S}}(v^k(x,y) - 2v^k(x,y-1) + v^k(x,y-2) - 2dv(x,y-1) + dv(x,y-2))) \\
&- (\psi^{k,l}_{\text{Data}}) \cdot (I^k_y I^k_z + \gamma(I^k_{yy} I^k_{yz} + I^k_{xy} I^k_{xz}))
\end{aligned} \tag{9}
$$

$$
\psi_{\text{Data}} = (I_z + I_x du + I_y dv)^2 + \gamma((I_{xz} + I_{xx} du + I_{xy} dv)^2 + (I_{yz} + I_{xy} du + I_{yy} dv)^2) \tag{10}
$$

$$
\psi_{\text{S}} = |\nabla_3(u + du)|^2 + |\nabla_3(v + dv)|^2 \tag{11}
$$

Like the previous method, this technique is also applied at different scales. The image is blurred and downsampled, and the displacement is calculated at the largest scale. This displacement is interpolated back to the original image scale, one of the images is shifted by this displacement, and the method is applied again at the next largest scale.

4. Experiment

4.1. Datasets

I have collected three 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. Three consecutive images from the Stanford dataset are shown in Figure 1 as an example of the images I'll be using.

4.2. Algorithms

I implemented the algorithms in MATLAB. For the SIFT feature-matching method 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 [3]. I match each feature in one image with a feature in the next image by finding the feature in the sec-

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

Figure 1: Three consecutive images from the Stanford dataset.

ond 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.

For this method, I generated the displacement at each point in the image by finding the average displacement of nearby features. At each pixel, the displacement is the average of the displacements of all features in a 5×5 window around the pixel's position in the first image. This allows me to calculate a displacement even at points where a feature is not detected, and imposes a weak smoothness constraint, since any feature will affect the displacement at all nearby points. However, this method is not ideal for a few reasons. First, there are points that are not near any features, and the displacement cannot be calculated at these points. Second, any mismatched features will have a disproportionate impact on the calculated displacements at nearby pixels. In navigation, these mismatches would not be a problem because there would also be some information available about the true displacement from other sensors or by extrapolating from previous motion. This additional information could be used to eliminate matches that were clearly incorrect. Furthermore, most navigation algorithms assume that all motion is due to the motion of the camera, so they would use all features to calculate a single displacement, rather than finding the displacement at each pixel.

I also successfully implemented the standard optical flow technique in MATLAB, and have compared it to the feature-matching technique. Unfortunately, I was unable to get the non-linearized optical flow technique working, though the code that I wrote for that is also included. For the optical flow technique, I blurred the image with Gaussian kernels of width 2^n, with $n \in \mathbb{N}$. I then downsampled the image by the same scales. Starting from the largest scale, I calculated the displacement, interpolated it back to the original image scale, shifted one image to match the other, and then repeated the process at the next smallest scale. The total displacement was the output of the final step, where I found the optical flow at the original image scale.

I compared the algorithms based on the accuracy and consistency of the estimates. For the initial tests, where the true displacements are known, the accuracy is the difference between the actual displacement and the average displacement for all pixels. The consistency is the percent of pixels where the calculated displacement was within 3σ of the average. These values only apply to pixels where the displacement could be calculated. For example, for the feature-matching technique, the displacement was not calculated at certain pixels

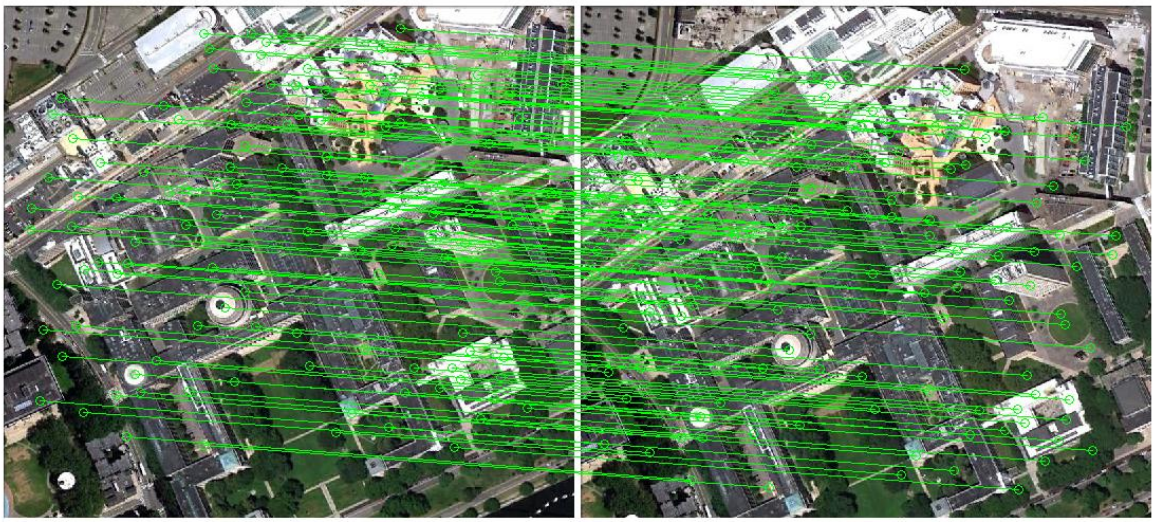


Figure 2: SIFT features matched between two images in the MIT dataset.

because there were no nearby features. Similarly, the optical flow equation cannot be solved in large regions of the image with no variation. These pixels are not included in the results.

4.3. Baseline Comparison

In order to compare the performance of each of the algorithms, I generated a set of small images with known displacements by taking parts of the first image in the Stanford dataset. I took one 256×256 image, then shifted my window by 1 to 5 pixels in the horizontal and vertical directions to get images for comparison. In addition to having known displacements, these images were identical other than the small displacement. They had identical brightness and illumination, and all points in the images had identical displacements. This was especially useful for testing the algorithms, and also allowed me to compare the algorithms based on only the displacement by eliminating any other possible differences between the images.

For each shifted image, I ran each of my algorithms (SIFT feature-matching and hierarchical optical flow) to find the displacement from the original image. The results for each algorithm are shown in Tables 1 and 2. For each algorithm, the results are calculated using all pixels where a displacement was found. The mean and standard deviation of the displacement solution in each direction are given. Also, the percentage of points within 3σ of the mean are given. This allows us to understand the shape of the distribution. For a Gaussian distribution, 99% of the points would be in this range. Ideally, to use optical flow-calculated displacement as an aiding source for vision, we'd like to see a Gaussian distribution.

For these images, both algorithms gave results that were fairly close to the true displacement. The optical flow algorithm's calculated displacements were slightly closer to truth, but it also had much larger standard deviations and many outliers. Figure 3 shows the displacement calculated at each point for a pair of images. From this, it is clear that for the feature matching algorithm, there are many points where the displacement could not be calculated. However, for the points where it was calculated, there are very few outliers. For the optical flow algorithm, on the other hand, there were certain areas of the images where the optical flow could not be calculated well. Outside of these regions, the flow is very

accurate and consistent. This algorithm would be much more useful if those regions were detected so they would not be used for the displacement calculation. This is very similar to what Shi and Tomasi do [5], and an algorithm like theirs could be used to improve the optical flow solution.

4.4. Comparison Using Real Images

For the images that I collected from Google Earth, I performed a similar comparison, though there was no truth data. I ran each algorithm on every pair of consecutive images for each dataset. The resulting mean displacements and their standard deviations are shown in Tables 3-5. Since there is no truth reference, we can only compare the algorithm results to each other.

For all of these datasets, the displacements calculated by the optical flow algorithm are in the right direction, but much smaller than the true displacements calculated by the feature matching algorithm. There are some possible reasons for this. First, the images may have been different enough that the optical flow algorithm based only on image brightness failed. Possible ways to correct this might include normalizing the image brightnesses or revising the algorithm to look at the image gradient instead of instensity, like the gradient constancy constraint in Brox *et al*. [2]. Also, some of the displacements may have been too large for even the largest scale of the algorithm. This could be solved by applying the algorithm at larger scales.

4.5. Applying Navigation Constraints

As mentioned earlier, calculating the displacement at individual pixels is not very relevant in a navigation context. UAVs, where computing speed and power is especially limited, do not generally fly very close to other moving objects. While there are exceptions, we can generally assume that all motion in the images is due to the motion of the vehicle itself. In this case, as long as nearly everything in the image is at the same depth, we only need to find one displacement for the whole image.

In navigation we also have the added benefit of additional information about our motion from other sensors. While vision is very useful, it would be used mainly as an aiding source to improve measurements from inertial sensors and GPS. We could use these sensors and information about previous motion to constrain the displacement that we calculate from the image, and even to eliminate mismatched features. For example, if we know that we've been moving forward, and our sensors don't show that we've changed direction, it's safe to not use features or pixels that show a large backwards or sideways motion. In this case, the displacement that we get from our images is used to refine our estimate of our motion, rather than being the initial estimate.

With all of this in mind, it makes the most sense from a navigation perspective to look at the average displacement of all good pixels for each technique and their distribution. From the results of the small displacement tests, it's clear that while the optical flow technique gives a slightly better estimate of the displacement, this estimate is much less consistent. The standard deviations on the estimate are larger for this technique, and the results are even more spread out, with typically 5-10% of the pixels having values outside the 3σ bound. The feature-matching technique doesn't give perfect results either. Although the standard deviations of the estimates are generally smaller and the distributions are closer to Gaussian, the mean estimates are less accurate.

For the larger images, we have even more outliers with the optical flow technique. Like with the smaller images, it would be useful to determine the regions where the optical flow cannot be calculated and ignore these regions in order to reduce the number of outliers. This would probably improve both the accuracy and the distribution of the solution, and could make an optical flow algorithm a useful and efficent method for finding the displacement between images in navigation.

Image	True Displacement		Mean		σ		% Points in 3σ
	x	y	x	y	x	y	
1 pixel vertical	0	1	0.1859	0.7998	0.2870	0.3089	90.11
2 pixel vertical	0	2	0.3701	1.6151	0.5020	0.5623	94.21
3 pixel vertical	0	3	0.5780	2.3356	0.7413	0.8222	93.67
4 pixel vertical	0	4	0.7723	3.1404	0.9803	1.0838	95.18
5 pixel vertical	0	5	2.4281	5.3702	10.8590	10.5979	97.87
1 pixel horizontal	1	0	0.8131	0.2380	0.3372	0.3862	96.03
2 pixel horizontal	2	0	1.5998	0.3619	0.5506	0.4891	95.63
3 pixel horizontal	3	0	2.3818	0.5778	0.8097	0.7457	93.37
4 pixel horizontal	4	0	3.1479	0.7462	1.0820	0.9678	97.35
5 pixel horizontal	5	0	3.9492	0.9954	1.3424	1.2686	96.57

Table 1: Results of the feature-matching algorithm on the images with small known displacements

Image	True Displacement		Mean		σ		% Points in 3σ
	x	y	x	y	x	y	
1 pixel vertical	0	1	-0.0869	0.8075	1.4063	1.4148	93.83
2 pixel vertical	0	2	-0.1530	2.1592	1.4539	1.7269	94.25
3 pixel vertical	0	3	-0.1054	2.7286	4.2426	4.5277	91.95
4 pixel vertical	0	4	-0.3483	3.3156	3.7681	4.2122	92.05
5 pixel vertical	0	5	-0.3348	4.0252	3.5875	3.5588	91.77
1 pixel horizontal	1	0	0.8221	-0.0045	2.4098	2.7696	94.03
2 pixel horizontal	2	0	2.1002	0.0691	2.2855	2.9317	94.27
3 pixel horizontal	3	0	2.7931	0.1942	3.8002	4.2296	91.27
4 pixel horizontal	4	0	3.1594	0.1151	4.9041	6.2516	92.47
5 pixel horizontal	5	0	4.1807	0.1164	3.9373	4.3918	92.22

Table 2: Results of the optical flow algorithm on the images with small known displacements

5. Conclusion

I was able to compare the hierarchical optical flow algorithm with a feature-matching algorithm for estimating the displacement between two images. While the results of initial tests were promising, the optical flow algorithm did not perform very well when applied to more realistic images like those that would be used for navigation. This may be because of the algorithm's dependence on matching regions actually having the same brightness. This is not always the case for these images, since there may be differences in illumination and also in the camera position and

Image Pair	Feature Matching					Optical Flow				
	Displacement		σ		% Points in 3σ	Displacement		σ		% Points in 3σ
	x	y	x	y		x	y	x	y	
1-2	-38.18	-29.30	19.89	18.28	96.17	-11.87	-6.76	18.30	16.57	85.05
2-3	-37.11	-15.11	15.47	14.84	61.75	-17.78	3.14	16.82	16.12	85.27
3-4	29.28	41.76	9.55	10.74	95.97	4.68	16.89	17.03	17.95	86.16
4-5	19.47	32.65	8.02	9.40	95.30	6.69	22.67	12.64	15.34	84.50
5-6	27.25	41.20	10.20	11.06	95.66	6.01	18.03	16.39	17.04	85.18

Table 3: Results of both algorithms on Stanford dataset

Image Pair	Feature Matching					Optical Flow				
	Displacement		σ		% Points in 3σ	Displacement		σ		% Points in 3σ
	x	y	x	y		x	y	x	y	
1-2	2.01	-16.12	27.66	29.58	98.64	7.50	-20.84	32.08	22.99	96.80
2-3	2.56	-24.63	26.35	26.26	97.98	6.26	-16.23	20.08	18.50	87.05
3-4	-4.28	-33.92	21.97	22.46	68.85	5.64	-15.53	23.62	23.30	86.46
4-5	-9.01	-31.01	16.07	16.68	65.98	5.43	-18.65	17.89	17.36	84.21
5-6	-13.03	-31.76	32.28	31.81	99.75	4.81	-17.17	20.07	17.74	86.78

Table 4: Results of both algorithms on MIT dataset

Image Pair	Feature Matching					Optical Flow				
	Displacement		σ		% Points in 3σ	Displacement		σ		% Points in 3σ
	x	y	x	y		x	y	x	y	
1-2	44.54	28.09	39.98	39.31	96.17	14.75	-2.18	26.77	26.19	81.96
2-3	69.62	52.95	74.38	78.32	95.40	11.01	4.26	32.75	29.91	81.38
3-4	68.47	40.74	48.54	43.44	93.38	7.58	6.81	33.78	30.67	84.46
4-5	60.37	34.44	27.24	24.85	95.92	6.93	5.22	33.08	30.14	84.26
5-6	71.43	38.92	44.10	41.36	96.78	4.41	5.19	35.20	33.17	84.45

Table 5: Results of both algorithms on rural dataset

orientation as the UAV it is on moves.

While this technique does show promise for use on this type of data, these problems would need to be solved first. Some simple steps, like normalizing the image intensities, could be taken to improve how well matching regions' intensities actually match. Additionally, a method like the second algorithm (which I was unable to implement) might be more effective for matching navigation images. The two-loop iterative method for solving the optical flow equations would be fairly slow, so it would probably not be very effective when speed and computational power are limited. However, the gradient constancy constraint that this algorithm uses to reduce its dependence on a region's absolute intensity and rely more on its gradient instead is promising, and could be incorporated into a more effective optical flow algorithm in the future.

References

[1] 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.

[2] 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.

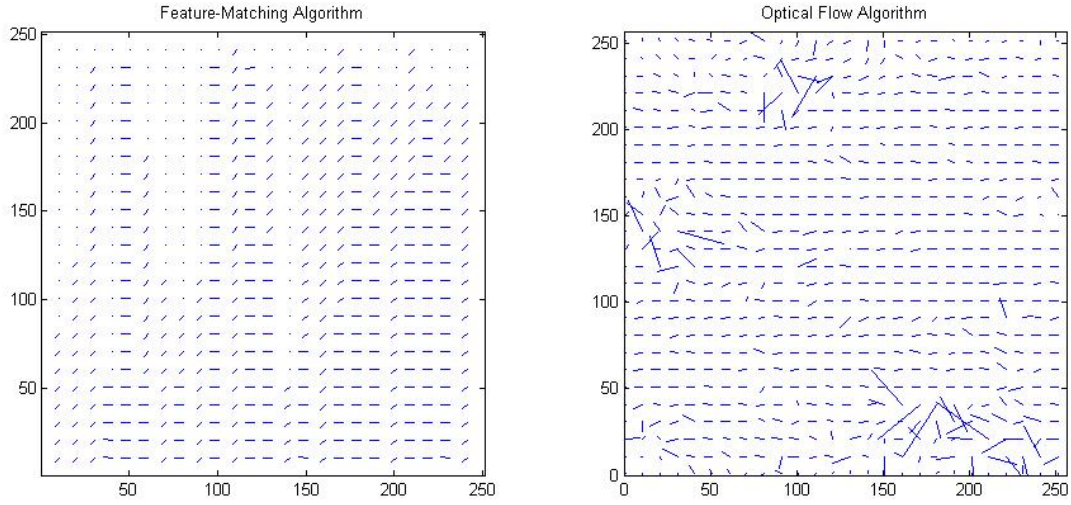
[3] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 10.1023/B:VISI.0000029664.99615.94.

[4] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3565–3572, Rome, Italy, April 10-14 2007.

[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.

]

(a) Two images with a 5-pixel horizontal displacement



]

(b) Displacements between these images calculated with each algorithm

Figure 3: Small image displacements