A Method for Computing Optical Flow via Graph Cuts

Alexander D. Healy
ahealy@fas.harvard.edu

Mike Vernal
vernal@eecs.harvard.edu

May 22, 2002

Abstract

One of the fundamental problems of computer vision is that of *optical flow* – the dense tracking of pixels as they move throughout a series of images. We examine a novel technique for computing the optical flow using an energy minimization problem that can be solved via the use of max-flow/min-cut algorithms. While we find that our algorithm requires more investigation as to the nature of certain constants, our technique shows promise, especially when the number of input images is limited.

1 Introduction

A fundamental problem in computer vision is that of *optical flow*. The goal of optical flow is to compute the two-dimensional motion field over a set of images that describes how each pixel moves. Optical flow has applications to a number of problems in computer vision, including image tracking, video interpolation, and motion sensing.

In this work, we consider a new approach to the question of optical flow based on the work of Kolmogorov and Zabih [Kol, KZb]. Kolmogorov and Zabih present a method for recovering the three-dimensional geometry of a scene using energy minimization techniques via graph cuts. Their method relies upon a set of depth labels to which every point is mapped. They iterate over the label set, greedily re-labelling pixels when a local improvement is encountered.

Our optical flow method approximates the flow at each pixel with one of a fixed set of flow vectors. We select the appropriate flow vectors by iterating over a set of flow vectors and constructing the optical flow using a similar greedy approach.

One of the key strengths of our algorithm is that it works well given only two images; other techniques often require a number of frames in order to compute the resultant optical flow – we compute it using only a source and destination image. We find that our algorithm performs well, though its parametrization requires further investigation.

In section 2, we describe some of the previous work in optical flow. We discuss our problem formulation and approach in sections 3 and 4. We describe our implementation in section 5, and we conclude in section 6.

2 Related Work

A number of techniques for optical flow have been presented in the past twenty years. Barron, Fleet and Beauchemin, in surveying them, divide them into four main categories: differential

methods, region-based methods, energy-based techniques and phase-based techniques [BFB].

Differential techniques generally compute the per-pixel velocity via derivatives of image intensity across a large set of pictures. They generally used first-order and second-order differential methods to constrain the two-dimensional velocity at each point. These techniques suffer from the need for numerous input images, as well as a high-susceptibility to noise.

Region-based methods help to address those cases where numerical differentiation techniques breakdown. Region-based methods attempt to find the best translation that matches image regions. These techniques generally take a top-down approach, matching large areas and then progressively smaller subareas. Quénot proposed a dynamic programming approach to the optical flow problem in [Qué92, Qué]. In his approach, the source and destination images are divided into parallel and overlapping strips. Using a pyramidal divide-and-conquer approach, a displacement field was induced between the two images that estimated the optical flow.

Energy-based techniques, though similar in name to our approach, are actually quite different. Common energy-based techniques, as Barron et al. refer to them, involve estimating the energy of the image through a Fourier filter that reveals the velocity in the frequency domain. Phase-based techniques are similar, exploiting the behavior of band-pass filters to induce the velocity of regions of the image.

3 Energy Minimization Using Graph Cuts

In [KZa], a technique is demonstrated for representing multi-scene reconstruction as a constrained energy minimization problem. The particular form of the energy minimization allowed for efficient approximation using max-flow/min-cut techniques. More generally, in [KZb] the same authors show that an energy minimization problem of binary variables $x_1, \ldots, x_n$ can be solved in polynomial time if $E(x_1, \ldots, x_n)$, the energy function, has the form:

$$E(x_1, \ldots, x_n) = \sum_{a<b} E^{a,b}(x_a, x_b) \qquad (1)$$

where the functions $E^{a,b}$ satisfy:

$$E^{a,b}(0,0) + E^{a,b}(1,1) \leq E^{a,b}(1,0) + E^{a,b}(0,1) \qquad (2)$$

Unfortunately, many interesting energy minimization problems do not consist of binary variables. The technique used in [KZa] is that of α-*expansion*, which proceeds as follows. Given an initial assignment of the (not necessarily binary) variables $x_1, \ldots, x_n \in X$, we construct a binary problem by choosing a value $\alpha \in X$ and then minimize $E(x_1, \ldots, x_n)$ subject to the constraint that either x_i remain the same or $x_i = \alpha$. By repeating such a process for all $\alpha \in X$ a good approximation to a global minimum is achieved.

4 Optical Flow As Energy Minimization

We wish to apply the techniques of [KZa, KZb] to the problem of optical flow. Formally, we are given two consecutive frames $F_1, F_2 \in (\{0, \ldots, 255\}^3)^{h \times w}$ from a video sequence, given as discrete images, and we wish to construct a discrete vector field

$$f : \{1, 2, \ldots, h\} \times \{1, 2, \ldots, w\} \to \mathbb{R}^2 \cup \{\infty\}$$

such that the (i, j) pixel in F_1 moves to position $(i, j) + f(i, j)$ in F_2, and we say $f(i, j) = \infty$ if the pixel (i, j) disappears or becomes occluded in the subsequent frame.

There are two qualities which we would like our optical flows f to have:

- Data Consistency: The pixels close to $(i, j) + f(i, j)$ in F_2 should have similar color to the pixel in position (i, j) in F_1.
- Smoothness: In general, $f(i, j)$ should be close to $f(i \pm 1, j)$ and $f(i, j \pm 1)$ (i.e. the vector field should be nearly "continuous"), except when the data suggests very strongly that there is a edge.

We measure the data consistency at a point $a = (i, j)$ by linearly interpolating the color at $(i, j) + f(i, j)$ in F_2 (as a convex combination of the 4 closest pixels) and by computing the ℓ_2 distance between this color and the color of the (i, j) pixel in F_1 (as elements of $RGB = \{0, \ldots, 255\}^3 \subset \mathbb{R}^3$). Another possibility is that the point (i, j) vanishes in the following frame. In this case we assign a constant penalty $\textit{penalty}_{\textit{data}}$. We denote this measure of data consistency by $E^{a,b}_{\textit{data}}(x_a, x_b)$ (for any b adjacent to a).

Smoothness between positions (i_1, j_1) and (i_2, j_2) (where $(i_1, j_1) - (i_2, j_2) \in \{(\pm 1, 0), (0, \pm 1)\}$, i.e. where the pixels are adjacent) is measured by taking the ℓ_2 distance between the vectors $f(i_1, j_1)$ and $f(i_2, j_2)$. If both of these vectors are ∞ we say the smoothness is 0 (i.e. perfectly smooth); however, if exactly one of them is ∞, we assign a constant penalty $\textit{penalty}_{\textit{smooth}}$. We denote this measure of smoothness by $E^{a,b}_{\textit{smooth}}(x_a, x_b)$.

The energy function which we wish to minimize is $E(\vec{x}) =$

$$\sum_{a<b} c_{\textit{data}} \cdot E^{a,b}_{\textit{data}}(x_a, x_b) + c_{\textit{smooth}} \cdot E^{a,b}_{\textit{smooth}}(x_a, x_b)$$

where the sum is taken over adjacent pixels a and b (i.e. if a and b are not adjacent, we define $E^{a,b}(x_a, x_b) = 0$), and where $c_{\textit{data}}$ and $c_{\textit{smooth}}$ are constants. Following [KZa], we use α-expansion to reduce this energy minimization to a sequence of binary energy minimizations. Since, in the binary case, each of $E^{a,b}_{\textit{data}}$ and $E^{a,b}_{\textit{smooth}}$ satisfy equation (2), their sum does as well, and so $E(x_1, \ldots, x_n)$ satisfies equation (1). In particular, for each α-expansion step we construct a graph with a source, a sink, and one node for each pixel in the image. The edges are placed as follows:

- If $E^{a,b}(1, 0) > E^{a,b}(0, 0)$, an edge of weight $E^{a,b}(1, 0) - E^{a,b}(0, 0)$ is added from the source to vertex a. Otherwise, an edge of weight $E^{a,b}(0, 0) - E^{a,b}(1, 0)$ is added from the vertex a to the sink.
- If $E^{a,b}(1, 0) > E^{a,b}(1, 1)$, an edge of weight $E^{a,b}(1, 0) - E^{a,b}(1, 1)$ is added from the vertex b to the sink. Otherwise, an edge of weight $E^{a,b}(1, 1) - E^{a,b}(1, 0)$ is added from the source to the vertex b.
- An edge of weight $E^{a,b}(0, 1) + E^{a,b}(1, 0) - E^{a,b}(0, 0) - E^{a,b}(1, 1)$ is added from vertex a to vertex b.

Using this construction, a minimum cut corresponds exactly to a minimizing assignment. In particular, the nodes in the source set of the cut will remain unchanged, and the nodes in the sink set will be changed to have value α. [KZa, KZb] contain further details of this graph construction.

5 Implementation & Results

We implemented the above algorithm in C++, using the max-flow/min-cut routines found in [Kol]. For an $n \times m$ image, we created an $n \times m$

set of vectors that mapped the original pixels to their new location, as calculated by our optical flow algorithm. In order to visualize our algorithm, we then linearly-interpolated our computed optical flow via a parameterizable animation algorithm that incrementally applied the optical flow. The original images and four interpolated images can be seen in figures 1 and 2.

One of the greatest drawbacks of this algorithm was the relative inflexibility of the flow vector set. We found that the movement of the pixels could not always be adequately described with our flow vector sets. A number of pixels spuriously mapped to infinity rather than following the actual optical flow. Increasing the size of the flow vector set helped to alleviate this problem, at the expense of computational complexity.

We found that increasing the smoothness coefficient created much more coherent but less adaptable images. For large values of c_{smooth}, we found that our entire image was translated along the path of the major optical flow. For smaller values of of c_{smooth}, we found that the lack of adequate flow vectors caused a lack of image coherence at times.

6 Conclusion

We have shown how optical flow can be formulated as an energy minimization problem that can be approximated using max-flow/min-cut. However, there are many seemingly arbitrary parameters in such an implementation, and we have not sought to find optimal values of these parameters. In figures 1 and 2, we used:

$$\begin{aligned} c_{data} &= 100 \\ c_{smooth} &= 5000 \\ penalty_{data} &= 50 \\ penalty_{smooth} &= 50 \end{aligned}$$

Naturally, it would be worthwhile to run further experiments to determine better values for these constants. It would also be worthwhile to fix the `gcc-2.96` compiler so that it actually worked.

Additional images, included animated images representing optical flow, can be found at `http://www.eecs.harvard.edu/~vernal/courses/cs276-2002/`.

References

[BFB] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *CVPR*, 92:236–252.

[Kol] Vladimir Kolmogorov. Vladimir kolmogorov's web page, http://www.cs.cornell.edu/people/vnk/.

[KZa] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts.

[KZb] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts?

[Qué92] G. Quénot. The orthogonal algorithm for optical flow detection using dynamic programming, March 1992.

[Qué] G. Quénot. Computation of optical flow using dynamic programming.

Figure 1: Frames 1 and 2

Figure 2: Interpolations between the above frames using the computed optical flow