An Algorithm for Real-Time Tracking of Non-Rigid Objects

John Woodfill and **Ramin Zabih**
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

We describe an algorithm for tracking an unknown object in natural scenes. We require that the object's approximate initial location be available, and further assume that it can be distinguished from the background by motion or stereo. No constraint is placed on the object's shape, other than that it not change too rapidly from frame to frame. Objects are tracked and segmented cross-temporally using a massively parallel bottom-up approach. Our algorithm has been implemented on a Connection Machine, and runs in real time (15–20 frames per second).

1 Introduction

Recently there has been a great deal of emphasis in AI on the construction of autonomous artifacts, such as Brooks' proposed mobile robots [Brooks, 1986]. Perception will be a vital part of such robots, as the primary source of information about the dynamic environment. Due to the difficulty of real-time vision, however, mobile robots have often had to rely on other modalities (such as touch or sonar). This in turn has restricted the tasks that such robots can perform. Our goal is to construct visual capabilities suitable for autonomous robots. Real-time performance in unstructured domains is a critical step towards this end.

In this paper, we concentrate on moving objects. In particular, we are interested in tracking objects whose shape is neither fixed nor known *a priori*. Such a capability will be important for navigation, interacting with humans, visual servoing and possibly model construction.

Work on motion tracking has concentrated on rigid objects of known 3-dimensional shape [Gennery, 1982, Verghese *et al.*, 1990]. These approaches have difficulty dealing with unstructured environments, such as office buildings or parks. A robot that required a model of every object that could appear in an office, or every animal that might wander by in a park, would not be very useful. Even if such a model database were available, vision techniques that assume rigidity would have trouble dealing with curtains, people, kittens and other highly deformable objects.

Many vision researchers have attempted to produce detailed information about the 3-D motion of objects, and have concentrated on rigid objects to simplify the task. Our approach is to produce less information about a broader class of objects. In particular, we can handle objects of unknown shape undergoing arbitrary non-rigid motions, as long as the object is intermittently separable from the background along some modality (currently either motion or stereo). The principal restrictions are that the object's shape and position not change too drastically from frame to frame.[1] Under these conditions, we can determine the 2-D motion of the object. The output of our algorithm can be combined with a depth map (from stereo, for example) to produce a useful description of the object's behaviour.

There are several reasons to believe that this approach is worthwhile. First, as noted above, many environments are full of non-rigid objects and objects that one would like to avoid having to model. Second, there is evidence that knowing the 2-dimensional motion of an object is sufficient for many tasks (especially if augmented with stereo depth). Horswill [Horswill and Brooks, 1988], for instance, has constructed a robot which follows moving objects based solely on two-dimensional tracking. Finally, our approach is computationally tractable and yields reasonable results. We have a parallel implementation of our algorithm which runs in real time on a Connection Machine, and we are currently exploring serial implementations on standard hardware.

We begin with some basic definitions and an

[1]We will elaborate on these requirements at the end of section 4.2.

overview of our algorithm. The algorithm has two parts: a motion computation, and a tracking and re-segmenting phase. The motion computation (and a related stereo computation) are described in section 3, and the tracking phase is discussed in section 4. We then describe our Connection Machine implementation and present some results. Finally, in section 6 we survey related work. Pictures of the results of our algorithm are included at the end.

2 Overview

Since we are concerned with real-time performance, the representations used in our algorithm consist entirely of retinotopic maps at the same scale as the image. We have worked on 128 by 128 8-bit gray level images. The object being tracked, for instance, is represented as a boolean bitmap (sometimes called an "intrinsic image" [Barrow and Tenenbaum, 1978]). Such representations are ideal for massively parallel implementation.

Our basic scheme operates on two images at a time. Denote the set of image pixels (points) by P, the gray-level intensity map of the first image by $I_t(P)$, and the second by $I_{t+1}(P)$. Assume that we have distinguished some object in the first image, and wish to determine its location in the second image. Let $O_t : P \to \{0, 1\}$ be a binary predicate (equivalently, a function from points to truth values) that holds at those points in the image I_t that are part of the object. The inputs to our algorithm are I_t, I_{t+1} and O_t, while the output is O_{t+1}. In other words, given a pair of images together with the position of an object in the first image, we compute the position of the object in the second image.[2]

The first step of the algorithm is to compute a dense motion field, which can be viewed as a map $M_t \colon P \to P$. M_t determines which point in the second image corresponds to each point in the first image. The method we use to compute M_t is detailed in section 3.

The second step to the algorithm is to compute O_{t+1} from M_t and O_t. To a first approximation, O_{t+1} will hold at those points in I_{t+1} that correspond to points in I_t where O_t holds. Our approach, described in section 4, applies M_t to O_t and adjusts the result towards motion or stereo boundaries, while also smoothing its shape.

3 Computing Motion

The job of the motion computation is to efficiently produce a dense field of discrete motion displacements. We take a sequential pair of images (I_t and I_{t+1}) as input, and produce a motion map M_t as output. The resulting motion map is a "goes to" map, indicating where each pixel on the old image is likely to have moved to on the new image. The computation we use has two steps: an initial motion estimate, and a smoothing step.[3] Our initial motion estimation scheme uses SSD (Sum of Squared Differences) correlation in a small local area. The smoothing step is similar to that used by Spoerri and Ullman [Spoerri and Ullman, 1987].

Initial motion estimation is intensity based. Motion displacements are bounded by a fixed radius r_v. Motions faster than this will not be detected. This parameter cannot be arbitrarily increased since the initial motion estimation takes time $O(r_v^2)$.

The SSD matching scheme determines for each pixel p on the old image, what the most likely displacement for p is. The similarity measure is an SSD match on p and its 4-connected neighbors. More precisely, for each possible displacement Δ, we compute the degree of mismatch by

$$E(\Delta) \quad = \quad \sum_{\|\delta\|=1} (I_t(p+\delta) - I_{t+1}(p+\delta+\Delta))^2. \quad (1)$$

The displacement for p is $\min_{\|\Delta\| \le r_v} E(\Delta)$. Usually this will assign each pixel a unique displacement.

After the initial estimation step, each pixel has a displacement. However, these displacements tend to be quite noisy. We assume that the scene's actual motions exhibit spatial coherence in order to smooth the initial estimates. We determine the most likely displacement for each pixel p, i.e. the most popular displacement in a region of fixed size r_i surrounding p. If a tie occurs, a displacement is chosen randomly from among the front runners.

The smoothing step has two important properties. First, polling the potential displacements of the neighborhood tends to weed out noisy motions. Second, it results in larger regions of coherent motion, hence improving the motion boundaries.

When two adjacent regions of the image move differently, a motion boundary occurs. Near a motion boundary many of the pixels polled will lie on the other side. As a result the most popular displacement may not get many more votes than the runner up. Various schemes have attempted to detect this situation using statistical tests [Black and Anandan, 1990, Spoerri and Ullman, 1987]. The problem appears to be difficult, however, and we take a different approach.

[2]We will assume that the object's approximate initial position O_1 is supplied in some task-specific manner. This will be discussed briefly in section 5.

[3]Our algorithm has no commitment to the underlying motion computation. Any procedure that produces such displacements, such as [Horn and Schunk, 1981], could be used instead.

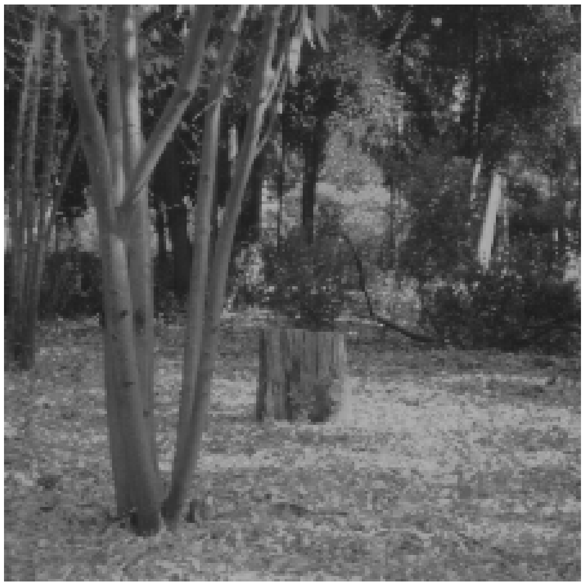
Figure 1: An outdoor scene.

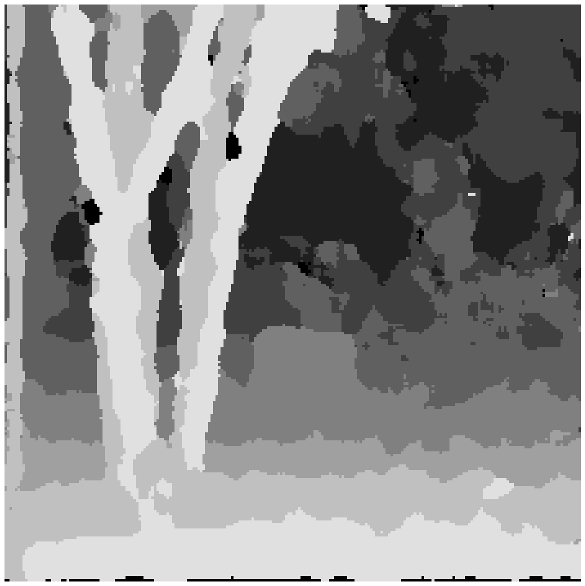
Figure 2: Stereo results for an outdoor scene.

Our motion computation produces motion boundaries that are not precise, but our tracking scheme attempts to compensate for this imprecision.

Our motion computation produces fairly good results provided that several conditions are met. Intensity levels must vary sufficiently; otherwise the initial matching step will give ambiguous results. Objects must neither move too fast, nor too slowly. Objects moving further than r_v between a pair of frames cannot produce reasonable motion displacements. Objects moving too slowly can also fail to produce reasonable motion displacements. No sub-pixel displacements are computed. Hence if an object moves less than a pixel between frames, no motion may be apparent.

3.1 Stereo

In order to produce a stereo displacement map, we perform a computation very similar to the motion one. In our current implementation, we assume that the cameras are aligned so that the epipolar lines are camera scan lines, and also that the principal camera axes are parallel. As a consequence, we only consider unidirectional horizontal displacements. We have obtained good results using an SSD initial estimation phase followed by a smoothing step. In the stereo SSD estimation phase the sum in equation 1 is done for $\delta \in \{\langle 1,0\rangle, \langle 0,1\rangle\}$.

Figure 2 shows the result of our stereo computation applied to the outdoor scene seen in figure 1. The computed displacements are shown; pale intensities in the figure correspond to large displacements, and hence to nearby objects.

4 Tracking and resegmentation

In a scene with a single uniform motion, ignoring the edge of the image, $O_{t+1} = O_t \circ M_t^{-1}$ is correct. However, real image sequences contain multiple non-uniform motions, and in general M_t is not an automorphism.[4] Imagine a scene with a square moving uniformly against a stationary background. In the second image I_{t+1}, immediately behind the square there will be pixels to which no pixel in I_t corresponds. Furthermore, immediately in front of the leading edge there will be pixels in I_t that have been occluded in I_{t+1}. These pixels in I_t will nevertheless determine pixels in I_{t+1} to which they appear have gone, resulting in pixels in I_{t+1} to which more than one pixel in I_t corresponds.

We first compute $\hat{O}_{t+1}$, which is a generalization of $O_t \circ M_t^{-1}$ to the case where M_t is not an automorphism. We then produce O_{t+1} by adjusting $\hat{O}_{t+1}$ towards motion or stereo boundaries.

4.1 Tracking

Formally, define $M_t^{-1}\colon P \to 2^P$ as the inverse of M_t, i.e.

$$M_t^{-1}(p) = \{\, p' \mid M_t(p') = p \,\}.$$

In general $M_t^{-1}(p)$ cannot be guaranteed to be a singleton set, so we define O'_{t+1}, an intermediate result,

[4]Recall that an automorphism is a bijection from a set to itself.

by

$$O'_{t+1}(p) = \begin{cases} \perp, & \text{if } M_t^{-1}(p) = \emptyset, \\ 1, & \text{if } \exists p' \in M_t^{-1}(p) \text{ where } O_t(p') = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Then $\hat{O}_{t+1}$ will be O'_{t+1} except at those points where $O'_{t+1} = \perp$.

At those undecided pixels we use a winner-takes-all solution, taking a poll of the pixels in a neighborhood around p. $\hat{O}_{t+1}$ holds when $O'_{t+1} = 1$ at the majority of these pixels. This tie-breaking scheme has the effect of filling holes caused by non-rigid motion and by the discretization of motion vectors.

4.2 Resegmentation

As noted in section 3, our method for computing M_t is somewhat inaccurate near motion boundaries. Consequently, $\hat{O}_{t+1}$ will tend to have inaccurate boundaries. Since we assume objects are distinguishable from the background by some labeling (such as motion or stereo displacements), we adjust $\hat{O}_{t+1}$ towards the boundaries induced by the labeling to produce O_{t+1}. This adjustment is accomplished by applying a winner-takes-all local algorithm that adds and deletes pixels near the boundaries of $\hat{O}_{t+1}$. We will describe the adjustment of $\hat{O}_{t+1}$ under the assumption that motion boundaries are used, but the same scheme can be used for stereo, or for other modalities.

As the motion displacements are discrete, M_{t+1} (produced from I_{t+1} and I_{t+2}) defines a set of connected components of pixels with the same label (motion displacement). These connected components are separated by motion boundaries. The resulting O_{t+1} ought to have two properties: its perimeter should be consistent with these motion boundaries, and it should diverge from $\hat{O}_{t+1}$ as little as possible.

The first of these properties can be naturally measured component by component. A given pixel p is part of exactly one connected component. $\hat{O}_{t+1}$ will hold at some of the pixels in that component; in particular, $\hat{O}_{t+1}$ will either hold at the majority of the pixels or will not. We define p to be a *minority* pixel if $\hat{O}_{t+1}$ holds at p but not at the majority of pixels in p's component, or if $\hat{O}_{t+1}$ does not hold at p but holds at the majority of pixels in p's component. The number of minority pixels is a natural measure of how inconsistent $\hat{O}_{t+1}$ is with the boundaries. In particular, if there are no minority pixels, then the boundaries of $\hat{O}_{t+1}$ line up precisely with the boundaries. We would like to produce a result O_{t+1} which minimizes the number of minority pixels.

The second desired property is also easy to measure. We would like to produce a result O_{t+1} which agrees with $\hat{O}_{t+1}$ wherever possible. Define p to be a *changed* pixel if $O_{t+1}(p) \neq \hat{O}_{t+1}(p)$. The number of changed pixels (the Hamming distance) is a natural measure of how different O_{t+1} is from $\hat{O}_{t+1}$. In particular, if there are no changed pixels, O_{t+1} and $\hat{O}_{t+1}$ are identical. We would like to produce a result O_{t+1} which minimizes the number of changed pixels.

Ideally, one would like to satisfy both these constraints at once. But if we start with $\hat{O}_{t+1}$, reducing the number of minority pixels introduces changed pixels. Similarly, trying to keep the number of changed pixels at zero cannot reduce the number of minority pixels. Thus it is impossible to minimize both measures simultaneously. Rather, we must minimize some combination of the two measures. Many combinations are possible; our approach is to minimize the number of minority pixels first, and only secondarily to minimize the number of changed pixels. We now describe a simple algorithm which is provably optimal under this criterion.

To produce O_{t+1} from $\hat{O}_{t+1}$, consider each connected component in turn. If $\hat{O}_{t+1}$ holds at the majority of pixels in the component, then O_{t+1} will hold at *every* pixel in the component. Otherwise, O_{t+1} will hold at no pixel in the component. The resulting segmentation, O_{t+1} is the union of the set of connected components in which the majority of pixels are in $\hat{O}_{t+1}$.

Unfortunately this simple algorithm has two undesirable properties. Both these properties result from the definition of minority pixels, which is non-local in nature.[5] First, the adjustments can be too drastic. If I_{t+1} and I_{t+2} are identical, M_{t+1} will be the identity map. There will be one connected component, and hence either $\hat{O}_{t+1}$ will consist of less than half the pixels, and O_{t+1} will be empty, or $\hat{O}_{t+1}$ will be a majority and O_{t+1} will be all pixels. Second, the labeling of connected components is slow on parallel machines such as the Connection Machine. To ameliorate these shortcomings we use a purely local computation, that is intended to approximate the notion of minority pixels in a local area around each pixel.

If there is no motion boundary within a local region around p, $O_{t+1}(p)$ is defined to be $\hat{O}_{t+1}(p)$. This condition on the adjustment phase allows us to track objects that are only intermittently separable from the background. So, for example, we can track an object which moves, and then stops, and then moves again, such as a pendulum.

The adjustment phase of the tracking algorithm is

[5]Finding the minority pixels requires computing the connected components of pixels with the same motion displacement. This can involve communication between pixels that are arbitrarily far apart.

intended to recover from inaccuracies in $\hat{O}_{t+1}$. If the distance between the boundaries of $\hat{O}_{t+1}$ and the motion boundaries gets too large, local adjustment cannot recover. In particular, when a large expanse of an object comes into view suddenly, the motion boundary on the periphery of the object may be far from the boundaries of $\hat{O}_{t+1}$. Thus the algorithm does not deal well with objects which change shape drastically from frame to frame.

An additional limitation comes from internal motion boundaries (in other words, motion boundaries that occur within the object we are tracking, rather than between the object and the background). If $\hat{O}_{t+1}$ ends up nearer to an internal motion boundary than the external motion boundary, local adjustment may actually force O_{t+1} to be further from the boundaries of the object than $\hat{O}_{t+1}$.

5 Our Implementation

Our algorithm has been implemented on a 16-kiloprocessor Connection Machine at Xerox PARC. The algorithm runs in real time at a rate of 15–20 frames per second on 128 by 128 images (subsampled from the 512 by 512 output of a camera). This includes the overhead for image digitization and the shipment of image data into the Connection Machine, which occupies about a third of our running time. The basic motion computation described in section 3 is very fast; we can compute M_t at a rate of over 50 frames per second. The stereo computation shown in figure 2 is comparably fast.

Our implementation has been tested on several dozen image sequences, which are typically 100 frames each. We have tried to make these sequences as different from each other as practical.

Some results are shown at the very end of this paper. Each column consists of four images. The top image is the original image. The second is O_1, the initial location of the object in that image. The third image is a later image in the sequence, and the fourth image is the location of the object in the later image. All the images are 128 by 128 except the kitten sequence, which is 256 by 256.

The initial segmentation is done on a case-by-case basis. In the data shown none of the initial segmentations are precise (they always include some of the background, and exclude some of the object). Empirically our algorithm is not very sensitive to the exact initial segmentation.

6 Related Work

Probably the closest approach to ours is the use of active contour models (also called "snakes") for tracking, first suggested in [Kass *et al.*, 1987]. If supplied with an appropriate potential function (such as one with a local minimum along motion or stereo boundaries), snake-based tracking might work for our purposes. However, there are important differences. Snake-based tracking does not take advantage of any bottom-up motion estimates. Instead, the snake from image I_t is placed on image I_{t+1} and relaxed into place. In the event that the background also contains motion or stereo boundaries near where the object was, the snake will not necessarily track the object, even if the motion of the object is clear and unambiguous. In addition, one needs to design a potential function that will enable the snake to seek motion or stereo boundaries, a problem that is not necessarily trivial. Finally, our approach has produced real-time performance on real data; we are not aware of any similar results for snake-based tracking.

Horswill's work [Horswill and Brooks, 1988] has somewhat similar goals to our own, in that he wants to deal with people and other non-rigid objects. However, his approach is highly specialized to the environment that his robot inhabits and to the objects he wishes to follow. His robot can follow highly textured objects against a background that does not have much texture, at the very low resolution that his system uses. In the examples that interest us, both the objects of interest and the background have a lot of texture. More generally we do not believe that there are simple bottom-up primitives (such as amount of texture) which can segment an arbitrary scene correctly.

7 Future Work

Most of the work we have done has assumed that the object to be tracked can be separated from the background by motion. We hope that stereo boundaries will prove a better modality for segmentation than motion. We have an initial implementation of tracking with stereo boundaries, which is fast but not yet real-time. A longer term goal is to use our system for for robotic applications. An implementation of our algorithm on standard serial hardware is also currently underway.

Our algorithm currently yields only 2-D positional information about the object. While this can produce some 3-D information when combined with stereo, additional descriptions of the object's structure will undoubtedly prove necessary for some applications.

Acknowledgements

We wish to thank Harlyn Baker for his advice and guidance. John Lamping provided useful insights on our algorithm. Jon Goldman of Thinking Machines and Hal Moroff of Datacube supplied technical support. We also received useful comments from David Chapman, and help from numerous people at Xerox PARC.

John Woodfill is supported by a Shell Doctoral fellowship. Ramin Zabih is supported by a fellowship from the Fannie and John Hertz Foundation. Additional financial support was provided by SRI, Xerox PARC, and Hewlett-Packard. We also wish to thank the Center for Integrated Facility Engineering.

References

[Barrow and Tenenbaum, 1978] H. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. In A. Hanson and E. Riseman, editors, *Computer Vision Systems*, pages 3–26. Academic Press, 1978.

[Black and Anandan, 1990] Michael Black and P. Anandan. Constraints for the early detection of discontinuity from motion. In *Proceedings of AAAI-90, Boston, MA*, pages 1060–1066, 1990.

[Brooks, 1986] Rod Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, March 1986.

[Gennery, 1982] Donald Gennery. Tracking known three-dimensional objects. In *Proceedings of AAAI-82, Pittsburgh, PA*, pages 13–17, 1982.

[Horn and Schunk, 1981] Berthold Horn and Brian Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[Horswill and Brooks, 1988] Ian Horswill and Rod Brooks. Situated vision in a dynamic world: Chasing objects. In *Proceedings of AAAI-88, St. Paul, MN*, pages 796–800. American Association for Artificial Intelligence, Morgan Kaufman, August 1988.

[Kass *et al.*, 1987] Michael Kass, Andrew Witkin, and Dmitri Terzopolous. Snakes: Active contour models for machine vision. In *International Conference on Computer Vision*, pages 259–268. IEEE, 1987.

[Spoerri and Ullman, 1987] Anselm Spoerri and Shimon Ullman. The early detection of motion boundaries. In *International Conference on Computer Vision*, pages 209–218, 1987.

[Verghese *et al.*, 1990] Gilbert Verghese, Karey Gale, and Charles Dyer. Real-time, parallel motion tracking of three dimensional objects from spatiotemporal sequences. In Vipin Kumar, editor, *Parallel Algorithms for Machine Intelligence and Vision*, pages 310–339. Springer-Verlag, 1990.