Detecting Moving Objects With a Moving Camera by Comparing Edge Contours*

Daniel P. Huttenlocher
Eric W. Jaquith

TR 94-1405
January 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* This work was supported in part by National Science Foundation PYI grant IRI-9057928 and matching funds from Xerox Corp., and in part by Air Force AASERT F49620-92-J-0247.

Detecting Moving Objects with a Moving Camera by Comparing Edge Contours*

Daniel P. Huttenlocher and Eric W. Jaquith
Computer Science Department
Cornell University
Ithaca NY 14853 USA

Abstract

This paper introduces a method for detecting moving objects in a monocular image sequence that is obtained using a moving camera. The method first estimates the motion of the edge contours in a given image frame, by recovering a transformation that best matches each edge contour with the edges in the subsequent frame. Any contour that is not well accounted for by a single transformation is split into subparts. The transformation of each edge contour together with the relative spatial locations of the contours is used to partition the image into regions with similar motions. Hypotheses about the locations of possible moving objects are then made based on these motion regions. One of the key aspects of the approach is that it is based on estimating the motion of entire edge contours, as opposed to recovering a velocity field that measures the motion of individual points. We present some examples for image sequences taken of animate objects using a hand-held video camera.

Keywords: Motion estimation, motion segmentation, edge matching.

*This work was supported in part by National Science Foundation PYI grant IRI-9057928 and matching funds from Xerox Corp., and in part by Air Force AASERT F49620-92-J-0247.

1 Introduction

We consider the problem of detecting moving objects in a scene, using a monocular image sequence obtained from a moving camera (where the camera motion is unknown). The ability to detect moving objects with a moving camera is important for a variety of tasks including navigation, surveillance and tracking. The problem involves first measuring the motion in the image, and then segmenting the image based on motion differences. The resulting motion regions delineate possible locations of moving objects in the image. Our approach to this problem operates solely in the image domain without three-dimensional information (in contrast with methods such as [5, 10]), and is based on comparing edge contours in two successive frames of an image sequence. For each edge contour in a given frame, a transformation is computed that maps the contour onto the edge contours in the subsequent frame (rather than computing point-based motion estimates as in [1, 2, 7, 8, 9]). Any contours that are not well accounted for by a single transformation are split into subparts (i.e., new smaller contours) which are then transformed separately. The transformation for each contour is used as a measure of the motion of that contour in the image.

The motion of each contour, together with the relative spatial locations of the contours, is used to segment the image into regions with similar motions. The Voronoi diagram (or distance transform) provides a natural representation of the spatial relations among a set of elements. The Voronoi diagram of a set P of points in the plane specifies for every point x in the plane which point of P is closest to x (and thus also gives the distance to this closest point). Therefore the distance transform of the edge contour points in an image specifies for each point of the image the nearest edge contour point. As an initial motion segmentation of an image, we simply assign each image point the transformation of the nearest edge contour point. These regions are then merged when neighboring regions have similar motions (as described below). This yields a partition of the image based on both spatial proximity of edges and on similarity of motions.

The motion regions obtained from this process appear to provide a good basis for hypothesizing the locations of moving objects in an image. For example, the first row of Figure 1 shows two successive frames from an image sequence in which a cat is walking towards the right with the camera moving so as to follow the cat (thus the rest of the image is moving towards towards the left). The first image in the second row shows the edge contours from the two frames superimposed on each other, with the gray edges corresponding to the first image frame and the black edges corresponding to the second image frame. Note that the gray edges are generally slightly to the right of "corresponding" black edges (because most of the image is moving leftwards). The second image in the row shows the transformed edges from the first image frame superimposed on the edges from the second frame. The transformed edges (in gray) are now generally more or less on top of a "corresponding" black edge, rather than off to the side. The transformation of each gray edge, which aligns it with a black edge, is what our method recovers as the motion for that edge (note that a given edge contour may be split into multiple parts with different motions, as discussed below).

The third row of Figure 1 shows the motion regions computed from the edge motions. The first image in the row contains the regions that result from the Voronoi-based process described above. Each point in the image is assigned the motion of the nearest edge contour point, and then neighboring regions with similar motions are merged. Each distinct motion is shown as a different gray level. The second image in the row shows the best guess as to a moving object in the image. This is based on filtering out the largest area motion region as the "background" and then using information about the remaining motion regions (e.g., area, difference from neighboring motions) to hypothesize moving objects. For this image there was only one significant-sized motion region other than the background, which corresponds to the body of the cat (moving slightly down and to the right, whereas the background is moving to the left). Note that the head of the cat moved somewhat differently from the rest of the body, and was not found to be part of the same region.

The basic steps of the method are as follows:

1. Compute the intensity edges for two successive frames of the image sequence using a method similar to [3]; call these E_t and E_{t+1}.

2. For each connected component of E_t, which we call an edge contour C, find the transformation T that best aligns the transformed contour, $T(C)$, with a local neighborhood in E_{t+1} (using a measure based on the Hausdorff distance).

3. Perform an initial segmentation of image frame t based on the Voronoi diagram of the edge contours, E_t, assigning every image pixel the motion of the nearest edge point.

4. Merge neighboring image regions (as computed in the previous step) which have similar motions. Hypothesize distinct motion regions as possible moving objects.

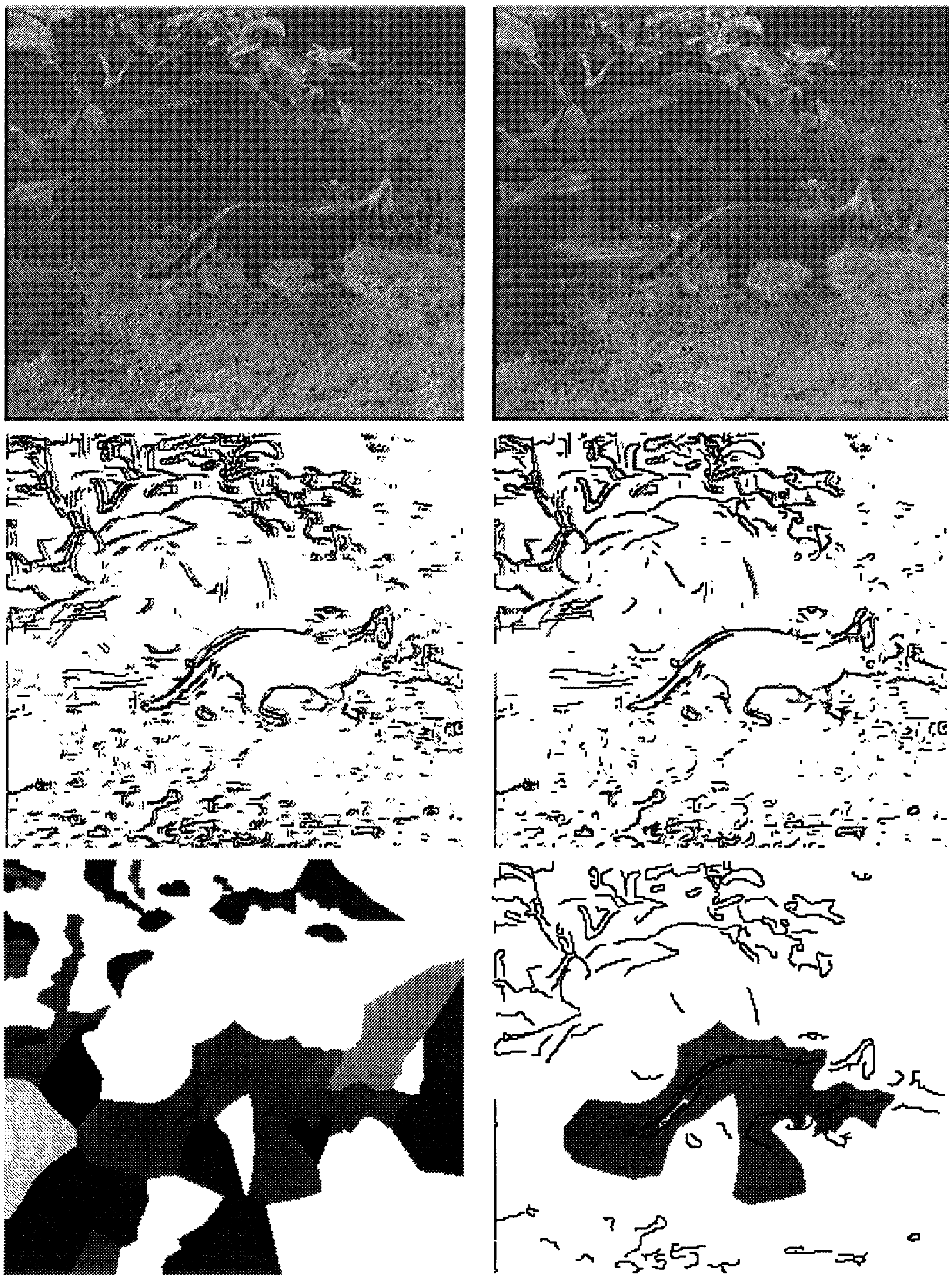

Figure 1: Example motion segmentation (see text). First row: two successive frames of an image sequence. Second row: edges from both frames superimposed (left), translated edges from first frame superimposed on edges from second (right). Third row: resulting motion regions (left), largest motion region in gray (right).

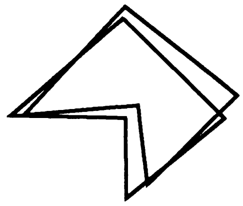

Figure 2: A polygonal curve undergoing a slight translation and rotation. The local difference vectors, $V(s)$, between the two curves vary considerably at different locations.

Note that when a single contour, C, in E_t is actually part of two structures that are moving differently, then the transformation T computed for that contour in Step 2 will not accurately reflect the true motion. Thus once T is computed for a given contour, our method analyzes whether this transformation accurately reflects the motion of the entire contour, and if not the contour is split into pieces and new transformations are computed for each piece (these new pieces are then recursively split if necessary).

In the following section we discuss how the best transformation T is computed for each contour, and in Section 3 we discuss the splitting of contours that have multiple underlying motions. Section 4 describes how the motion regions are computed, using a form of Voronoi diagram, and discusses the criteria that are used to merge neighboring motion regions. Then Section 5 presents some experimental results.

2 Recovering Motion by Matching Edge Contours

We estimate the motion of entire edge contours, rather than recovering a velocity field that measures the motion of individual image points, because the resulting motions seem better suited to the problem of detecting moving objects. That is, given an edge contour C we recover a single transformation T that maps C to the "best matching" contour C' in the subsequent frame of the image sequence. This transformation provides a concise representation of the motion of C as long as the true motion is reasonably well approximated by the class of transformations (e.g., translation or rigid motion) used in mapping C to C'. The more common approach to motion estimation is to represent the motion of a contour using a vector field $V(s)$ that specifies for each point of $C(s)$ the corresponding point of C'. This representation is considerably less concise. For example, consider the polygon in Figure 2 which has undergone a rigid motion $T = (x, y, \theta)$ (a slight translation and rotation), and for which the local motion vectors $V(s)$ differ greatly at different points along the curve.

Moreover, $V(s)$ cannot in general be recovered by direct local computations involving C and C', because only the component of $V(s)$ normal to C can be directly observed (this is referred to as the aperture problem). Thus in order to compute $V(s)$, certain assumptions about the motion are usually made (e.g., that it is rigid).

For each edge contour, C, at time t we seek the "best" transformation of that contour, under some allowable class of transformations. In this paper we limit the class of transformations to translation, although rigid body motions or even affine transformations may be useful for certain kinds of image sequences. Even restricting the transformations to translational motion, however, we are able to segment moving objects in relatively unstructured image sequences. This is primarily due to the fact that locally (in time) most motions seem to be relatively well approximated by a small translation. The sequences that we analyze are obtained at 30 frames per second, and thus the motions between successive frames are only a few pixels (out of a several hundred pixels). The sequences are primarily of animate objects moving around in the world, taken with a hand-held video camera (and thus do not have very large rotational motions in the image plane).

We use two criteria to determine what translation T of a contour C from E_t is the best one. The first criterion is based on how well the translated contour, $T(C)$, matches the edge contours at the next time frame, E_{t+1}. The second criterion is based on the magnitude of the translation T, because smaller motions are in general preferable to larger ones. To limit the search for the best translation, we restrict the maximal allowed motion between two image frames (currently it is ± 7 pixels). Thus, the approach is to evaluate the quality of the match of $T(C)$ over the range of allowable translations, and then pick the best match that is also the smallest translation. The exact details of how this is done are described below.

We first discuss the distance measure used to compare $T(C)$ with E_{t+1}, and then discuss how the allowable translations are searched.

2.1 The Hausdorff Distance

An edge contour $C = \{c_1, \ldots, c_m\}$ and the edges at the subsequent time frame $E_{t+1} = \{e_1, \ldots, e_n\}$ are both finite point sets, where each point represents a pixel at which an edge was detected. Let $T(C) = \{c + T \| c \in C\}$ be the contour C translated by T. The directed Hausdorff distance between $T(C)$ and E_{t+1} is defined as

$$h(T(C), E_{t+1}) = \max_{c \in T(C)} \min_{e \in E_{t+1}} \|c - e\|, \tag{1}$$

and $\|\cdot\|$ is some underlying norm. Here, we use the L_2 or Euclidean norm.

In effect, $h(\cdot,\cdot)$ ranks each point of $T(C)$ based on its distance to the nearest point of E_{t+1}, and then the largest ranked such point (the most mismatched point of $T(C)$) specifies the value of the distance. Intuitively, if $h(T(C), E_{t+1}) = \varepsilon$, then each point of $T(C)$ is within distance ε of some point of E_{t+1}, and there also is some point of $T(C)$ that is exactly distance ε from the nearest point of E_{t+1} (the most mismatched point). Note that the directed distance $h(T(C), E_{t+1})$, given above, is not in general equal to $h(E_{t+1}, T(C))$ — in particular there will generally be points of the image, E_{t+1}, that are far from any point of the translated contour, $T(C)$.

The directed Hausdorff distance can be computed using the distance transform (or Voronoi diagram) of E_{t+1}, where each point of $T(C)$ acts as a "probe" into the distance transform. The distance transform of a binary image specifies for each pixel, the distance to the nearest nonzero pixel of the binary image. Thus the distance transform is zero at each point that corresponds to a nonzero pixel, one at each immediate neighbor of a nonzero pixel, and so on. This distance map specifies for any probe point, c, the value $\min_{e \in E_{t+1}} \|c - e\|$. Thus the directed Hausdorff distance, given in (1), is simply the maximum over the probed distance transform values for the points of $T(C)$.

Note that the computation of $h(T(C), E_{t+1})$ does not involve determining an explicit pairing (or correspondence) between points of $T(C)$ and points of E_{t+1} (for example many points of $T(C)$ may be close to the *same* point of E_{t+1}). This contrasts with most model-based matching methods which determine a correspondence between points of a "model" and an "image".

As described above, $h(T(C), E_{t+1}) \leq \varepsilon$ only when every point of $T(C)$ is within ε of some point of E_{t+1}. However we do not in general expect *every* point of $T(C)$ to be within ε of some point if E_{t+1} (e.g., part of the contour may have become occluded at the next time, or may otherwise not match). We therefore use a generalization of the directed Hausdorff distance,

$$h_k(T(C), E_{t+1}) = \underset{c \in T(C)}{k^{\text{th}}} \min_{e \in E_{t+1}} \|c - e\|, \tag{2}$$

where k^{th} denotes the k-th largest value of a set of values. We then set $k = fm$ for some fraction $0 \leq f \leq 1$ (where recall m is the number of points in C). Thus, when $f = 1$ this is the same as maximizing as in equation (1) and when $f = .5$ it is the same as computing the median rather than the maximum. This generalized Hausdorff distance can be thought of as separately accounting for missing data or gross outliers (using the fraction f) and for small perturbations in the locations of the points (using the distance ε). That is, f specifies the fraction of the contour C that we expect to be present, and ε allows the contour to be slightly distorted in ways that are not accounted for by the translation T.

2.2 Finding Possible Translations

For each contour C in the image, we want to consider all of the translations over some local range of translations (we currently use a ± 7 pixel grid of translations). For each such translation we evaluate $h_k(T(C), E_{t+1})$, in order to find possible good translations of C. While in principle computing these matching translations involves computing h_k for each translation, the method described in [4] uses certain properties of the distance to speed up the computation, and need not fully evaluate h_k for each translation. Rather, given a specified ε and $k = fm$, for each translation where $h_k(T(C), E_{t+1}) \leq \varepsilon$ the method returns a pair (d, f') as a measure of the quality of the match at that translation, or indicates that there is no match for translations where the distance is larger than ε. The pair (d, f') asserts that the fraction f' of the points of $T(C)$ are within distance $d = h_k(T(C), E_{t+1})$ of some point of E_{t+1} (note that $f' \geq f$ and $d \leq \varepsilon$). One way to think of this is that the larger the fraction f' the more points of $T(C)$ are within ε of some point of E_{t+1}, thus larger values of f' are better. For a given fraction f', the smaller the value of d the better.

In the current implementation we consider those edge contours that are larger than some minimum size (currently 15 pixels), and for each such contour use the Hausdorff distance computation to find all translations in the given range for which $h_k(T(C), E_{t+1}) \leq 2$, where $k = .6m$ (i.e., we use $\varepsilon = 2$ and $f = 0.6$). A matching translation is thus any translation that brings 60% of the points of $T(C)$ within two pixels of some edge point in E_{t+1}. We score each matching translation by its fraction (larger is better), and then by its distance for two translations that have the same fraction. These match scores are then used in combination with the magnitude of T to identify the "best" translation.

2.3 The Best Translation

In defining the "best" translation of a contour, C, we use both the quality of the match (as given by the directed Hausdorff distance) and the magnitude of the translation. This is because smaller motions are nearly always preferable to larger motions (i.e., given multiple matches of approximately the same quality, the interpretation that involves a smaller motion is usually better). Using the match score alone could result in choosing a translation with a relatively large motion, even though other translations of smaller magnitude had match scores nearly as good as the one with the best score. In order to combine the criteria of minimizing the magnitude of the motion and maximizing the match quality, we consider all of the acceptable translations (i.e., those for which there is a (d, f') pair with $d \leq \varepsilon$ and $f' \geq f$). We initially find the translation, T_i, with the smallest magnitude (i.e., closest to a translation of $(0,0)$). This initial translation is then refined by searching in a local area around T_i for better matches, as it should be possible to "slide" from T_i to the final best match translation, without any intervening bad translations.

For a given edge contour, C, an $N \times N$ grid of translations centered at $(0,0)$ and with $\pm\lfloor N/2 \rfloor$ pixels in x and y is considered (we currently use ± 7 pixels which forms a 15×15 grid). At each location of this grid there is either a (d, f') pair or an indication that there was no match at this translation (currently we use a Hausdorff distance of more than 2 with a fraction of 0.6 to mean there is no match). We pick the matching translation with the smallest magnitude (i.e., the one closest to the center of the grid). If there is more than one match with the same magnitude translation then the one of them with the largest fraction, f', is used. If there are also multiple matches with the same fraction then the one of them with the smallest distance, d, is chosen. This translation is the initial hypothesis, T_i.

Say that $T_i = (x_i, y_i)$ and the fraction returned by the matcher for that translation is f_i'. This hypothesis is improved by finding the "neighboring" translation in the grid that has the highest fraction. The neighborhood is defined to be the connected component of translations having a fraction at least as large as f_i'. That is, each location in the grid of translations is colored black if its fraction is larger than f_i', and additionally location (x_i, y_i) is colored black. The other locations are colored white. Then the connected component of black pixels containing (x_i, y_i) is found. This component is defined to be the local neighborhood of (x_i, y_i) – those translations that have a better fraction and have a connected path of such good translations back to (x_i, y_i). The point in the local neighborhood with the largest fraction (or the centroid of such locations if there is more than one) is used as the final hypothesized translation.

3 Splitting Contours

When a given contour, C, is composed of multiple subparts with different motions, then a single translation T computed for the entire contour will in general not accurately reflect the underlying motions of the contour. We check for this kind of situation by comparing each point of $T(C)$ with the corresponding point of C, in order to determine whether the match for particular portions of $T(C)$ was not improved by the translation. In such a case, C is split into subparts which are then matched separately.

The fact that the directed Hausdorff distance, used for finding the best matching location, is based on a rank order statistic (quantile) of the distance transform values, means that each point of the translated contour should be closer to some edge point in E_{t+1} than the corresponding point of the original untranslated contour was. This defines a local measure which determines whether on not the match for each point of the contour improved (decreased) as a result of the translation. Thus, we expect that for most of the points $c \in C$, $T(c)$ will be closer to some point of E_{t+1} than c is, or in other words, we expect that

$$D_{t+1}(T(c)) < D_{t+1}(c), \tag{3}$$

where D_{t+1} is the distance transform of E_{t+1} (i.e., $D_{t+1}(c)$ is the distance from c to the nearest edge point in E_{t+1}). For each point c, we call c *good* if inequality (3) holds (the distance from c to the nearest edge point in E_{t+1} improved as a result of the translation) or if $D_{t+1}(c) = 0$ (because it is impossible for $D_{t+1}(T(c))$ to be negative).

Given the good and bad points of a contour C, the connected components of the bad points are found. Each component that is larger than the minimum component size (currently 15 pixels) becomes a new contour, or subpart. Remaining bad points that do not belong to some component above the minimum size are reclassified as good. All of the good points then form another contour. Thus note that if no new contours are formed based on the bad pixels, then the good points are the same as the original contour points, and the original contour will remain unsplit.

If one or more new contours were created from the bad points, then each of the new contours is compared with E_{t+1} in order to find the best translation of that contour. Each of these contours may itself (recursively) be split, and so on. The result of this process is that most of the points of each translated contour $T(C)$ match E_{t+1} better than do the corresponding points in the original untranslated contour C. The contours are simply split into pieces (down to the minimum component size of 15 pixels) until this criterion is met.

The reasoning behind the splitting computation is that those portions of a contour for which the match did not improve may be parts of some other underlying motion, and thus they are split into their own contours and re-matched. The remaining points of the contour (those that improved) are left as a single "contour", even though they may not be connected any longer, because these points all seemed to move together (i.e., they all got better as a result of the translation). For the image sequence of the cat, illustrated in Figure 1, none of the contours needed splitting, but for the sequence in Figure 3, contour splitting was done.

4 Voronoi-Based Motion Regions

The computations described above yield a translation for each edge contour in E_t, such that the translated contour matches E_{t+1} better than the original contour did. In this section we describe how the motions of these contours are used to partition the image at time t into an array A with regions of similar motions. The basic idea is to assign each point of A a translation based on the translation T of the nearest edge contour. We use a method similar to that described in [6], which given a binary image computes an array that specifies for each (x, y) location the location of the closest nonzero point in the binary array. Thus, given the edges E_t, the translation for each edge contour point can be distributed to the image pixels that are closest to that point.

The algorithm first performs a row-by-row distance transform: for each row of the image array, it determines which black point (edge point) of E_t on that row is the closest. It then scans down each column, maintaining a list of the black points of E_t which might be the closest to some point further down this column. This pass determines, for each point in the transform array, the location of the closest black point of E_t lying above it. A similar pass, but scanning up each column, then completes the computation of determining the closest black point of E_t to each point of the array. The output of this process is an array A where each $A[x, y] = (u, v)$, where (u, v) is the closest black point of E_t.

We modify the array A to now contain the translation of each (u, v). That is, each $A[x, y] = T(u, v)$ where $T(u, v)$ is the best translation of the contour C to which the edge point (u, v) belongs. Thus A has been partitioned into regions of the same translation. This can be seen in Figure 1, row three, where each distinct translation is displayed as a unique gray-level.

Given this partition of translation regions, we find the most predominant translation in A (the translation with the greatest number of pixels over the entire array) and call this translation T_p. We then consider all regions with translation T_p to be background and not moving objects. Further, we consider those regions which are both adjacent to some region with translation T_p and have translations similar to T_p to be part of the background. We define a translation to be similar to T_p when it is within 1.4 pixels (i.e., an 8-connected neighbor) of T_p. The background is not considered to be a region for the purpose of merging adjacent regions, as described in the following paragraph.

To merge regions with similar translations, we consider only regions that are adjacent to each other in the following manner. We scan A looking for some region. Upon finding a new region R with translation T, we maintain two sets of translations G and B, where initially $G = \{T\}$ and B is the set of translations of all neighboring regions of R. For each $b \in B$ that is a similar translation to some $g \in G$, b is added to G and removed from B. Region R is expanded to contain those neighbors with the translation b, and the translations of any new neighboring regions of R are added to B. This process continues until the sets G and B remain unchanged. Region R is then reported as a moving object region and removed from the array A. Moving object regions are extracted from A until it is empty.

5 Examples

In this section we present some additional examples of the recovered motion and the resulting motion segmentation. Each example has the same six parts as Figure 1: the top left is the image at time t, the top right is the image at time $t + 1$, the middle left is the edges E_t superimposed on E_{t+1} (with E_t in gray), the middle right is each contour $C \in E_t$ translated to its best location superimposed on E_{t+1}, the bottom left is the partition into motion regions, and the bottom right is the most significant motion region (in all the examples except the final one, this region is simply the motion region with the second largest area).

The example in Figure 3 makes use of the contour splitting discussed in Section 3, because the people walking in the corridor form part of the same connected component with the background. The predominant motion region in this image is one of the two people (as the other person's motion was not significantly different from the background, and was merged into the background motion as described in Section 4).

The example in Figure 4 shows a relatively cluttered scene, in which a person is walking towards the camera and to the left. This scene is very cluttered and has quite large depth differences, resulting in a relatively complicated motion segmentation. The largest motion region in this case primarily overlaps with the walking person.

Another heuristic for detecting moving objects (other than the area of the motion region) is to consider the difference between a given region and its neighboring regions. Figure 5 shows a scene where a box is being tossed between two people. In this case, the motion of the box is very different (down and to the left) from the motion of the rest of the image (primarily to the right). Thus the "most significant" motion region here is taken to be the one with the largest difference with its neighbors. Given motion segmentations such as the ones produced by our method, there are clearly a number of different criteria (including area and distinctiveness of a motion region) that can be used to hypothesize important regions. This is a problem domain that we plan to investigate further, given that the motion detection and segmentation computations work well enough that we can examine a large number of different image sequences.

6 Summary

We have presented a method of estimating the motion for two successive frames of an image sequence, and for segmenting the image based on these motions. The method does not make any assumptions about the nature of the camera motion, or about the smoothness of the image motion. The approach is not based on a local computation of image change, such as the optical flow or the estimation of point motions along edge contours. Rather, a transformation (in this case translation) is recovered that best maps an entire edge contour at time t onto the edges at time $t+1$. A given contour is split into subparts if portions of the transformed contour do not match well (i.e., if the match of part of the transformed contour is not better than for the corresponding part of the untransformed contour). The motions of the edge contours are then used to partition the image into regions, based on the spatial proximity relations defined by the Voronoi diagram (distance transform) of the edges. The largest motion region is taken to be the background motion, and other significant sized motion regions (or regions that have motions significantly different from the background motion) can be used to hypothesize possible locations of moving objects. A major area for further study is the use of these motion regions to track moving objects across multiple frames.

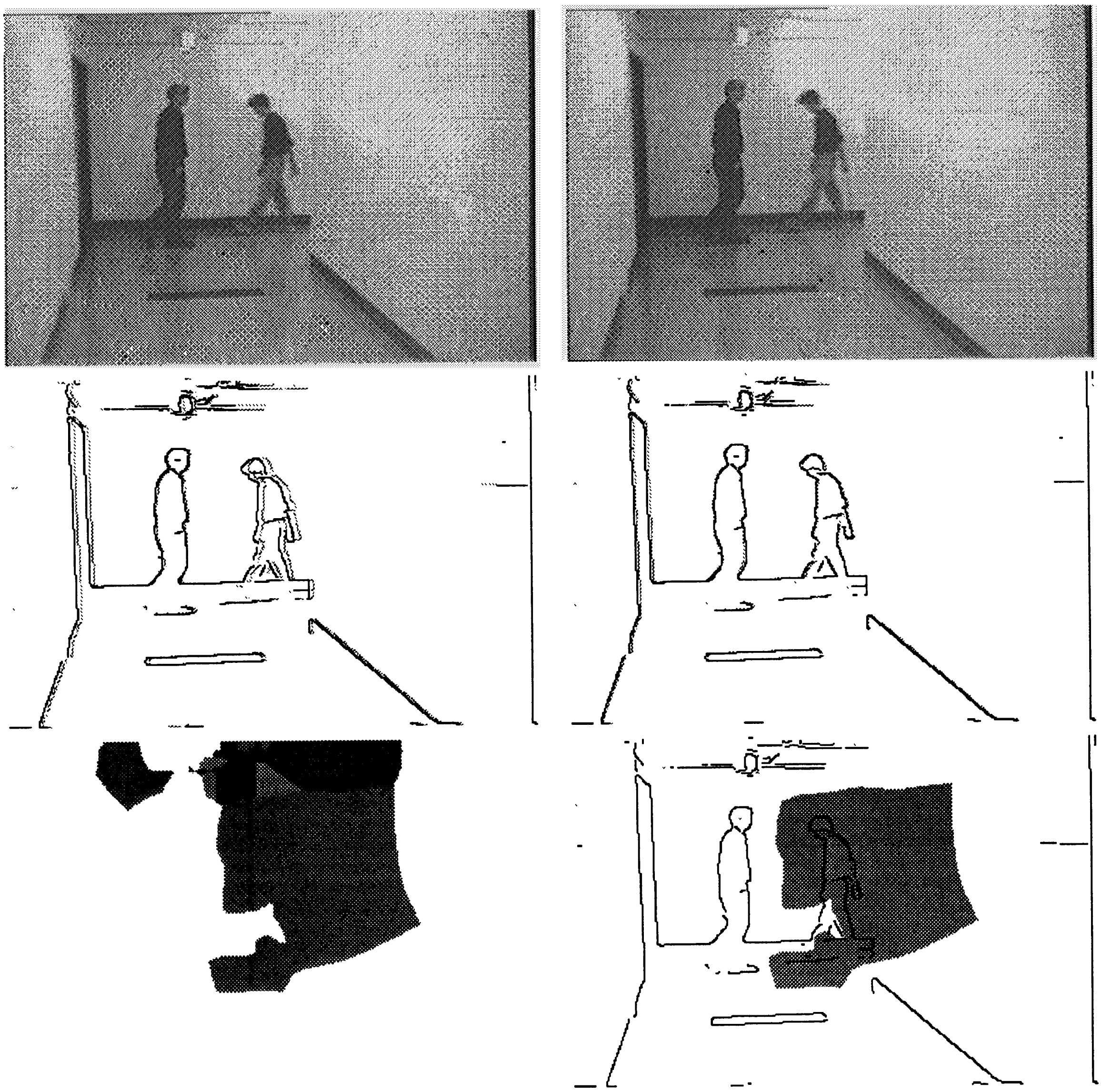

Figure 3: Example motion segmentation (see text).

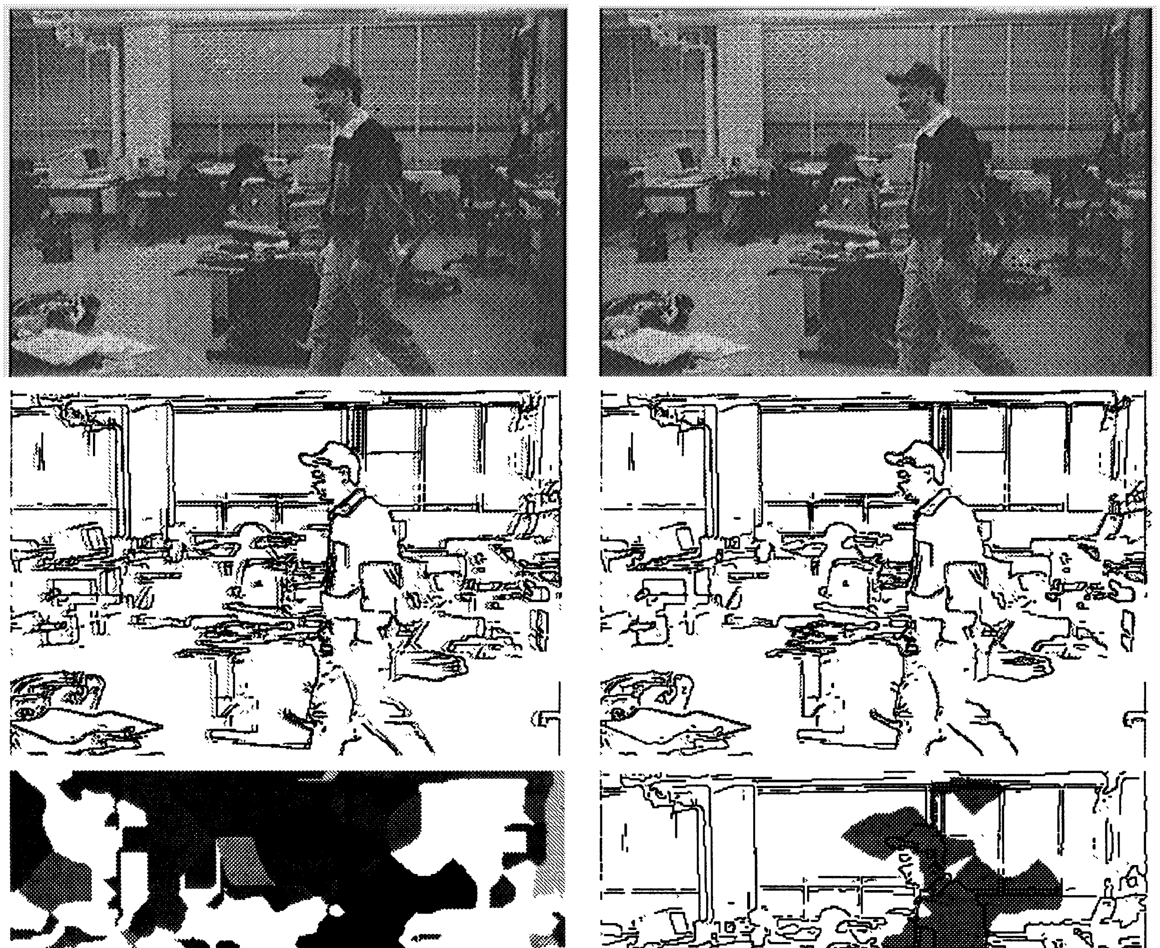

Figure 4: Example motion segmentation (see text).

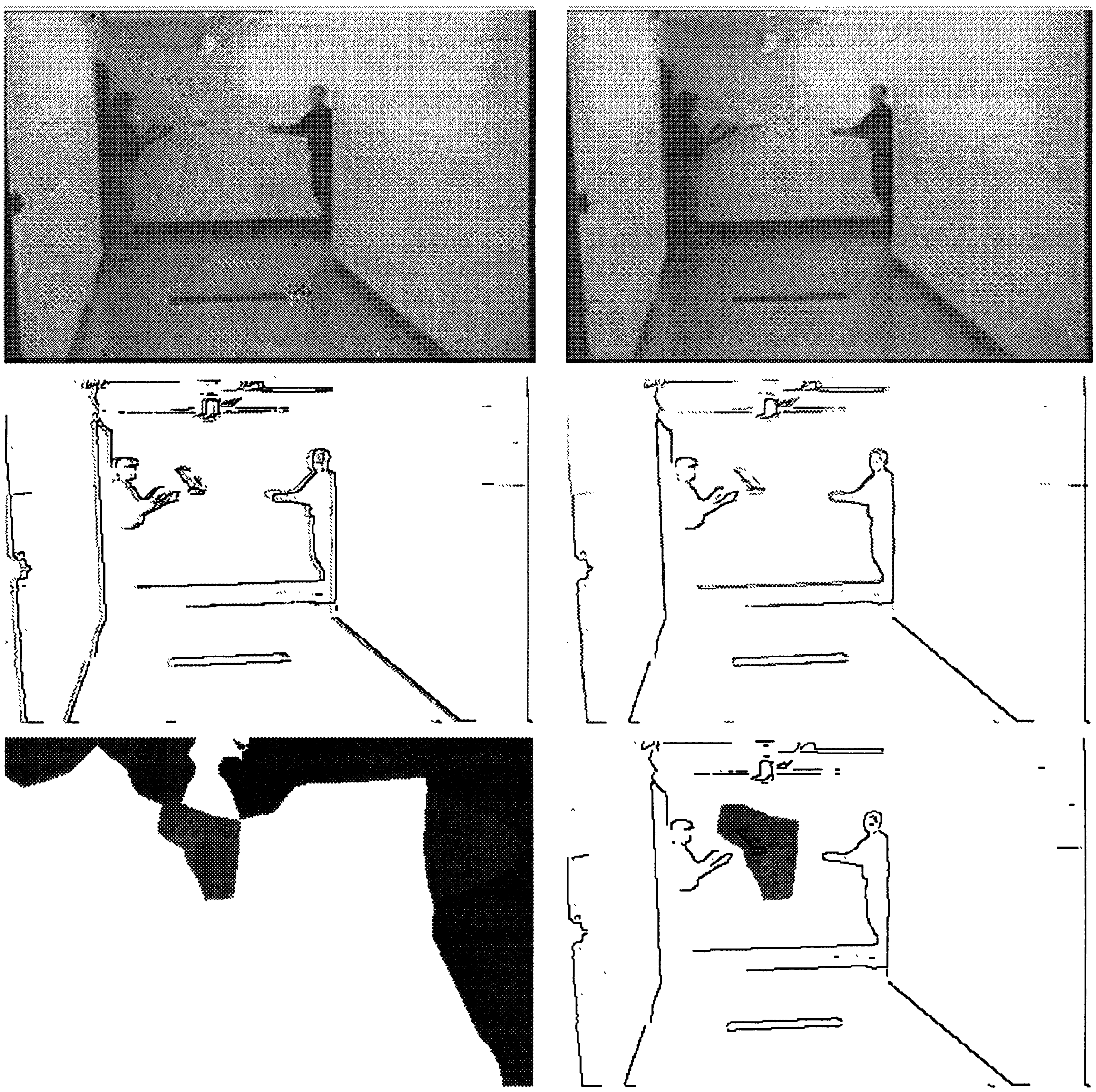

Figure 5: Example motion segmentation (see text).

References

[1] P. Bouthemy and E. Francois. Motion segmentation and qualitative dynamic scene analysis from a motion sequence. *Intl. J. of Computer Vision*, 10(2):157–182, 1993.

[2] P.J. Burt, J.R. Bergen, R. Hingoorani, R. Kolczynski, W.A. Lee, A. Leung, J. Lubin and H. Shvayster. Object tracking with a moving camera. *Workshop on Visual Motion*, 2–12, 1989.

[3] J.F. Canny. A computational approach to edge detection. *IEEE Trans. Pat. Anal. and Mach. Intel.*, 8(6):34–43, 1986.

[4] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pat. Anal. and Mach. Intel.*, 15(9):850–863, 1993.

[5] D.W. Murray and B.F. Buxton. Scene segmentation from visual motion using global optimization. *IEEE Trans. Pat. Anal. and Mach. Intel.*, 9(2):220–228, 1987.

[6] D.W. Paglieroni. Distance transforms: Properties and machine vision applications. *Computer Vision, Graphics and Image Proc.: Graphical Models and Image Processing*, 54(1):56–74, 1992.

[7] S. Peleg and H. Rom. Motion based segmentation. *Proc. Intl. Conf. on Pattern Recog.*, 109–133, 1990.

[8] W.B. Thompson and T.G. Pong. Detecting moving objects. *Intl. J. of Computer Vision*, 4:39–57, 1990.

[9] J. Woodfill and R.D. Zabih. An algorithm for real-time tracking of non-rigid objects. *Proc. American Association for Artificial Intelligence Conference*, 1991.

[10] S. Sull and N. Ahuja. Segmentation, matching and estimation of structure and motion of textured piecewise planar surfaces. *Workshop on Visual Motion*, 274–279, 1991.