7//37

Stereo-Based Region-Growing using String Matching [*]

Robert Mandelbaum and Max Mintz

General Robotics and Active Sensory Perception (GRASP) Laboratory
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

October 7, 1995

Abstract

We present a novel stereo algorithm based on a coarse texture segmentation preprocessing phase. Matching is performed using string comparison. Matching substrings correspond to matching sequences of textures. Inter-scanline clustering of matching substrings yields regions of matching texture. The shapes of these regions yield information concerning objects' height, width and azimuthal position relative to the camera pair. Hence, rather than the standard dense depth map, the output of this algorithm is a segmentation of objects in the scene. Such a format is useful for integration of stereo with other sensor modalities on a mobile robotic platform. It is also useful for *localization*: height and width of a detected object may be used for *landmark recognition*, while depth and relative azimuthal location determine *pose*.

The algorithm does not rely on the monotonicity of order of image primitives. Occlusions, exposures, and foreshortening effects are not problematic. The algorithm can deal with certain types of transparencies. It is computationally efficient and very amenable to parallel implementation. Further, the epipolar constraints may be relaxed to some small but significant degree. A version of the algorithm has been implemented and tested on various types of images. It performs best on random dot stereograms, on images with easily filtered backgrounds (as in synthetic images), and on real scenes with uncontrived backgrounds.

1 Introduction

1.1 Motivation

A common deficiency among standard stereo algorithms is that they do not provide a *segmented* representation of the scene, with each region corresponding to a distinct object in the scene. This reduces the potential usefulness of such algorithms within the context of a mobile robotic system. A common assumption is that if a mobile robot is to use a stereo system as a sensing modality, it

[*]Portions of this research were supported by the following grants and contracts: ARPA Contracts N00014-92-J-1647, and DAAH04-93-G-0419; ARO Contracts DAAL03-89-C-0031PRI, and DAAL03-92-G0153; NSF Grants CISE/CDA-88-22719, IRI92-10030, IRI92-09880, IRI93-03980, and IRI93-07126.

is the task of other modules further along in the dataflow pipeline to segment the output (usually a dense depth-map) and extract from it whatever information is required by the system.

In contrast, we designed the stereo algorithm described in this paper with the *specific intention* of using the output on a mobile robot to aid in localization of the agent. Furthermore, we wished to use stereo in conjunction with other sensor modalities. In essence, we addressed the design of a stereo processing technique by (a) deciding what type of output would be most useful for the task of localization, and (b) taking into consideration how the modality would be integrated into the system as a whole.

For these reasons, we stressed the following attributes:

1. *Computational efficiency.*

2. *Predictive power:* If the data representation allows prediction of expected sensor measurements, then correspondence matching between extracted and stored features is facilitated. This supports localization. The dense depth-map output by standard stereo algorithms does *not* facilitate this prediction.

3. *Robustness:* The stereo algorithm should work well on real, uncontrived indoor images with ordinary high-textured backgrounds.

This paper describes a stereo algorithm which is computationally highly efficient, performs well on real scenes with highly textured backgrounds, and produces output in a form which is very compatible with other sensor modalities. In particular, the output of height, width, azimuthal location and range of an object in the scene is very useful for landmark recognition and localization.

1.2 Overview

In [5] it is pointed out that there are three major components of any stereo algorithm: preprocessing, matching and 3-D structure determination. This is based on the assumption that the data-flow consists of (i) an input of two or more images of the same scene from different vantage points, and (ii) an output of a dense depth-map of the scene. Algorithms differ in the type of preprocessing performed, and hence in the nature of the primitives upon with the matching is executed, in the type of matching, as well as in the post-matching 3-D reconstruction.

Many trade-offs are made. Several algorithms extract *features* from the images. Examples of typical features are single-edge points (often extracted by finding the zero-crossings of the convolution of the image with the $L \circ G$ operator)[1] [2, 7, 15], and linear edge segments extracted using some edge detector [1, 13]. Feature extraction reduces the number and ambiguity of the primitives to be matched, thus reducing the complexity of the matching problem. Feature-based methods also lead potentially to great accuracy since features can be located in each image to sub-pixel precision. On the other hand, feature extraction can itself be computationally expensive. Furthermore, the product of matching features is a sparse depth-map, which must then be interpolated. Not only can the interpolation be a difficult and computationally expensive task, but it can also lead to ambiguities in reconstruction and the blurring of the very edges which were used for matching.

Area-based correlation methods compare windows surrounding points in both images. Various metrics are used to evaluate how well the windows are correlated [6, 8, 14]; the windows around points on epipolar lines with greatest correlation values are deemed to match. While such an

[1] where L is some discretized form of the Laplacian (second derivative) and G is a Gaussian smoothing operator.

approach leads to a dense disparity map (in theory, *all* non-occluded pixels have associated disparities), the correlation process is very computationally intensive. In effect, the set of primitives is the set of all windows. A large set of primitives (i) necessitates a large amount of matching, and (ii) leads to frequent occurrences of ambiguities. A trade-off exists in selecting the window size: the window size must be large enough to include enough intensity variation for reliable matching, but small enough to avoid the effects of projective distortion [9]. Also, the larger the window size, the greater the computational expense. For $n \times n$ images and a window size of $w \times w$, the naive correlation-based stereo algorithm has complexity $O(n^3w^2)$. An effective adaptive windowing technique is discussed in [9]. Being pixel-based, the precision of area-based correlation methods is limited to pixel-sized discretization [3].

There are several algorithms which are pixel-based and yet do not rely on windowing techniques [4]. This allows them both to yield a dense disparity map and to be faster since no features need be extracted, nor are window comparisons necessary. Indeed, the algorithm described in this paper is of this type.

In general, the greater the distinctiveness of the features extracted, the less matching is required, but also the more time has to be spent in feature extraction, and the sparser the resultant disparity map.

In many streo algorithms, assumptions are made regarding the nature of the scene:

1. Many algorithms rely on the *monotonicity* assumption, i.e. that the ordering of primitives along epipolar lines is the same in both images. In fact, not only does this assumption not always hold, but it is violated in cases where the effects of taking two different views of a scene are greatest; it would seem that an algorithm exploiting stereo effects should *utilize* rather than *avoid* these cases.

2. In the interests of computational efficiency, many stereo algorithms limit the search for matches to small disparities. This is, in effect, assuming that all objects of interest are far enough away that disparities will not be large. Once again, it would seem such algorithms are avoiding the very effects upon which stereo is based. By limiting themselves to small disparities, such algorithms constrain the area of interest to relatively great depths, where errors are greatest [12, 3].

3. Most stereo algorithms have difficulty dealing with *occlusions* and *exposures*. In actual fact, *any* object which stands out from the background and thus differs from its background in disparity will cause part of that background to be occluded in one of the images. Moreover, any occlusion in one image corresponds to an exposure in the other. Once again, it would seem that occlusion and exposures are necessary and expected artifacts of stereopsis, and should be manageable, if not exploited, by stereo algorithms.

4. Many stereo algorithms assume a frontal planar nature to detected surfaces. This arises from the assumption of orthogonal rather than perspective projection, combined with a parallel-axis stereo geometry: under these conditions, the projections of a surface onto the two image planes are identical, regardless of the orientation of the surface; therefore, upon reconstruction, nothing more complex than frontal planar surfaces is justified. In reality, some information may be gleaned from the effects of foreshortening and the use of perspective projection in stereo.

5. During 3D reconstruction, some algorithms interpolate a dense depth-map from a sparse disparity map. Many such algorithms assume a continuous underlying "rubber" surface,

which has been stretched to pass through the detected depth points. Such a reconstruction model blurs and smooths the edges between objects in the scene; rather than facilitating the segmentation of the scene, clustering of points of similar depth into "objects" is made more difficult.

While these assumptions may indeed be valid in certain environments, the invocation of these assumptions detracts from the versatility and applicability of an algorithm. Moreover, it seems the previous assumptions all arise from a *single* underlying requirement: that the product of a stereo system should be a dense depth-map. It is assumed that if a system is to use a stereo system as a sensing modality, it is the task of other modules further along in the dataflow pipeline to segment this dense depth-map and extract from it whatever information is required by the system.

In this paper we present a stereo algorithm whose output is *not* a dense depth-map of the scene. In fact, the stereo modality is not used to yield depth information at all. In this work, we move in the opposite direction to the standard data-flow. Our stereo algorithm makes use of depth information acquired from ultrasound sensors to guide the search for correspondences. The output of our algorithm is a *partial segmentation* of the scene: at the very least, the algorithm yields information regarding the extents and azimuthal position in the scene of the particular "segment" (i.e. object) which reflected the ultrasonic energy. We believe the output of clustered, segmented data to be a novel aspect of stereo algorithm design. Such a format has proved to be very useful for the integration of stereo with other sensor modalities, especially for the purposes of landmark recognition and localization.

Several other attributes of the stereo algorithm presented in this paper are:

1. The *monotonicity* constraint is not inherently necessary: a string-matching algorithm which handles transposition can be used. However, in our current implementation, we do not make use of this capability.
2. Highly textured, or easily filtered backgrounds (as in synthetic images) are preferred.
3. Since the output of the algorithm is *not* a dense depth-map, occlusions and exposures do not, in general, hinder the operation of the algorithm.
4. Foreshortening effects can be detected by the algorithm, and may, therefore, be exploited for the reconstruction of surfaces at a non-zero angle to the image plane. We do not look for forshortening effects in our implementation.
5. The algorithm can deal with certain types of transparencies.
6. The algorithm is computationally efficient.
7. The algorithm is very amenable to parallel implementation.
8. Preliminary experimentation has shown the approach to be qualitatively robust to slight relaxation of the epipolar constraints.

1.3 Basic operation

The basic operation of the algorithm is described in detail in Section 2. Like most stereo algorithms, it is divided into three stages:

1. In the pre-processing phase, each image is segmented very coarsely according to texture. Texture-segmentation here includes color-segmentation and intensity-based segmentation, among others. Each texture region is then labeled with a letter from the alphabet of possible textures $\mathcal{T}$.

2. In the matching phase, the two *strings* of texture labels associated with epipolar scanlines in the two images are compared. Matching substrings are extracted. Each substring corresponds to a sequence of texture labels, regardless of whether members of that sequence have been foreshortened or not. In fact, once matches have been established, the pixel widths of corresponding texture regions may be compared; a change in region width indicates either foreshortening or occlusion. Some higher-level reasoning system may be used to disambiguate the two cases.

   A string matching approach has several advantages: By the nature of string matching, the uniqueness constraint[2] is propagated automatically. If long substrings are searched for first, cohesivity is stressed, possibly at the expense of the number of total matches. Occlusions and exposures in the scene correspond to string deletions and insertions respectively, and cause no problems. Non-monotonicity of order of image primitives corresponds to substring transposition, and is manageable by most sophisticated string matching algorithms. Any of a host of new string matching algorithms developed for use with genetic data may be called upon for the efficient execution of this phase. Finally, since each scanline is processed independently, this phase is amenable to parallel implementation.

3. In the 3D reconstruction phase, we cluster matching substrings over multiple adjacent scanlines. Substrings beginning or ending at approximately the same horizontal location in the image plane over multiple scanlines are clustered together.In this way, objects consisting of similar texture patterns are segmented. Since we are interested in properties of the multi-scanline *segment* and not each *scanline*, the algorithm allows for a certain amount of relaxation of the epipolar constraint. Misalligned scanlines will simply result in the top or bottom of a region being in error.

1.4 Domain of applicability

The algorithm described in this paper has been implemented and tested on various types of images including:

- random dot stereograms,
- synthetic "blocks world" images with controlled backgrounds,
- real images of indoor scenes with controlled (low texture) backgrounds, and
- real images of indoor scenes with uncontrived (highly textured) backgrounds.

Several of the image pairs and the resultant output of the algorithm are shown in Section 3. In general, since the algorithm treats the background in exactly the same way as the foreground objects, it performs better on more highly textured backgrounds. For cases where the background is less textured than the foreground objects of concern, and where foreground objects are sparse,

[2]The uniqueness constraint states that each primitive in the left image can be matched to only one primitive in the right image.

most of the matching is performed on the background, and the foreground objects do not stand out in the resulting set of matches. Matches on a low-texture background do not generally yield accurate disparity estimates since there are no texture changes on which to fixate; background pixels match background pixels with many different disparities. Note that low texture causes problems only for the *background*; low-textured foreground objects will still result in reliable string matches.

For this reason, the algorithm seems to perform best on random dot stereograms, on synthetic images for which the background can be filtered out, and on real scenes with uncontrived backgrounds. See Section 3 for examples. We are currently investigating the application of this algorithm to mobile robot localization.

1.5 Related work in stereo

Reference [5] presents a comprehensive survey of recent developments in establishing stereo correspondence for the extraction of the 3D structure of a scene. In particular, we mention here the work of Lim and Binford [10], Ohta and Kanade [15], and Cox et al [4], with which our work shares some similarities.

In [10], preprocessing of each image consists of edgel (edge elements) detection. Edges are linked into connected edges and curves. Surfaces are identified by boundary-tracing, and bodies are identified as groups of surfaces that share edges. Matching is attempted at the highest level. Results of matching are propagated to each successive lower level (surfaces, curves, edgels) [5]. As is pointed out in [5], "the advantage of this hierarchical stereo system is that the depth-map obtained is already segmented and ready for surface interpolation." The algorithm described in this paper shares this desireable property. One of the differences, however, is that in this work, stereo matching is used in the region-growing phase.

In [15], the search for matches "is formulated as a path-finding problem in a 2D search space in which vertical and horizontal axes are the right and left scanlines, respectively" [5]. Dynamic programming is used to find the path which minimizes a cost function "based upon variances of gray-level intensities of the scanline intervals being matched." The results of the intrascanline search "are used to establish global consistency among matches achieved in neighboring scanlines using an *interscanline* search." Edge connectivity is used to impose a consistency constraint. Dynamic programming is also used in [4], though in that work a *maximum likelihood* cost function is optimized. Certain assumptions are made about underlying probability distributions. Several *cohesivity constraints* are imposed to guarantee "that solutions minimize the original cost function and preserve discontinuities" [4]. "The constraints are based on minimizing the total number of horizontal and/or vertical discontinuities along and/or between adjacent epipolar lines, and local smoothing is avoided."

There are several correspondences between the work described in this paper and those in [4] and [15]:

1. String matching is similar to the dynamic programming approach. A fundamental difference, however, is that for string matching, no cost or regularization function need be defined; rather, specific behavior can be guaranteed by the appropriate selection of string matching criteria. As an example, in [4], it is shown that more accurate results are obtained if the sum of horizontal discontinuities is minimized. The cost function is then modified to incorporate this criterion. Using string matching, this cohesivity constraint may be satisfied more directly by simply searching for long substring matches first; in effect, a set of matches

involving m_a *adjacent* pixels is deemed superior to a set of $m_{\rm nc}$ non-contiguous matches, even if $m_{\rm nc} > m_a$.

2. Differences exist in the approach taken for *inter-scanline* clustering. In [15] the problem is posed as that of finding the least-cost path in a 3-D search space [5]. In [4], the problem is formulated as that of minimizing the sum of horizontal *and vertical* discontinuities. Since "minimizing vertical discontinuities between epipolar lines cannot be performed by dynamic programming" [4], an approximation is obtained by using a relaxation technique to "minimize the *local* discontinuities between adjacent epipolar lines." In our implementation, matching substrings over multiple scanlines are compared; those with similar starting or ending locations in the image plane are deemed to belong to the same cluster.

Though the algorithm described in this paper is pixel-based, it differs from area-based correlation or sum of squared differeces (SSD) approaches such as those in [6, 8, 9, 14] in that no windowing is used. Similarly, though the preprocessing phase of coarse texture segmentation may be seen as a form of feature detection, this approach does not really fall within the genre of *feature*-based stereo algorithms such as those in [1, 2, 7, 13, 15], since we are using *strings* of these features for matching.

2 Stereo based on string matching

Our approach for extracting segmentation information using stereopsis comprises three phases: texture segmentation, string matching and inter-scanline clustering.

2.1 Texture segmentation

In the pre-processing phase, each image is segmented very coarsely according to texture. The purpose of the stereo algorithm is to "grow" these coarse segments into regions of similar texture *patterns* in both images. Possible types of texture-segmentation here include color-segmentation and intensity-based segmentation, among others. Each texture patch is then classified and labeled with a letter from the alphabet of possible textures $\mathcal{T}$. Ideally, each texture segment would correspond to a part of an object in a scene. Since foreshortening of texture patches is expected, ideally texture classification should also be invariant under scaling in the horizontal direction. Similarly, the texture classification should be invariant under small changes in illumination. See Figure 1 for an example image and the desired type of segmentation.

Let P be a texture-based segmentation operator which partitions an image into patches according to texture. Let T denote a texture classifier assigning one of $|\mathcal{T}|$ labels to each patch. Let R be a binary relation over elements of $\mathcal{T}$, $R \subseteq \mathcal{T} \times \mathcal{T}$, such that for any textures t_i, $t_j \in \mathcal{T}$, $(t_i, t_j) \in R$ implies that texture t_i is similar to texture t_j. In other words, t_i, $t_j \in \mathcal{T}$, $(t_i, t_j) \in R$ implies that a patch of texture t_i in one image may be considered to correspond to a patch of texture t_j in the other image.

Hence, an implementation of this phase of the algorithm consists of

- a texture segmentation algorithm $T \circ P$ capable of reliably segmenting a scene into texture patches and classifying each patch into one of $\mathcal{T}$ textures, and

- a similarity relation R defined below.

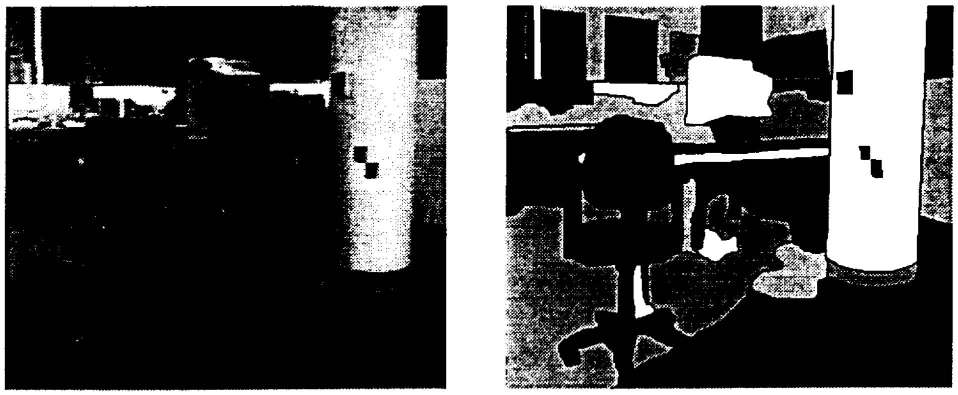


Figure 1: (Left) Example of scene to be analyzed using stereopsis. (Right) Idealized example of the type of texture segmentation suitable for stereo analysis by string matching (as described in this paper). This segmentation was obtained manually.

Since efficient region-growing based on texture is still an open problem, in our implementation we simplify the texture-segmentation phase and rely on the string-matching phase to "grow" the texture segments. We consider each pixel to be its own texture patch, and we discretize the only information we have about the pixel's "texture", i.e. its intensity, into $|\mathcal{T}|$ levels. Each texture patch (i.e. each pixel in our implementation) is therefore assigned one of $|\mathcal{T}|$ labels. Represent these labels with the first $|\mathcal{T}|$ letters of the alphabet. In our implementation, $|\mathcal{T}| = 16$. The coarseness of the discretization makes the texture classification robust in the face of small foreshortening and illumination effects. Though this is a simple "segmentation" requiring very little computation, the trade-off is that, for $n \times n$ images, each scanline consists of long strings of n texture patches, making the string matching phase more computationally intensive. The stereo algorithm is not inherently pixel-based, and the use of any reliable texture-segmentation algorithm yielding texture patches larger than a single pixel would enhance the performance of the string matching greatly. The specification of such a texture-segmenter is beyond the scope of this work.

Due to the simplicity of the texture space we selected for our implementation, the similarity relation R we chose is also simple: each texture is considered to match its immediate neighboring textures in the alphabet, and to be dissimilar to all other textures. A binary matrix representing R is shown in Figure 2.

If color images are available, a possible "texture" classification would consist of the discretization of 3-dimensional (Red$\times$Green$\times$Blue) color space into appropriately sized bins. A possible similarity relation R would then be

$$R = \{(t_i, t_j) \ : \ t_i \text{ is 26-adjacent to } t_j,\ t_i, t_j \in \mathcal{T}\} \ \cup \ \{(t_i, t_i) \ : \ t_i \in \mathcal{T}\}$$

The crucial attribute of any such $(T \circ P) - R$ combination is that T discretizes the space of textures sufficiently finely to distinguish textures, and yet R is sufficiently rich to allow matches among non-identical texture patches. This is necessary since noise in each image, foreshortening and illumination effects often result in the same surface in the scene being classified differently in the two images. In other words, if p_l and p_r are corresponding texture patches in the left and right images respectively, we do not insist that $T(p_l) = T(p_r)$, but merely that $(T(p_l), T(p_r)) \in R$. This is a much easier criterion to meet since R may be designed with knowledge of the behavior of $T \circ P$ and the types of scenes involved. Indeed, R may be changed for various scene and illumination types.

R	a	b	c	d	e	f	$\cdots$
a	1	1	0	0	0	0	
b	1	1	1	0	0	0	
c	0	1	1	1	0	0	
d	0	0	1	1	1	0	
e	0	0	0	1	1	1	
f	0	0	0	0	1	1	
$\vdots$							$\ddots$

Figure 2: Simple similarity relation R between elements of the texture alphabet $\mathcal{T}$. A "1" in position (i, j) indicates that the texture designated by the ith character in the texture alphabet $\mathcal{T}$ is deemed "similar" to the texture designated by the jth character. In this instantiation of R, all texture are considered similar only to themselves and to their immediate neighbors in the alphabet.

Area-based correlation approaches to stereo may be thought of as consisting of a classifier T which assigns a texture label to each pixel based on the surrounding window. Matches are determined by finding the *closest* point in texture space on the corresponding epipolar scanline. This relies on the underlying assumption that texture is numerically quantifiable, and that the distance metric used over texture space imposes a topology which adequately reflects reality. In our approach, the use of the binary relation R as the metric results in a very different topology. In effect the metric is of the $0 - 1$ type, and two textures are considered either to match or not. The relation R gives us much greater control over the topological neighborhoods of texture space, and hence over the model of reality it reflects.

2.2 String matching for each scan-line

In the matching phase, the two *strings* of texture labels associated with epipolar scan-lines in the two images are compared. Matching sequences of texture patches are extracted, even if some of those patches have undergone foreshortening; a difference in patch widths of corresponding textures indicates either occlusion or foreshortening.

The field of efficient string-matching alogrithms has received extensive attention in recent years in the context of genetics and the human genome project. Any of a host of new algorithms may be utilized for this phase of the stereo algorithm. Desireable properties of a selected string-matching algorithm are:

- Efficient handling of deletions and insertions. These correspond to occlusions and exposures in the image scanlines respectively.

- Efficient handling of substring transpositions. This corresponds to non-monotonicity in the order of matching primitives in the scene.

In our implementation, we use an algorithm which does not explicitly look for transpositions, but does produce substring matches in a form which is easily amenable to a search for transpositions. The string matching algorithm begins by looking for long substring matches, and if no such matches are found, progressively shorter and shorter substring matches are searched for. In this way, cohesivity rather than high pixel-match count is stressed.

Let `left` and `right` represent corresponding epipolar strings of texture labels in the left and right images respectively. Let the length of `left` and `right` be n. Let $\mathtt{string}_i^m$ denote the substring of string beginning at the ith location and ending at the mth. Denote the kth *character* of string by string[k]. Note that $\texttt{string}_i^n[k]$ denotes the same character as $\texttt{string}_1^n[i+k-1]$.

If a matching substring is found, say between the first p characters of $\texttt{left}_i^n$ and $\texttt{right}_j^n$, then the algorithm recursively searches for matches of length smaller than p between the substrings $\texttt{left}_1^{i-1}$ and $\texttt{right}_1^{j-1}$, as well as for matches of length p and smaller between substrings $\texttt{left}_{i+p}^n$ and $\texttt{right}_{j+p}^n$. The algorithm used to compare two strings left and right in our implementation is shown in Figure 3.

The output of the algorithm consists of the two strings with various substrings designated as matches. Transpositions may be taken into account in a second pass: matches may searched for among unmatched portions of left and right which have not yet been compared.

If a naive approach is used for the string comparison function string_match, the complexity of algorithm in Figure 3, per scanline, ranges from $O(n)$ in the best case (complete match) to $O(n^3)$ in the worst case (no matches found). In our implementation, we employ a dynamic programming approach in order to expedite the string_match function in the worst case scenario.

Let M be a $n \times n$ matrix such that $M_{i,j}$ contains the length of the longest common prefix of $\texttt{left}_i^n$ and $\texttt{right}_j^n$. In other words, for R as in Figure 2,

$$\begin{array}{rl} \forall_{1\le k\le M_{i,j}} & \left(\texttt{left}_i^n[k],\ \texttt{right}_j^n[k]\right) \in R \\ \text{and} & \left(\texttt{left}_i^n[M_{i,j}+1],\ \texttt{right}_j^n[M_{i,j}+1]\right) \notin R \end{array}$$

As an example, consider the strings left = 'aaddbbe' and right = 'gcdbbdd'. Then $\texttt{left}_3^7 =$ 'ddbbe', and $\texttt{right}_2^7 =$ 'cdbbdd'. Since $(d,\, c) \in R$ and $(e,\, d) \in R$, the longest matching prefix of these substrings has length 5. Hence $M_{3,2} = 5$. For this example,

$$M = \begin{bmatrix} 0 & 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 5 & 1 & 0 & 0 & 2 & 1 \\ 0 & 1 & 4 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 3 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Note that if $M_{i,j} > 0$, then $M_{i+1,j+1} = M_{i,j} - 1$. It is this property of M that we exploit to expedite the search for substring matches. If the matrix M for two strings left and right was known *a priori*, then the comparison of two p-length substrings of left and right would consist of a simple table look-up rather than the $O(p)$ character-by-character comparison naive approach; the overall complexity of a scanline match would drop to $O(n^2)$ in the worst case, a significant improvement. Unfortunately, the matrix M is *not* known a priori. In our implementation, however, we partially construct M as string matching progresses.

We begin by setting all values of M to -1. When searching for a substring match of length p between $\texttt{left}_i^{i+q}$ and $\texttt{right}_j^{j+r}$, where $q, r \ge p$, we consult the current value of $M_{i,j}$. If this is still -1, then these substrings have not been compared before. We therefore compare the substrings and find the true value of $M_{i,j}$. $M_{i,j}$ cannot be larger than p since if it was, then we would have found a match while searching for longer substring matches. If $M_{i,j} = p$, then we have located a match. If $M_{i,j} < p$, we update M as follows: for each value of $0 \le k \le p$, set $M_{i+k,j+k} = M_{i,j} - k$.

If upon consulting the current value of $M_{i,j}$, we find that it is *not* equal to -1, then these substrings have been compared before; there is no need to compare them again. Hence, though the update of M requires $O(M_{i,j})$ operations, it potentially saves $O(M_{i,j}^2)$ character comparisons

```
match_strings(left, right, left_length, right_length)
   max_length = min(left_length, right_length);
   if (max_length == 0) {
      /* Base case */
      record_match(left, right, 0);
      return (0);
      /* 0 matches found */ }
   else {
      p = max_length;
      while (p > minimum_match_size) {
         i = 0;
         while (i <= left_length - p) {
            j = 0;
            while (j <= right_length - p) {
               if (string_match(left$_i^{i+p}$, right$_j^{j+p}$, p) == TRUE) {

                  /* Recursively search for matches between left$_1^{i-1}$ and right$_1^{j-1}$ */
                  pre_match = match_strings(left$_1^{i-1}$, right$_1^{j-1}$, i-1, j-1);

                  record_match(left$_i^{i+p}$, right$_j^{j+p}$, p);

                  /* Recursively search for matches between left$_{i+p}^n$ and right$_{j+p}^n$ */
                  post_match = match_strings(left$_{i+p}^n$, right$_{j+p}^n$, n-i-p+1, n-j-p+1);

                  return (pre_match + 1 + post_match);
                  /* Number of matches found */ }

               j = j+1; }
            i = i+1; }
         p = p-1; } }
```

Figure 3: The algorithm used to compare two strings `left` and `right` in our implementation. The ranges over which i, j and p vary may be changed to take into account minimum possible disparities between the left and right images.

```
string_match(left$_i^{i+q}$, right$_j^{j+r}$, length)
    if ($M_{i,j}$ > -1) {
        /* We have already tried a match starting at these points before */
        if ($M_{i,j} \geq$ length) return (TRUE);
        else                 return (FALSE); }
    else {
        /* Must do the comparison. If successful, there is no need to update
            $M$ since we will not be accessing these portions of the strings again */
        count = 0;
        while (count < length AND (left$_i^{i+q}$[count+1], right$_j^{j+r}$[count+1]) $\in R$) {
            count = count + 1; }
        if (count == length) return (TRUE);
        else {
            /* Update $M$ */
            m = 0;
            while (m $\leq$ count) {
                $M_{i+m,j+m}$ = count - m;
                m = m+1; }
            return (FALSE); } }
```

Figure 4: An efficient algorithm for the string_match function which checks whether strings left$_i^{i+q}$ and right$_j^{j+r}$ share a common prefix of length length.

in subsequent (shorter) substring comparisons. Efficiency is improved significantly. Hence, an efficient algorithm for the string_match function is shown in Figure 4.

2.3 Inter-scanline clustering

Once matching substrings, and their associated disparities, have been found, a 3D reconstruction of parts of the scene may be performed: each substring match corresponds to a horizontal strip of an object in space. By analyzing the distortion in pixel width of each "character" (i.e. texture patch) in the match, foreshortening or partial occlusion may be inferred. In our implementation, however, since each primitive texture patch is a single pixel, foreshortening is more difficult to detect.

In our implementation, we choose to cluster matching substrings over multiple scanlines by examining their disparities, as well as their beginning and ending points: substrings with similar disparities and beginning or ending at approximately the same horizontal location in the image plane over multiple scanlines are clustered together. The output is a polygon in the image plane circumscribing all contributing substrings. The polygon is assigned a disparity corresponding to the average disparity of the contributing substrings. Thus, each polygon corresponds to a region of corresponding texture *patterns* of similar disparity in the left and right images. For calibrated cameras, the polygon may be projected back into 3D space, and corresponds to a section of a frontal planar surface. This approach blurs information concerning disparity variation *within* each region. However, for our purposes — the extraction of objects' extents and azimuthal position within the scene — such information is not necessary.

One advantage of this approach to inter-scanline clustering is that it can handle certain types of transparencies. Consider, for example, a scene comprising an object behind a Venetian blind. Each scanline either contains the object, or one of the blades of the Venetian blind. In either case, matching substrings are extracted, though the set of disparities will be easily partitioned into two subsets: those corresponding to the depth of the object, and those corresponding to the depth of the Venetian blind. The output of the algorithm in this case will consist of two polygons: one circumscribing the object, the other outlining the Venetian blind. Rotation of the Venetian blind by 90 degrees will cause a completely different effect: Each blade of the Venetian blind will now be segemented into its own region, and the object will similarly be dissected into "vertical strips" See Section 3.2 and Figure 7 for experiments involving synthetic images of Venetian blind scenes.

3 Experiments

The algorithm described in this paper has been implemented and tested on various types of images including:

- random dot stereograms,
- synthetic "blocks world type" images with controlled backgrounds,
- real images of indoor scenes with controlled (low texture) backgrounds, and
- real images of indoor scenes with non-contrived (highly textured) backgrounds.

In general, since the algorithm treats the background in exactly the same way as the foreground objects, it performs better on more highly textured backgrounds. For cases where the background is less textured than the foreground objects of concern, and where foreground objects are sparse, most of the matching is performed on the background, and the foreground objects do not stand out in the resulting set of matches. Matches on a low-texture background do not generally yield accurate disparity estimates since there are no texture changes on which to fixate ; background matches background with many different disparities. For this reason, random dot stereograms, synthetic images for which the background can be filtered out, and real scenes with uncontrived backgrounds are most suitable for application of our algorithm.

3.1 Random dot stereograms

In order to test the correctness of the stereo matching algorithm, the algorithm was tested on a random dot stereogram pair. Figure 5 shows the output generated. The output is in the form of a polygon circumscribing the region deemed to match. Note the jagged vertical edges on both sides of the polygon. Since the algorithm detects matches by way of string comparison, "extra" pixels on either end of the "genuine" shifted string will often be included in the match: there is a 50% chance of a single random pixel matching, a 25% chance of two consecutive pixels matching, etc. Once these random artifacts of random dot stereograms are taken into account, the algorithm is seen to perform flawlessly, correctly matching 100% of the shifted pixels.

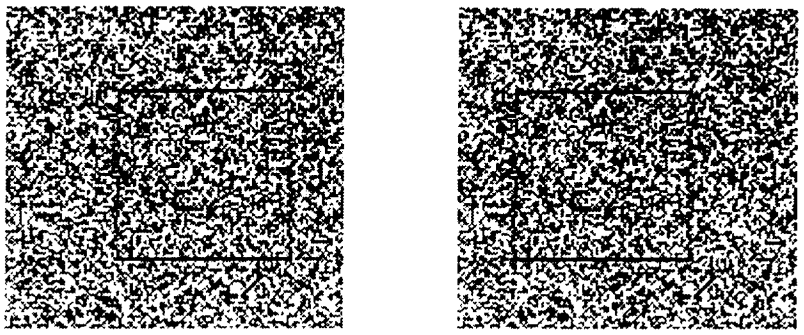


Figure 5: A random dot stereogram pair and the output of the algorithm in the form of a polygon circumscribing the matching region. The jagged vertical edges are a result of random "extra" matching pixels adjacent to the shifted region. 100% of actual shifted pixels are identified. Processing time for these 125 × 120 images, including the clustering into the polygon, is approximately 20 seconds on a Sparc 10.

3.2 Synthetic images

Figure 6 shows a pair of blocks world images of an assortment of objects. Directly below the images is the output of the algorithm where no distinction has been drawn between background and objects; the longest matching strings therefore comprise mostly background, and swamp the output. The next image illustrates the output once the algorithm has been instructed to filter out the dark background and black "floor". Regions correspond very well to the objects. The last pair of images shows the polygons thus found superimposed on the original images to show how the regions match the underlying images.

Figure 7 illustrates the ability of the stereo algorithm to handle certain restricted types of transparency. The top figure shows a rectangular block behind a horizontal Venetian blind. The output of the algorithm is shown below it: two regions are found, each corresponding to one of the objects in the scene. The numbers next to each polygon are the disparities associated with the region: The Venetian blind has a disparity of 18 pixels, whereas the more distant block has been successfully segmented at a lower disparity of 8 pixels, despite the foreground occlusion by the Venetian blind.

When the Venetian blind is placed vertically, the algorithm fails to cluster all portions of the block into a single region. The block has been "dissected" into three vertical strips. Due to large perspective differences between the two images, the blades of the Venetian blind are not found.

3.3 Real images

3.3.1 Non-textured background

Figure 8 shows two sets of real images involving a chair against a relatively low-texture background. In the first set of images, the illumination was from above; in the second set of images, an additional illumination source was placed facing the chair so that a shadow of the chair was cast on the background wall. In both sets of images, the output polygons of the algorithm are superimposed on the images.

In both sets of images, the non-textured background is seen to swamp the output: the largest polygons correspond to the background wall or curtain, or to the floor. Since these are real images,

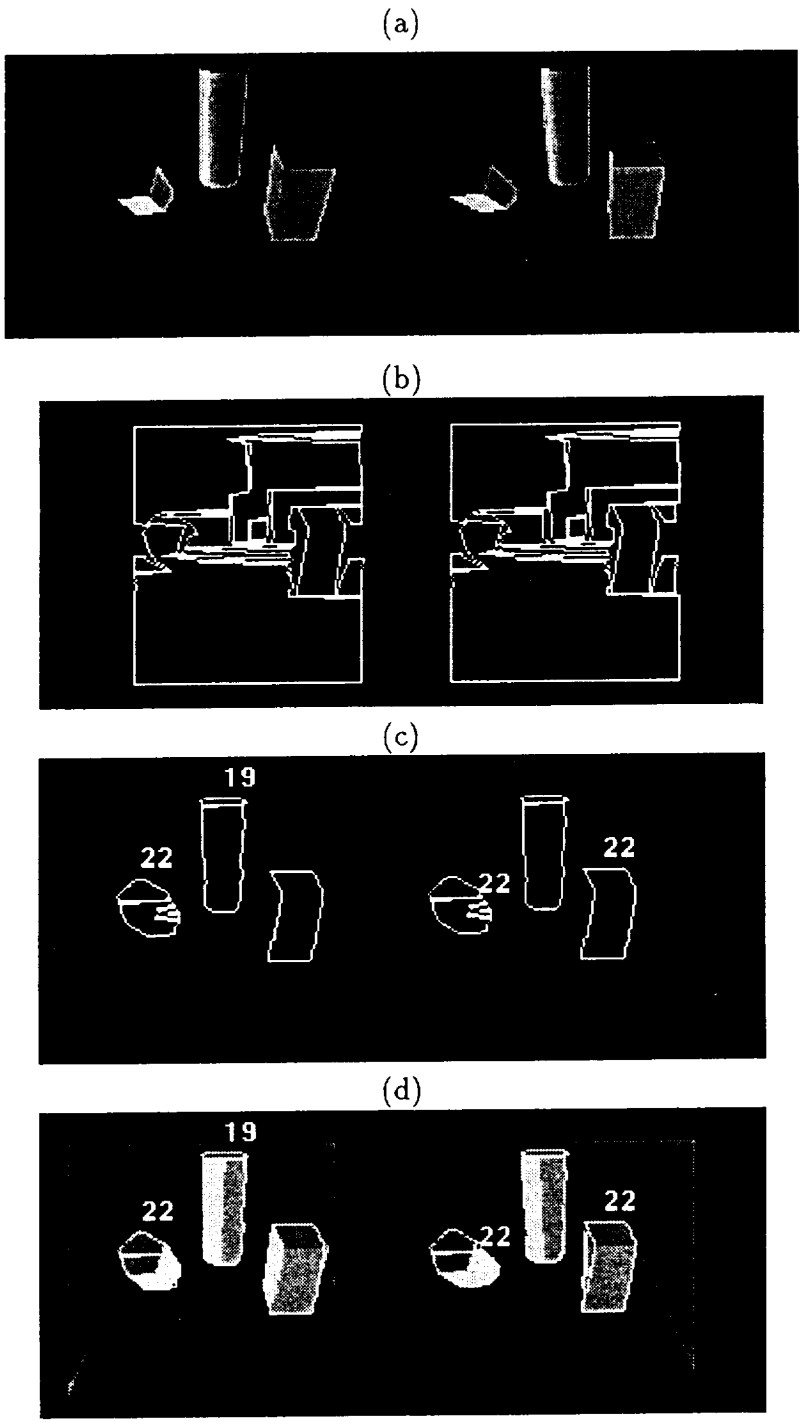
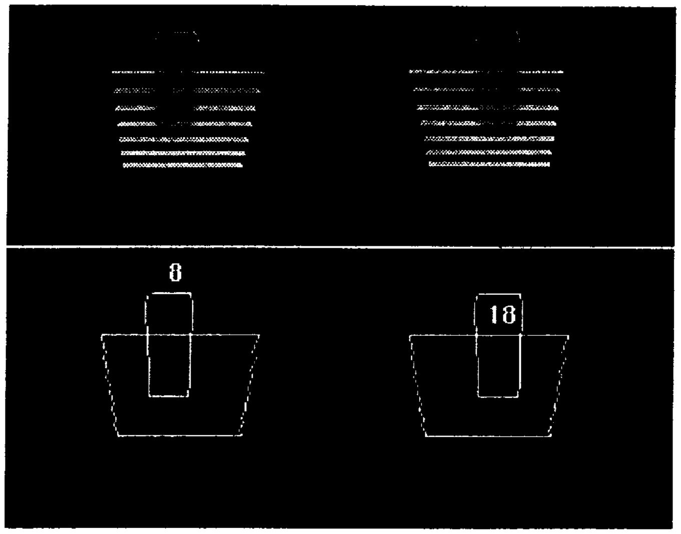
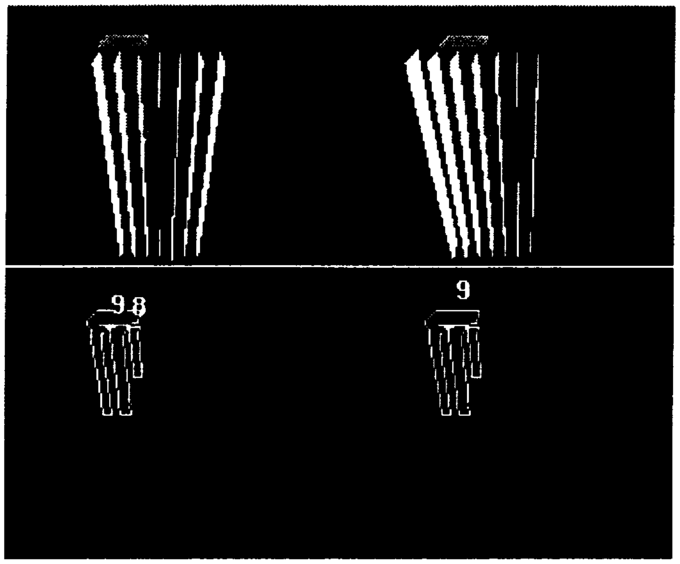


Figure 6: (a) Left and right blocks world images of an assortment of objects. (b) Output of algorithm on images including background and floor. (c) Output of algorithm on objects only (background and floor filtered out). Numbers represent average disparity for each region: more distant cylinder has lower disparity than front two objects. (d) Corresponding regions superimposed on the images.

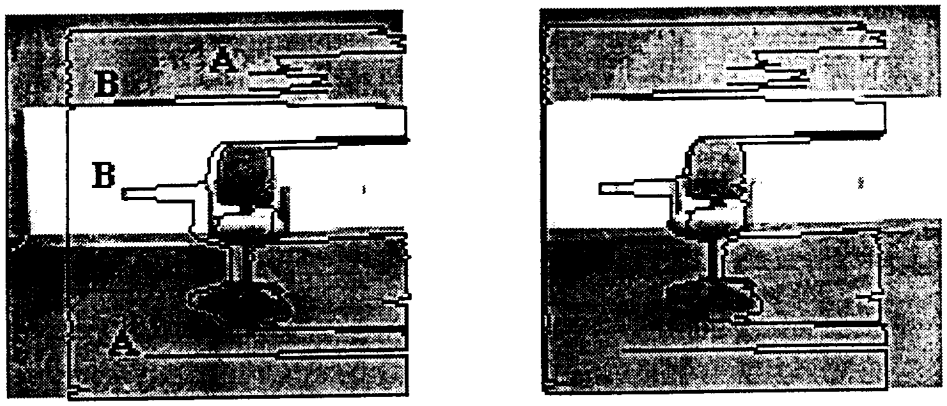


(a)

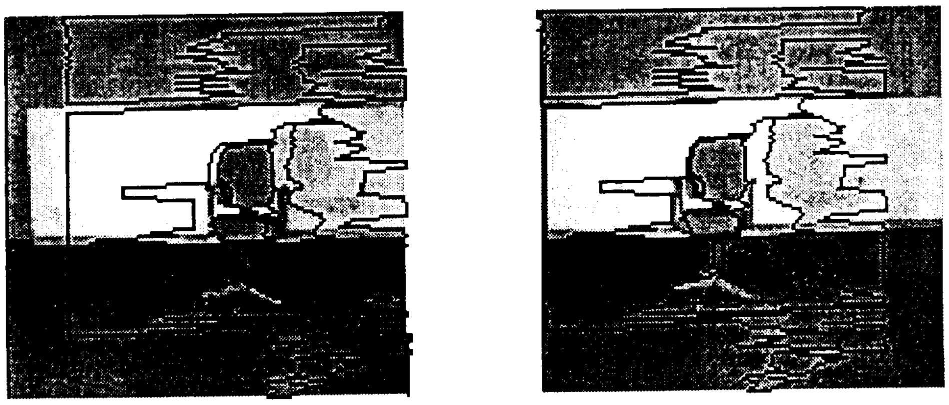


(b)

Figure 7: Example illustrating ability to handle certain types of transparency. (a) Left and right images of block behind *horizontal* Venetian blind, and corresponding output: two regions are found, one corresponding to the Venetian blind (average disparity 18), and the other to the more distant block (average disparity 8). (b) Left and right images of block behind *vertical* Venetian blind, and corresponding output: the block has been "dissected" into three vertical strips of average disparities 8, 9 and 9. Large perspective differences between the two images precludes the matching of the blades of the Venetian blind.



(a)



(b)

Figure 8: (a) Real left and right images of a chair against a controlled (low-texture) background, and the associated output. Though the shape of the chair is segmented, the largest regions correspond to the background. Since the intensities of objects and background are similar in real images, it is much more difficult to filter out background before processing than in synthetic images. (b) Real left and right images of a chair against a controlled (low-texture) background, and the associated output. Additional illumination from the front casts a shadow of the chair on the wall. Both chair and shadow are segmented, though the output is swamped by matches of the background.

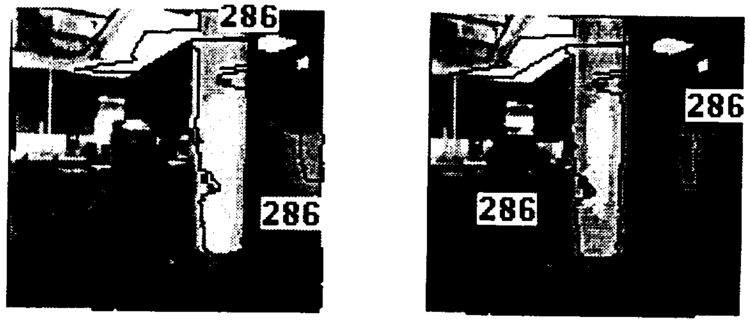


Figure 9: Output for a pair of real images with a non-contrived background. All regions are found to be portions of a frontal planar surface at distance 286 centimeters. Most notable region corresponds to the column. Examine the distances of the extracted region from the portion around the checker-board pattern at the lower left of the column: the algorithm has matched corresponding portions of the column, despite changes in perspective between the two images.

it is much more difficult to filter out the background and floor without also removing parts of the foreground objects of interest. Nevertheless, in the first set of images, the back and seat of the chair are segmented. Similarly, in the second set of images, both the chair and its shadow are clustered well into regions.

The errors in matching which are labelled "A" in the first set of images are due to noise in the images and differences in illumination between the left and right image. Those labelled "B" are due either to image noise, illumination effects, or to a misalignment of epipolar lines. The upper error occurs at a horizontal junction between dark and light areas, while the lower error is caused by a a small spot (electric socket) on the rear wall. The additional illumination in the second set of images creates greater contrast, and hence exacerbates these effects.

3.3.2 Textured background

Finally, Figure 9 shows the algorithm operating on a real set of images involving an non-contrived, textured background. Most of the regions corresponding to background objects do not exceed the minimum size requirement to be considered interesting. In fact, only five regions are displayed: two correspond to carpet, one to a patch of the side wall, one to the light fixture (matched with incorrect disparity), and one to the foreground column. The scene is similar to that of Figure 1, except that no chair is present.

The quality of the region corresponding to the column is significant: at first glance, it may appear that the algorithm has matched *different* portions of the column: in the left image, the region is aligned with the right-hand edge of the column, whereas in the right image, the region is aligned with the left edge of the column. However, the algorithm has, in fact, matched *corresponding* portions of the column very well. This can most easily be checked by examining the portion around the checker-board pattern on the lower left of the column: the polygon weaves around the pattern at approximately the same distance in both images. The reason for the *apparent* misalignment is that the two images show different perspectives of the column. The numbers in the image represent approximate depths; all regions were found with the same approximate disparity, and hence are shown with the same approximate depth of 286 centimeters. As mentioned in Section 2.3, each region is deemed to correspond to a portion of a frontal planar surface in space.

4 Conclusions

We have presented a novel stereo algorithm based on a coarse texture segmentation preprocessing phase. We selected a very simple texture segmentation preprocessing phase for our implementation: each pixel defines its own texture patch, and its intensity level is used to infer a texture label. More sophisticated texture segmentation would enhance the performance of the algorithm considerably. The use of texture *labels* rather than *values* allows greater versatility in the choice of the similarity relation between textures, and hence in the choice of topology in texture space: instead of the relation imposed by the usual metric on real numbers, any similarity relation may be specified.

Matching is performed using string comparison. A string matching approach has several advantages: By the nature of string matching, the *uniqueness* constraint is propagated automatically. If long substrings are searched for first, cohesivity is stressed, possibly at the expense of the number of total matches. Occlusions and exposures in the scene correspond to string deletions and insertions respectively, and cause no problems. The *monotonicity* constraint is not inherently necessary: a string-matching algorithm which handles transposition can be used. Any of a host of new string matching algorithms developed for use with genetic data may be called upon for the efficient execution of this phase. Finally, since each scanline is processed independently, this phase is amenable to parallel implementation.

Matching substrings correspond to matching sequences of textures, even in the presence of foreshortening. By analyzing the widths in the image plane of matching members of a sequence, foreshortening effects can be detected. In theory, these effects may be exploited for the reconstruction of surfaces at a non-zero angle to the image plane. In practice, however, it is difficult to disambiguate foreshortening from occlusion.

Inter-scanline clustering of matching substrings yields *regions* of matching texture. The shapes of these regions yield information concerning objects' heights, widths and azimuthal positions relative to the camera pair, while the average disparity of pixels in these regions may be used to estimate the objects' distances. Hence, rather than the standard dense depth map which must still be segmented and further processed, the output of this algorithm is a partial description of *objects* in the scene. We believe the output of clustered, partially segmented data by a stereo algorithm to be novel. Such a format has proved to be very useful for the integration of stereo with other sensor modalities. In particular, we are currently investigating the utility of this approach, in conjunction with other modalities, for pose estimation and mobile robot localization [11].

The algorithm can deal with certain types of transparencies. Further, the nature of the approach permits a slight realxation of the epipolar constraints. Furthermore, the algorithm is computationally efficient: the particular version of the algorithm which was implemented for the experiments in this paper uses a dynamic programming approach to keep the complexity somewhere between $O(n^2)$ (best case) and $O(n^3)$ (worst case) for an $n \times n$ image, *including the time required for intra-scanline clustering.* This compares favorably with the complexity for window-based correlation or SSD methods ($O(n^3w^2)$ for w-sized windows and for *unclustered* output). Comparison of efficiency with feature-based approaches is difficult, since the complexity of a feature-based approach depends on the type of feature detection performed.

The speed of the algorithm may be greatly enhanced by integration with other sensing modalities. In our implementation, for example, the ultrasound modality is used to measure depth of an object of interest. This measurement is used as to infer approximate disparity, and hence to guide the search for substring matches. The output of the stereo algorithm is *not* the depth of the object, which has already been measured accurately by the ultrasound modality. The output is, instead, the physical extents (width and height) and azimuthal location (relative to the camera pair) of that object of interest. This information, in conjuction with data from other sensor modalities, is very useful for landmark recognition and localization of a mobile robot. We are currently investigating this area of application.

The version algorithm has been tested on random dot stereograms, synthetic "blocks world" images with controlled backgrounds, indoor real images with controlled (low texture) backgrounds, and indoor real images with uncontrived (highly textured) backgrounds. The results are encouraging, with 100% successful matching on shifted pixels in the random dot stereogram, and good qualitative segmentation of images with easily filtered backgrounds (such as in synthetic images). Perhaps most significant, however, is the successful segmentation of foreground objects against a highly textured backgound in real uncontrived scenes.

References

[1] N. Ayache and B. Faverjon. Efficient registration of stereo images by matching graph descriptions of edge segments. *International Journal of Computer Vision*, pages 107–131, 1987.

[2] H. Baker and T. Binford. Depth from edge and intensity based stereo. *Proc. of 7th Int. Joint Conf. Artificial Intell.*, pages 631–636, August 1981.

[3] S. Blostein and T. Huang. Error analysis in stereo determination of 3-d point positions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):752–765, November 1987.

[4] I. J. Cox, S. Hingorani, B. M. Maggs, and S. B. Rao. A maximum likelihood stereo algorithm. *Submitted to CVGIP*, 1994.

[5] U. R. Dhond and J. K. Aggarwal. Structure from stereo — a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(16), November/December 1989.

[6] D. Gennery. Object detection and measurement using stereo vision. In *Proceedings of the 1980 Image Understanding Workshop*, pages 161–167, April 1980.

[7] W. Grimson. Computational experiments with a feature-based stereo algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(1):17–34, January 1985.

[8] M. Hannah. Bootstrap stereo. In *Proceedings of the 1980 Image Understanding Workshop*, pages 201–208, April 1980.

[9] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920–932, September 1994.

[10] H. S. Lim and T. O. Binford. Stereo correspondence: A hierarchical approach. In *Proceedings of the 1987 Image Understanding Workshop*, pages 234–241, February 1987.

[11] R. Mandelbaum. *Sensor Processing for Mobile Robot Localization, Exploration and Navigation.* PhD thesis, University of Pennsylvania, 1995.

[12] L. Matthies and S. Shafer. Error modeling in stereo navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, June 1987.

[13] G. Medioni and R. Nevatia. Segment-based stereo matching. *Computer Vision, Graphics, Image Processing*, 31:2–18, 1985.

[14] H. Moravec. Towards automatic visual obstacle avoidance. *Proc. of 5th Int. Joint Conf. Artificial Intell.*, page 584, 1977.

[15] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(2):139–154, March 1985.