A Benchmark for the Comparison of 3-D Motion Segmentation Algorithms

Roberto Tron René Vidal
Center for Imaging Science, Johns Hopkins University
308B Clark Hall, 3400 N. Charles St., Baltimore MD 21218, USA
http://www.vision.jhu.edu

Abstract

Over the past few years, several methods for segmenting a scene containing multiple rigidly moving objects have been proposed. However, most existing methods have been tested on a handful of sequences only, and each method has been often tested on a different set of sequences. Therefore, the comparison of different methods has been fairly limited. In this paper, we compare four 3-D motion segmentation algorithms for affine cameras on a benchmark of 155 motion sequences of checkerboard, traffic, and articulated scenes.

1. Introduction

Motion segmentation is a very important pre-processing step for several applications in computer vision, such as surveillance, tracking, action recognition, etc. During the nineties, these applications motivated the development of several 2-D motion segmentation techniques. Such techniques aimed to separate each frame of a video sequence into different regions of coherent 2-D motion (optical flow). For example, a video of a rigid scene seen by a moving camera could be segmented into multiple 2-D motions, because of depth discontinuities, occlusions, perspective effects, etc.

However, in several applications the scene may contain several moving objects, and one may need to identify each object as a coherent entity. In such cases, the segmentation task must be performed based on the assumption of several motions in 3-D space, not simply in 2-D. This has motivated several works on 3-D motion segmentation during the last decade, which can be roughly separated into two categories:

1. *Affine methods* assume an affine projection model, which generalizes orthographic, weak-perspective and paraperspective projection. Under the affine model, point trajectories associated with each moving object across multiple frames lie in a linear subspace of dimension at most 4. Therefore, 3-D motion segmentation can be achieved by clustering point trajectories into different motion subspaces. At present, several algebraic and statistical methods for performing this task have been developed (see §2 for a brief review). However, all existing techniques have been typically evaluated on a handful of sequences, with limited comparison against other methods. This motivates a study on the real performances of these methods.

2. *Perspective methods* assume a perspective projection model. In this case, point trajectories associated with each moving object lie in a multilinear variety (bilinear for two views, trilinear for three views, etc.) Therefore, motion segmentation is equivalent to clustering these multilinear varieties. Because this problem is nontrivial, most prior work has been limited to algebraic methods for factorizing bilinear and trilinear varieties (see *e.g.* [18, 7]) and statistical methods for two [15] and multiple [13] views. At present, the evaluation of perspective methods is still far behind that of affine methods. It is arguable that perspective methods still need to be significantly improved, before a meaningful evaluation and comparison can be made.

In this paper, we present a benchmark and a comparison of 3-D motion segmentation algorithms. We choose to compare only affine methods, not only because the affine case is better understood, but also because affine methods are at present better developed than their perspective counterparts. We compare four state-of-the-art algorithms, GPCA [16], Local Subspace Affinity (LSA) [21], Multi-Stage Learning (MSL) [14] and RANSAC [4], on a database of 155 motion sequences. The database includes 104 indoor checkerboard sequences, 38 outdoor traffic sequences, and 13 articulated/non-rigid sequences, all with two or three motions. Our experiments show that LSA is the most accurate method, with average classification errors of 3.45% for two motions and 9.73% for three motions. However, for two motions, GPCA and RANSAC are faster and have a limited 1%-2% drop in accuracy. More importantly, the results vary depending on the type of sequences: LSA is more accurate for checkerboard sequences, while GPCA is more accurate for traffic and articulated scenes. The MSL algorithm is often very accurate, but significantly slower.

2. Multibody Motion Segmentation Problem

In this section, we review the geometry of the 3-D motion segmentation problem from multiple affine views and show that it is equivalent to clustering multiple low-dimensional linear subspaces of a high-dimensional space.

2.1. Motion Subspace of a Rigid-Body Motion

Let $\{\boldsymbol{x}_{fp} \in \mathbb{R}^2\}_{p=1,\dots,P}^{f=1,\dots,F}$ be the projections of P 3-D points $\{\boldsymbol{X}_p \in \mathbb{P}^3\}_{p=1}^{P}$ lying on a rigidly moving object onto F frames of a rigidly moving camera. Under the affine projection model, which generalizes orthographic, weak perspective, and paraperspective projection, the images satisfy the equation

$$\boldsymbol{x}_{fp} = \mathtt{A}_f \boldsymbol{X}_p, \tag{1}$$

where $\mathtt{A}_f = \mathtt{K}_f \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathtt{R}_f & \boldsymbol{t}_f \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{2\times 4}$ is the *affine camera matrix* at frame f, which depends on the camera calibration parameters $\mathtt{K}_f \in \mathbb{R}^{2\times 3}$ and the object pose relative to the camera $(\mathtt{R}_f, \boldsymbol{t}_f) \in SE(3)$.

Let $\mathtt{W}_1 \in \mathbb{R}^{2F\times P}$ be the matrix whose P columns are the image point trajectories $\{\boldsymbol{x}_{fp}\}_{p=1}^{P}$. It follows from (1) that $\mathtt{W}_1$ can be decomposed into a *motion matrix* $\mathtt{M}_1 \in \mathbb{R}^{2F\times 4}$ and a *structure matrix* $\mathtt{S}_1 \in \mathbb{R}^{P\times 4}$ as

$$\mathtt{W}_1 = \mathtt{M}_1 \mathtt{S}_1^\top$$
$$\begin{bmatrix} \boldsymbol{x}_{11} \cdots \boldsymbol{x}_{1P} \\ \vdots \quad\quad \vdots \\ \boldsymbol{x}_{F1} \cdots \boldsymbol{x}_{FP} \end{bmatrix}_{2F\times P} = \begin{bmatrix} \mathtt{A}_1 \\ \vdots \\ \mathtt{A}_F \end{bmatrix}_{2F\times 4} \begin{bmatrix} \boldsymbol{X}_1 \cdots \boldsymbol{X}_P \end{bmatrix}_{4\times P}, \tag{2}$$

hence rank$(\mathtt{W}_1) \le 4$. Note also that the rows of each $\mathtt{A}_f$ involve linear combinations of the first two rows of the rotation matrix $\mathtt{R}_f$, hence rank$(\mathtt{W}_1) \ge$ rank$(\mathtt{A}_f) = 2$. Therefore, under the affine projection model, the 2-D trajectories of a set of 3-D points seen by a rigidly moving camera (the columns of $\mathtt{W}_1$) live in a subspace of $\mathbb{R}^{2F}$ of dimension $d_1 =$ rank$(\mathtt{W}_1) = 2$, 3 or 4.

2.2. Segmentation of Multiple Rigid-Body Motions

Assume now that the P trajectories $\{\boldsymbol{x}_{fp}\}_{p=1}^{P}$ correspond to n objects undergoing n rigid-body motions relative to a moving camera. The *3-D motion segmentation problem* is the task of clustering these P trajectories according to the n moving objects. Since the trajectories associated with each object span a d_i-dimensional linear subspace of $\mathbb{R}^{2F}$, the 3-D motion segmentation problem is equivalent to clustering a set of points into n subspaces of $\mathbb{R}^{2F}$ of unknown dimensions $d_i \in \{2, 3, 4\}$ for $i = 1, \dots, n$.

Notice that the data matrix can be written as

$$\mathtt{W} = \begin{bmatrix} \mathtt{W}_1, \mathtt{W}_2, \cdots, \mathtt{W}_n \end{bmatrix} \Gamma \in \mathbb{R}^{2F\times P}, \tag{3}$$

where the columns of $\mathtt{W}_i \in \mathbb{R}^{2F\times P_i}$ are the P_i trajectories associated with the ith moving object, $P = \sum_{i=1}^{n} P_i$, and $\Gamma^\top \in \mathbb{R}^{P\times P}$ is an unknown matrix permuting the P trajectories according to the n motions. Since $\mathtt{W}_i$ can be factorized into matrices $\hat{\mathtt{M}}_i \in \mathbb{R}^{2F\times d_i}$ and $\hat{\mathtt{S}}_i \in \mathbb{R}^{P_i\times d_i}$ as

$$\mathtt{W}_i = \hat{\mathtt{M}}_i \hat{\mathtt{S}}_i^\top \quad i = 1, \dots, n, \tag{4}$$

the matrix associated with all the objects can be factorized into matrices $\mathtt{M} \in \mathbb{R}^{2F\times \sum_{i=1}^{n} d_i}$ and $\mathtt{S} \in \mathbb{R}^{P\times \sum_{i=1}^{n} d_i}$ as

$$\begin{aligned} \mathtt{W} &= \begin{bmatrix} \mathtt{W}_1, \mathtt{W}_2, \cdots, \mathtt{W}_n \end{bmatrix} \Gamma \in \mathbb{R}^{2F\times P} \\ &= \begin{bmatrix} \hat{\mathtt{M}}_1, \hat{\mathtt{M}}_2, \cdots, \hat{\mathtt{M}}_n \end{bmatrix} \begin{bmatrix} \hat{\mathtt{S}}_1^\top & & & \\ & \hat{\mathtt{S}}_2^\top & & \\ & & \ddots & \\ & & & \hat{\mathtt{S}}_n^\top \end{bmatrix} \Gamma \\ &= \mathtt{M}\mathtt{S}^\top \Gamma. \end{aligned} \tag{5}$$

It follows that one possible way of solving the motion segmentation problem is to find a permutation matrix Γ, such that the matrix $\mathtt{W}\Gamma^\top$ can be decomposed into a motion matrix $\mathtt{M}$ and a *block diagonal* structure matrix $\mathtt{S}$. This idea has been the basis for most existing motion segmentation algorithms [1, 3, 5, 8, 10, 11, 19]. However, as shown in [10], in order for $\mathtt{W}$ to factor according to (5), the motion subspaces $\{\mathcal{W}_i \subset \mathbb{R}^{2F}\}_{i=1}^{n}$ must be *independent*, that is, for all $i \neq j = 1, \dots, n$, we must have $\dim(\mathcal{W}_i \cap \mathcal{W}_j) = 0$, so that rank$(\mathtt{W}) = \sum_{i=1}^{i} d_i$, where $d_i = \dim(\mathcal{W}_i)$.

Unfortunately, most practical motion sequences exhibit *partially dependent* motions, *i.e*. there are $i, j \in \{1, \dots, n\}$ such that $0 < \dim(\mathcal{W}_i \cap \mathcal{W}_j) < \min\{d_i, d_j\}$. For example, when two objects have the same rotational but different translational motion relative to the camera [14], or for articulated motions [20]. This has motivated the development of several algorithms for dealing with partially dependent motions, including statistical methods [6, 14], spectral methods [21, 22] and algebraic methods [16]. We review some of these methods in the next section.

3. Multibody Motion Segmentation Algorithms

3.1. Generalized PCA (GPCA) [17, 16]

Generalized Principal Component Analysis (GPCA) is an algebraic method for clustering data lying in multiple subspaces proposed by Vidal *et al.* [17]. The main idea behind GPCA is that one can fit a union of n subspaces with a set of polynomials of degree n, whose derivatives at a point give a vector normal to the subspace containing that point. The segmentation of the data is then obtained by grouping these normal vectors, which can be done using several techniques. In the context of motion segmentation, GPCA operates as follows [16]:

1. *Projection*: Project the trajectories onto a subspace of $\mathbb{R}^{2F}$ of dimension 5 to obtain the projected data matrix
$$\hat{\mathtt{W}} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_P] \in \mathbb{R}^{5\times P}.$$
The reason for projecting is as follows. Since the maximum dimension of each motion subspace is 4, projecting onto a generic subspace of dimension 5 preserves the number and dimensions of the motion subspaces. As a byproduct, there is an important reduction in the dimensionality of the problem, which is now reduced to clustering subspaces of dimension at most 4 in $\mathbb{R}^5$. Another advantage of the projection, is that it allows one to deal with missing data, as a rank-5 factorization of $\mathtt{W}$ can be computed using matrix factorization techniques for missing data (see [2] for a review).
2. *Multibody motion estimation via polynomial fitting*: Fit a homogeneous polynomial representing all motion subspaces to the projected data. For example, if we have n motion subspaces of dimension 4, then each one can be represented with a unique normal vector in $\mathbb{R}^5$ as $\{\boldsymbol{w} : \boldsymbol{b}_i^\top \boldsymbol{w} = 0\}$. The union of n subspaces is represented as $\{\boldsymbol{w} : q_n(\boldsymbol{w}) = (\boldsymbol{b}_1^\top \boldsymbol{w}) \cdots (\boldsymbol{b}_n^\top \boldsymbol{w}) = 0\}$. q_n is a polynomial of degree n in $\boldsymbol{w}$ that can be written as $\boldsymbol{c}^\top \nu_n(\boldsymbol{w})$, where $\boldsymbol{c}$ is the vector of coefficients, and $\nu_n(\boldsymbol{w})$ is the vector of all monomials of degree n in $\boldsymbol{w}$. The vector of coefficients is of dimension $\mathrm{O}(n^4)$ and can be computed from the linear system
$$\boldsymbol{c}^\top \begin{bmatrix}\nu_n(\boldsymbol{w}_1) & \nu_n(\boldsymbol{x}_2) & \cdots & \nu_n(\boldsymbol{w}_P)\end{bmatrix} = 0. \quad (6)$$
3. *Feature clustering via polynomial differentiation*: For $n = 2$, $\nabla q_2(\boldsymbol{w}) = (\boldsymbol{b}_2^\top \boldsymbol{w})\boldsymbol{b}_1 + (\boldsymbol{b}_1^\top \boldsymbol{w})\boldsymbol{b}_2$, thus if $\boldsymbol{w}_p$ belongs to the first motion, then $\nabla q_2(\boldsymbol{w}) \sim \boldsymbol{b}_1$. More generally, one can obtain the normal to the hyperplane containing point $\boldsymbol{w}_p$ from the gradient of $q_n(\boldsymbol{w})$ at $\boldsymbol{w}_p$
$$\boldsymbol{b}(\boldsymbol{w}_p) \sim \nabla q_n(\boldsymbol{w}_p). \quad (7)$$
One can then cluster the point trajectories by applying spectral clustering [12] to the similarity matrix $\mathtt{S}_{ij} = \cos^2(\theta_{ij})$, where θ_{ij} is the angle between the vectors $\nabla q_n(\boldsymbol{w}_i)$ and $\nabla q_n(\boldsymbol{w}_j)$ for $i, j = 1, \ldots, P$.

The first advantage of GPCA is that it is an algebraic algorithm, thus it is computationally very cheap. Second, as each subspace is represented with a hyperplane containing the subspace, intersections between subspaces are automatically allowed, and so the algorithm can deal with both independent and partially dependent motions. Third, GPCA can deal with missing data by performing the projection step using matrix factorization techniques for missing data [2].

The main drawback of GPCA is that $\boldsymbol{c}$ is of dimension $\mathrm{O}(n^4)$, while there are only $4n$ unknowns in the n normal vectors. Since $\boldsymbol{c}$ is computed using least-squares, this causes the performance of GPCA to deteriorate as n increases. Also, the computation of $\boldsymbol{c}$ is sensitive to outliers.

3.2. Local Subspace Affinity (LSA) [21]

The LSA algorithm proposed by Yan and Pollefeys in [21] is also based on a linear projection and spectral clustering. The main difference is that LSA fits a subspace *locally* around each projected point, while GPCA uses the gradients of a polynomial that is *globally* fit to the projected data. The main steps of the local algorithm are as follows:

1. *Projection*: Project the trajectories onto a subspace of dimension $D = \mathrm{rank}(\mathtt{W})$ using the SVD of $\mathtt{W}$. The value of D is determined using model selection techniques. The resulting points in $\mathbb{R}^D$ are then projected onto the hypersphere $\mathbb{S}^{D-1}$ by setting their norm to 1.
2. *Local subspace estimation*: For each point i, compute its k nearest neighbors using the angles between the vectors or their Euclidean distance as a metric. Then fit a local subspace $\mathcal{W}_i$ to the point and its neighbors. The dimension d_i of the subspace $\mathcal{W}_i$ depends on the kind of motion (e.g., general motion, purely translational, etc.) and the position of the 3-D points (e.g. general position, all on the same plane, etc.). The dimension d_i is also determined using model selection techniques.
3. *Spectral clustering*: Compute a similarity matrix between two points $i, j = 1, \ldots, P$ as
$$\mathtt{S}_{ij} = \exp\{-\sum_{m=1}^{d_{ij}} \sin^2(\theta_m)\}, \quad (8)$$
where the $\{\theta_m\}_{m=1}^{d_{ij}}$ are the principal angles between the two subspaces $\mathcal{W}_i$ and $\mathcal{W}_j$, and d_{ij} is the minimum between $\dim(\mathcal{W}_i)$ and $\dim(\mathcal{W}_j)$. Finally, cluster the features by applying spectral clustering [12] to S.

The LSA algorithm has two main advantages when compared to GPCA. First, outliers are likely to be “rejected”, because they are far from all the points and so they are not considered as neighbors of the inliers. Second, LSA requires only $Dn \leq 4n^2$ point trajectories, while GPCA needs $\mathrm{O}(n^4)$. On the other hand, LSA has two main drawbacks. First, the neighbors of a point could belong to a different subspace – this case is more likely to happen near the intersection of two subspaces. Second, the selected neighbors may not span the underlying subspace. Both cases are a source of potential misclassifications.

During our experiments, we had some difficulties in finding a set of model selection parameters that would work across all sequences. Thus, we decided to avoid model selection in the first two steps of the algorithm and fix both the dimension of the projected space D and the dimensions of the individual subspaces $\{d_i\}_{i=1}^n$. We used two choices for D. One choice is $D = 5$, which is the dimension used by GPCA. The other is $D = 4n$, which implicitly assumes that all motions are independent and full-dimensional. In our experiments in §5 we will refer to these two variants as *LSA 5* and *LSA* $4n$, respectively. As for the dimension of the individual subspaces, we assumed $d_i = 4$.

3.3. Multi-Stage Learning method (MSL) [14]

The Multi-Stage Learning (MSL) algorithm is a statistical approach proposed by Sugaya and Kanatani in [14]. It builds on Costeira and Kanade's factorization method (CK) [3] and Kanatani's subspace separation method (SS) [10, 11]. While the CK and SS methods apply to independent and non-degenerate subspaces, MSL can handle some classes of degenerate motions by refining the solution of SS using the Expectation Maximization algorithm (EM).

The CK algorithm proceeds by computing a rank-D approximation $\mathtt{V} \in \mathbb{R}^{P \times D}$ of $\mathtt{W}$ from its SVD $\mathtt{W} = \mathtt{U}\Sigma\mathtt{V}^\top$. As shown in [10], when the motions are *independent*, the shape interaction matrix $\mathtt{Q} = \mathtt{V}\mathtt{V}^\top \in \mathbb{R}^{P \times P}$ is such that

$$\mathtt{Q}_{ij} = 0 \text{ if points } i \text{ and } j \text{ belong to different objects.} \quad (9)$$

With noisy data, this equation holds only approximately. CK's algorithm obtains the segmentation by maximizing the sum of squared entries of the noisy $\mathtt{Q}$ in different groups. However, this process is very sensitive to noise [5, 10, 19].

The SS algorithm [10, 11] deals with noise using two principles: *dimension correction* and *model selection*. Dimension correction is used to induce exact zero entries in $\mathtt{Q}$ by replacing points in a group with their projections onto an optimally fitted subspace. Model selection, particularly the Geometric Akaike Information Criterion [9] (G-AIC), is used to decide whether to merge two groups. This can be achieved by applying CK's method to a scaled version of $\mathtt{Q}$

$$\mathtt{S}_{ij} = \frac{\text{G-AIC}_{\mathcal{W}_i, \mathcal{W}_j}}{\text{G-AIC}_{\mathcal{W}_i \cup \mathcal{W}_j}} \max_{k \in \mathcal{W}_i, l \in \mathcal{W}_j} |\mathtt{Q}_{kl}| \,. \quad (10)$$

However, in most practical sequences the motion subspaces are degenerate, *e.g.* of dimension three for 2-D translational motions. In this case the SS algorithm gives wrong results, because the calculation of the G-AIC uses the incorrect dimensions for the individual subspaces. The MSL algorithm deals with degenerate motions by assuming that the type of degeneracy is known (*e.g.* 2-D translational), and computing the G-AIC accordingly. Another issue is that in most practical sequences the motion subspaces are partially dependent. In this case, the SS algorithm also gives wrong results, because equation (9) does not hold even with perfect data. To overcome these issues, the MSL algorithm iteratively refines the segmentation given by the SS algorithm using EM for clustering subspaces as follows:

1. Obtain an initial segmentation using SS adapted to independent 2-D translational motions.
2. Use the current solution to initialize an EM algorithm adapted to independent 2-D translational motions.
3. Use the current solution to initialize an EM algorithm adapted to independent affine subspaces.
4. Use the current solution to initialize an EM algorithm adapted to full and independent linear subspaces.

The intuition behind the MSL algorithm is as follows. If the motions are degenerate, then the first two stages will give a good solution, which will simply be refined by the last two stages. On the other hand, if the motions are not degenerate, then the third stage will anyhow provide a good initialization for the last stage to operate correctly.

As with all algorithms based on EM, the MSL method suffers from convergence to a local minimum. Therefore, good initialization is needed to reach the global optimum. When the initialization is not good, it often happens that the algorithm takes a long time to converge (several hours), as it performs a series of optimization problems. Another disadvantage is that the algorithm is not designed for partially dependent motions, thus sometimes its performance is not ideal. In spite of these difficulties in theory, in practice the algorithm is quite accurate, as we will see in §5.

3.4. Random Sample Consensus (RANSAC) [4, 15]

RANdom SAmple Consensus (RANSAC) is a statistical method for fitting a model to a cloud of points corrupted with outliers in a statistically robust way. More specifically, if d is the minimum number of points required to fit a model to the data, RANSAC randomly samples d points from the data, fits a model to these d points, computes the residual of each data point to this model, and chooses the points whose residual is below a threshold as the inliers. The procedure is then repeated for another d sample points, until the number of inliers is above a threshold, or enough samples have been drawn. The outputs of the algorithm are the parameters of the model and the labeling of inliers and outliers.

In the case of motion segmentation, the model to be fit by RANSAC is a subspace of dimension d. Since there are multiple subspaces, RANSAC proceeds iteratively by fitting one subspace at a time as follows:

1. Apply RANSAC to the original data set and recover a basis for the first subspace along with the set of inliers. All points in other subspaces are considered as outliers.
2. Remove the inliers from the current data set and repeat step 1 until all the subspaces are recovered.
3. For each set of inliers, use PCA to find an optimal basis for each subspace. Segment the data into multiple subspaces by assigning each point to its closest subspace.

The main advantage of RANSAC is its ability to handle outliers explicitly. Also, notice that RANSAC can deal with partially dependent motions, because it computes one subspace at a time. However, the performance of RANSAC deteriorates quickly as the number of motions n increases, because the probability of drawing d inliers reduces exponentially with the number of subspaces. Another drawback of RANSAC is that it uses $d = 4$ as the dimension of the subspaces, which is not the minimum number of points needed to define a degenerate subspace (of dimension 2 or 3).

3.5. Reference

Data from real sequences contain not only noise and outliers, but also some degree of perspective effects, which are not accounted for by the affine model. Therefore, obtaining a perfect segmentation is not always possible.

In order to verify the validity of the affine model on real data, we will also compare the performance of affine algorithms with an "oracle" algorithm (here called *Reference*). This algorithm cannot be used in practice, because it requires the ground truth segmentation as an input. The algorithm uses least-squares to fit a subspace to the data points in each group using the SVD. Then, the data are re-segmented by assigning each point to its nearest subspace.

This Reference algorithm shows, with a perfect estimation of the subspaces, if the data can be segmented using the approximation of affine cameras and constitutes a good term of comparison for all the other (practical) algorithms.

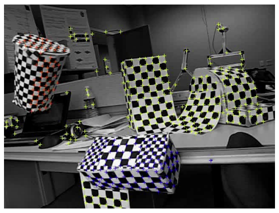
4. Benchmark

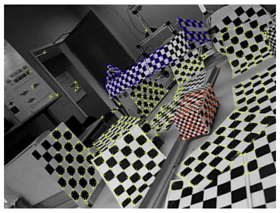
We collected a database of *50 video sequences* of indoor and outdoors scenes containing two or three motions. Each video sequence X with three motions was split into three motion sequences *X_g12*, *X_g13* and *X_g23* containing the points from groups one and two, one and three, and two and three, respectively. This gave a total of *155 motion sequences*: 120 with two motions and 35 with three motions.

Figure 1 shows a few sample images from the videos in the database with feature points superimposed. The entire database is available at `http://www.vision.jhu.edu`. These sequences contain degenerate and non-degenerate motions, independent and partially dependent motions, articulated motions, nonrigid motions, etc. To summarize the amount of motion present in all the sequences, we estimated the rotation and translation between all pairs of consecutive frames for each motion in each sequence. This information was used to produce the histograms shown in Figure 2.

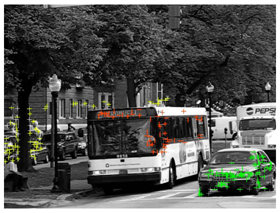
Based on the content of the video and the type of motion, the sequences can be categorized into three main groups:

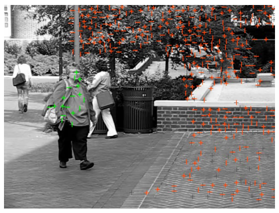
Checkerboard sequences: this group consists of 104 sequences of indoor scenes taken with a handheld camera under controlled conditions. The checkerboard pattern on the objects is used to assure a large number of tracked points. Sequences *1R2RC–2T3RTCR* contain three motions: two objects (identified by the numbers *1* and *2*, or *2* and *3*) and the camera itself (identified by the letter *C*). The type of motion of each object is indicated by a letter: *R* for rotation, *T* for translation and *RT* for both rotation and translation. If there is no letter after the *C*, this signifies that the camera is fixed. For example, if a sequence is called *1R2TC* it means that the first object rotates, the second translates and the camera is fixed. Sequence *three-cars* is taken from [18] and contains three motions of two toy cars and a box moving on a plane (the table) taken by a fixed camera.


(a) 1R2RCT_B


(b) 2T3RCRT

(c) cars3


(d) cars10


(e) people2

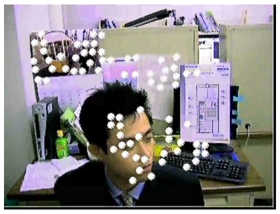
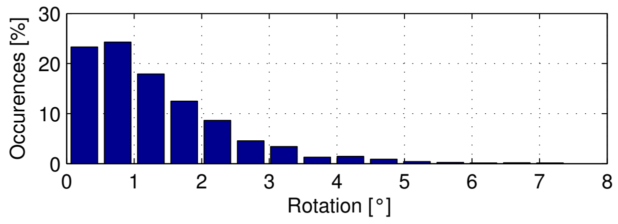
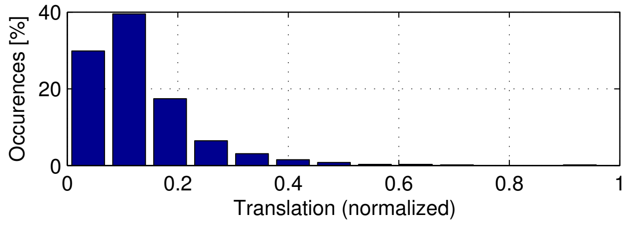

(f) kanatani3

Figure 1: Sample images from some sequences in the database with tracked points superimposed.



(a) Amount of rotation $\theta = \text{acos}\big((\text{trace}(\mathtt{R}_{f+1}^\top \mathtt{R}_f)) - 1)/2\big)$ in degrees.



(b) Amount of translation $\tau = \|\boldsymbol{t}_f\| / \max\{\text{depth}\}$.

Figure 2: Histograms with the amount of rotation and translation between two consecutive frames for each motion.

Traffic sequences: this group consists of 38 sequences of outdoor traffic scenes taken by a moving handheld camera. Sequences *carsX–truckX* have vehicles moving on a street. Sequences *kanatani1* and *kanatani2* are taken from [14] and display a car moving in a parking lot. Most scenes contain degenerate motions, particularly linear and planar motions.

Articulated/non-rigid sequences: this group contains 13 sequences displaying motions constrained by joints, head and face motions, people walking, etc. Sequences *arm* and *articulated* contain checkerboard objects connected by arm articulations and by strings, respectively. Sequences *people1* and *people2* display people walking, thus one of the two motions (the person walking) is partially non-rigid. Sequence *kanatani3* is taken from [14] and contains a moving camera tracking a person moving his head. Sequences *head* and *two_cranes* are taken from [21] and contain two and three articulated objects, respectively.

For the sequences used in [14, 18, 21], the point trajectories were provided in the respective datasets. For all the remaining sequences, we used a tool based on a tracking algorithm implemented in OpenCV, a library freely available at `http://sourceforge.net/projects/opencvlibrary`. The ground-truth segmentation was obtained in a semi-automatic manner. First, the tool was used to extract the feature points in the first frame and to track them in the following frames. Then an operator removed obviously wrong trajectories (*e.g.*, points disappearing in the middle of the sequence due to an occlusion by another object) and manually assigned each point to its corresponding cluster.

Table 1 reports the number of sequences and the average number of tracked points and frames for each category. The number of points per sequence ranges from 39 to 556, and the number of frames from 15 to 100. The table contains also the average distribution of points per moving object, with the last group corresponding to the camera motion (motion of the background). This statistic was computed on the original 50 videos only. Notice that typically the number of points tracked in the background is about twice as many as the number of points tracked in a moving object.

Table 1: Distribution of the number of points and frames.

	2 Groups			3 Groups		
	# Seq.	Points	Frames	# Seq.	Points	Frames
Check.	78	291	28	26	437	28
Traffic	31	241	30	7	332	31
Articul.	11	155	40	2	122	31
All	120	266	30	35	398	29
Point Distr.	35%-65%			20%-24%-56%		

5. Experiments

We tested the algorithms presented in §3 on our benchmark of 155 sequences. For each algorithm on each sequence, we recorded the classification error defined as

$$\text{classification error} = \frac{\text{\# of misclassified points}}{\text{total \# of points}} \qquad (11)$$

and the computation time (CPU time). Statistics with the classification errors and computation times for the different types of sequences are reported in Tables 2–5. Figure 3 shows histograms with the number of sequences in which each algorithm achieved a certain classification error. More detailed statistics with the classification errors and computation times of each algorithm on each of the 155 sequences can be found at `http://www.vision.jhu.edu`.

Because of the statistical nature of RANSAC, its segmentation results on the same sequence can vary in different runs of the algorithm. To have a meaningful result, we run the algorithm 1,000 times on each sequence and report

Table 2: Classification error statistics for two groups.

Check.	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	2.76%	6.41%	8.84%	2.57%	4.46%	6.52%
Median	0.49%	1.48%	3.43%	0.27%	0.00%	1.75%
Traffic	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	0.30%	1.31%	2.15%	5.43%	2.23%	2.55%
Median	0.00%	0.00%	1.00%	1.48%	0.00%	0.21%
Articul.	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	1.71%	3.65%	4.66%	4.10%	7.23%	7.25%
Median	0.00%	0.00%	1.28%	1.22%	0.00%	2.64%
All	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	2.03%	4.84%	6.73%	3.45%	4.14%	5.56%
Median	0.00%	0.30%	1.99%	0.59%	0.00%	1.18%

Table 3: Average computation times for two groups.

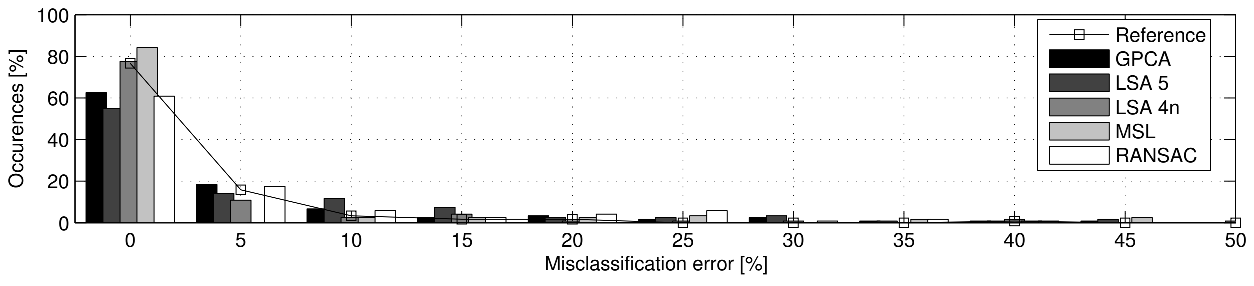
	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Check.	353ms	7.286s	8.237s	7h 4m	195ms
Traffic	288ms	6.424s	7.150s	21h 34m	107ms
Articul.	224ms	3.826s	4.178s	9h 47m	226ms
All	324ms	6.746s	7.584s	11h 4m	175ms

Table 4: Classification error statistics for three groups.

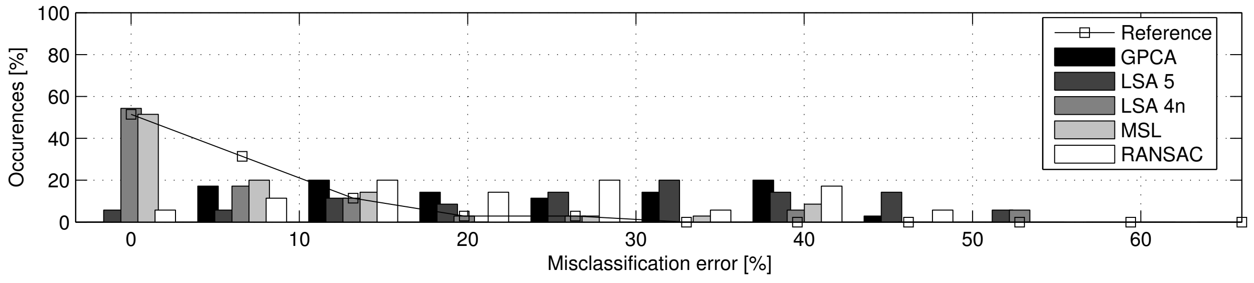
Check.	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	6.28%	27.48%	30.37%	5.80%	10.38%	25.78%
Median	5.06%	29.32%	31.98%	1.77%	4.61%	26.01%
Traffic	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	1.30%	14.03%	27.02%	25.07%	1.80%	12.83%
Median	0.00%	10.49%	34.01%	23.79%	0.00%	11.45%
Articul.	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	2.66%	14.77%	23.11%	7.25%	2.71%	21.38%
Median	2.66%	14.77%	23.11%	7.25%	2.71%	21.38%
All	REF	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Average	5.08%	24.06%	29.28%	9.73%	8.23%	22.94%
Median	2.40%	20.21%	31.63%	2.33%	1.76%	22.03%

Table 5: Average computation times for three groups.

	GPCA	LSA 5	LSA $4n$	MSL	RANSAC
Check.	842ms	16.711s	17.916s	2d 6h	285ms
Traffic	529ms	12.657s	12.834s	1d 8h	135ms
Articul.	125ms	1.175s	1.400s	1m 19.993s	338ms
All	738ms	15.013s	15.956s	1d 23h	258ms



(a) Two groups



(b) Three groups

Figure 3: Histograms with the percentage of sequences in which each method achieves a certain classification error.

the average classification error. Also, the thresholds were set with some hand-tuning on a couple of sequences (and then the same values were used for all the others).

The reference machine used for all the experiments is an Intel Xeon MP with 8 processors at 3.66GHz and 32GB of RAM (but for each simulation each algorithm exploits only one processor, without any parallelism).

6. Discussion

By looking at the results, we can draw the following conclusions about the performance of the algorithms tested.

Reference. The results from this “oracle” algorithm show that the affine camera approximation (linear subspaces) gives reasonably good results for nearly all the sequences. Indeed, the reference method gives a perfect segmentation for more than 50% of the sequences, with a classification error of 2% and 5% for two and three motions, respectively.

GPCA. For GPCA, we have to comment separately the results for sequences with two and three motions. For two motions, the classification error is 4.8% with an average computation time of 324 ms. For three motions, the results are completely different: the increase of computation time is reasonable (about 738 ms), but the segmentation error is significantly higher (about 24%). This is expected, because the number of coefficients fitted by GPCA grows exponentially with the number of motions. Nevertheless, notice that GPCA has higher errors on the checkerboard sequences, which constitute the majority of the database. Indeed, for the traffic and articulated sequences, GPCA is among the most accurate methods, both for two and three motions.

LSA. When the dimension for the projection is chosen as $D = 5$, the LSA algorithm performs worse than GPCA. This is because points in different subspaces are closer to each other when $D = 5$, and so a point from a different subspace is more likely to be chosen as a nearest neighbor. GPCA, on the other hand, is not affected by points near the intersection of the subspaces. The situation is completely different when we use $D = 4n$. The LSA $4n$ algorithm has the smallest error among all methods: 3.45% for two groups and 9.73% for three groups. We believe that these errors could be further reduced by using model selection to determine D. Another important thing to observe is that LSA $4n$ is the best method on the checkerboard sequences, but has larger errors than GPCA on the traffic and articulated sequences. On the complexity side, both variations of LSA have computation times in the order of 7-15 s, which are far greater than those of GPCA and RANSAC.

MSL. If we look only at the average classification error, we can see that MSL and LSA $4n$ are the most accurate methods. Furthermore, their segmentation results remain consistent when going from two to three motions. However, the MSL method has two major drawbacks. First, the EM algorithm can get stuck in a local minimum. This is reflected by high classification errors for some sequences where the Reference method performs well. Second, and more importantly, the complexity does not scale favorably with the number of points and frames, as the computation times grow in the order of minutes, hours and days. This may prevent the use of the MSL algorithm in practice, even considering its excellent accuracy.

RANSAC. The results for this purely statistic algorithm are similar to what we found for GPCA. Again, in the case of two sequences we obtain good segmentation results and the computation times are small. On the other and, the accuracy for three motions is not satisfactory. This is expected, because as the number of motions increases, the probability of drawing a set of points from the same group reduces significantly. Another drawback of RANSAC is that its performance varies between two runs on the same data.

7. Conclusions

We compared four different motion segmentation algorithms on a benchmark of 155 motion sequences. We found that the best performing algorithm (and the only one usable in practice for sequences with three groups) is the LSA approach with dimension of the projected space $D = 4n$. However, if we look only at sequences with two motions, GPCA and RANSAC can obtain similar results in a fraction of the time required by the others. Thus, they are apt to be used in real-time applications. Moreover, GPCA outperforms LSA when they work on the same dimension $D = 5$.

From the results given by the reference method, we conclude that there is still room for improvement using the affine camera approximation (as one can note from the gap between the best approaches and the reference algorithm, which is in the order of 1.5%-5%). It remains open to find a fast and reliable segmentation algorithm, usable in real-time applications, that works on sequences with three or more motions. We hope that the publication of this database will encourage the development of algorithms in this domain.

Acknowledgements

We thank M. Behnisch for helping with data collection, Dr. K. Kanatani for providing his datasets and code, and Drs. Y. Yan and M. Pollefeys for providing their datasets. This work has been supported by startup funds from Johns Hopkins University and by grants NSF CAREER IIS-04-47739, NSF EHS-05-09101, and ONR N00014-05-1083.

References

[1] T. Boult and L. Brown. Factorization-based segmentation of motions. *IEEE Workshop on Motion Understanding*, pages 179–186, 1991.

[2] A. Buchanan and A. Fitzgibbon. Damped Newton algorithms for matrix factorization with missing data. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 316–322, 2000.

[3] J. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[4] M. A. Fischler and R. C. Bolles. RANSAC random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 26:381–395, 1981.

[5] C. W. Gear. Multibody grouping from motion images. *International Journal of Computer Vision*, 29(2):133–150, 1998.

[6] A. Gruber and Y. Weiss. Multibody factorization with uncertainty and missing data using the EM algorithm. *IEEE Conference on Computer Vision and Pattern Recognition*, volume I, pages 707–714, 2004.

[7] R. Hartley and R. Vidal. The multibody trifocal tensor: Motion segmentation from 3 perspective views. *IEEE Conference on Computer Vision and Pattern Recognition*, volume I, pages 769–775, 2004.

[8] N. Ichimura. Motion segmentation based on factorization method and discriminant criterion. *IEEE International Conference on Computer Vision*, pages 600–605, 1999.

[9] K. Kanatani. Geometric information criterion for model selection. *International Journal of Computer Vision*, pages 171–189, 1998.

[10] K. Kanatani. Motion segmentation by subspace separation and model selection. In *IEEE International Conference on Computer Vision*, volume 2, pages 586–591, 2001.

[11] K. Kanatani and C. Matsunaga. Estimating the number of independent motions for multibody motion segmentation. *European Conference on Computer Vision*, pages 25–31, 2002.

[12] A. Ng, Y. Weiss, and M. Jordan. On spectral clustering: analysis and an algorithm. In *NIPS*, 2001.

[13] K. Schindler, J. U, and H. Wang. Perspective *n* -view multibody structure-and-motion through model selection. In *ECCV (1)*, pages 606–619, 2006.

[14] Y. Sugaya and K. Kanatani. Geometric structure of degeneracy for multi-body motion segmentation. In *Workshop on Statistical Methods in Video Processing*, 2004.

[15] P. Torr. Geometric motion segmentation and model selection. *Phil. Trans. Royal Society of London*, 356(1740):1321–1340, 1998.

[16] R. Vidal and R. Hartley. Motion segmentation with missing data by PowerFactorization and Generalized PCA. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume II, pages 310–316, 2004.

[17] R. Vidal, Y. Ma, and S. Sastry. Generalized Principal Component Analysis (GPCA). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1–15, 2005.

[18] R. Vidal, Y. Ma, S. Soatto, and S. Sastry. Two-view multibody structure from motion. *International Journal of Computer Vision*, 68(1):7–25, 2006.

[19] Y. Wu, Z. Zhang, T. Huang, and J. Lin. Multibody grouping via orthogonal subspace decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 252–257, 2001.

[20] J. Yan and M. Pollefeys. A factorization approach to articulated motion recovery. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–821, 2005.

[21] J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *European Conference on Computer Vision*, pages 94–106, 2006.

[22] L. Zelnik-Manor and M. Irani. Degeneracies, dependencies and their implications in multi-body and multi-sequence factorization. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 287–293, 2003.