Projective Factorization of Multiple Rigid-Body Motions

Ting Li[†] Vinutha Kallem[‡] Dheeraj Singaraju[†] René Vidal[†]
[†]Center for Imaging Science and [‡]Department of Mechanical Engineering, Johns Hopkins University
308B Clark Hall, 3400 N Charles St., Baltimore MD 21218, USA
http://www.vision.jhu.edu/

Abstract

Given point correspondences in multiple perspective views of a scene containing multiple rigid-body motions, we present an algorithm for segmenting the correspondences according to the multiple motions. We exploit the fact that when the depths of the points are known, the point trajectories associated with a single motion live in a subspace of dimension at most four. Thus motion segmentation with known depths can be achieved by methods of subspace separation, such as GPCA or LSA. When the depths are unknown, we proceed iteratively. Given the segmentation, we compute the depths using standard techniques. Given the depths, we use GPCA or LSA to segment the scene into multiple motions. Experiments on the Hopkins155 *motion segmentation database show that our method compares favorably against existing affine motion segmentation methods in terms of segmentation error and execution time.*

1. Introduction

The ability to extract scene geometry and motion is critical to many applications in computer vision, such as image based rendering, 3D localization and mapping, mosaicing, etc. Often, only a video sequence of the scene is available, with no prior knowledge about its structure or motion. This has motivated the following problem in computer vision:

Given multiple images taken by a rigidly moving camera observing a static scene, recover camera motion and scene structure from point correspondences in multiple views.

When the projection model is affine, this problem can be solved via direct factorization of the matrix of point correspondences $\mathtt{W} = \mathtt{M}\mathtt{S}^\top$ into its motion and structure components $\mathtt{M}$ and $\mathtt{S}$, respectively [12]. In the case of perspective cameras, the depths λ of the point correspondences are not known, thus the matrix of point correspondences $\mathtt{W}(\lambda)$ cannot be directly factorized. Algebraic methods proceed by algebraically eliminating the depths, solving for camera motion using two-view and three-view geometry, and computing the depths using triangulation [4]. However, these methods have difficulties handling all views simultaneously.

The Sturm/Triggs (ST) algorithm [10] obtains an initial estimate of the depths λ using two-view geometry. Then the matrix $\mathtt{W}(\lambda)$ containing point correspondences in all views, is factorized into the motion and structure of the scene. Simple iterative extensions to the ST algorithm (SIESTA) [14, 4] alternate between the estimation of motion + structure and the estimation of the depths. Unfortunately, without proper initialization, SIESTA can converge to a trivial solution where all the depths are zero. [7] proposed an extension of SIESTA that incorporates additional constraints into the optimization problem. However, [8] showed that in spite of such constraints, SIESTA can still result in trivial solutions. To overcome these issues, [8] proposed a provably convergent method called CIESTA which uses regularization in the optimization to adjust the estimated depths so as to keep them close to their correct values.

Over the past few years, there has been an increasing interest on extending motion estimation methods to scenes with multiple motions. This requires one to group the points according to the different motions before applying standard motion estimation techniques to each group. In computer vision, this is known as the motion segmentation problem:

Given point trajectories corresponding to n objects undergoing n different rigid-body motions relative to the camera, cluster the trajectories according to the n motions.

This problem has been addressed mostly under the assumption of an affine camera model, where-in the trajectories associated with each motion live in a linear subspace of dimension four or less [12]. This subspace constraint was used by Costeira and Kanade (CK) [1] to propose a multi-frame 3-D motion segmentation algorithm based on thresholding the entries of the so-called *shape interaction* matrix $\mathtt{Q}$. This matrix is built from the singular value decomposition (SVD) of the matrix of point trajectories $\mathtt{W}$, and has the property that $\mathtt{Q}_{ij} = 0$ when points i and j correspond to independent motions. However, this thresholding process is very sensitive to noise [2, 5]. Kanatani scales the entries of $\mathtt{Q}$ using the geometric Akaike's information criterion for linear [5] and affine [6] subspaces. Gear [2] uses bipartite graph matching to threshold the entries of the row echelon canonical form of $\mathtt{W}$. Unfortunately, the equation $\mathtt{Q}_{ij} = 0$ holds only when the motion subspaces are linearly independent [5]. That is, CK's algorithm and its extensions are not provably correct for most practical motion sequences which usually exhibit partially dependent motions. The local subspace affinity method (LSA) [19] deals with partially dependent motions by locally estimating a subspace passing through each point trajectory. The point trajectories are then segmented by applying spectral clustering to a similarity matrix built from the angles between pairs of subspaces. As the subspaces are estimated locally, LSA cannot deal with transparent motions. [16] deals with transparent and partially dependent motions and also with missing data. This is done by globally fitting a union of subspaces to the data using GPCA+PowerFactorization. As the method is mostly algebraic, it has difficulties dealing with outliers.

Motion segmentation using perspective cameras is relatively less well studied, because in this case the trajectories associated with each motion live on a manifold. Algebraic methods for two [18] and three [3] views are based on polynomial fitting and differentiation. While these methods perform well for small number of motions and moderate noise levels, they fail in the presence of outliers and when the number of motion increases. Statistical methods combine two-view geometry with model selection and RANSAC [13] or Monte-Carlo sampling [9]. While these methods perform well for a small number of motions, as the number of motion increases, the number of candidate motions generated by random sampling grows exponentially. Moreover, as these methods depend on an initial clustering obtained from two views, some post-processing is needed to handle multiple views. Thus, these methods can fail if the motion between two particular views is extremely noisy.

In this paper, we present what is to the best of our knowledge the first algorithm for segmenting multiple rigid-body motions using multiple perspective views simultaneously. Our method is a natural extension of the iterative ST algorithm to multiple motions, by alternating between the estimation of the depths and the segmentation of the point trajectories. Given the depths, we use subspace separation (GPCA or LSA) to segment the scene into multiple motions. Given the segmentation, we obtain the depth of each point using standard techniques. Therefore, our method can also be seen as a natural generalization of subspace separation methods from multiple affine to multiple perspective views.

We test our algorithm on the *Hopkins155* motion segmentation database, which includes 155 motion sequences with two and three motions. We explore several variants of the main algorithm, including different choices for the initialization method, subspace separation method, data normalization method, etc. The best method achieves a classification error of 3.27% for two motions and 6.23% for three motions, with a reasonable execution time.

2. Projective Factorization of a Single Motion

Projective factorization is a method for computing the structure and motion of a single moving object from image points in multiple affine [12] or perspective [10] views.

In the case of affine cameras, it is well known that $\boldsymbol{x}_{fp} \in \mathbb{R}^2$, which is the inhomogeous coordinate of the image of point $\boldsymbol{X}_p \in \mathbb{P}^3$ in frame f, satisfies the projection equation

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

$\mathtt{A}_f \in \mathbb{R}^{2\times 4}$ is the affine camera matrix for frame f, which depends on the relative motion between the object and the camera at frame f. Using (1), we see that the projection equation for all the P points in F frames can be written as

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

Each column of the $2F \times P$ measurement matrix $\mathtt{W}$ corresponds to the trajectory of a $3D$ point in the image plane. Given their forms, $\mathtt{M}$ is called the motion matrix and $\mathtt{S}$ is called the structure matrix. It is clear from equation (2) that the rank of $\mathtt{W}$ is at most four, and hence the trajectories should live in a subspace of dimension at most four. In practice, noise can cause an apparent increase of rank. This can be accounted for by projecting $\mathtt{W}$ to have rank four using the SVD of $\mathtt{W}$. More specifically, if $\mathtt{W} = \mathtt{U}\Sigma\mathtt{V}^\top$, we can obtain the rank four projection of $\mathtt{W}$ as

$$\hat{\mathtt{W}} = \hat{\mathtt{U}}\hat{\Sigma}\hat{\mathtt{V}}^\top = \mathtt{U}(:,1:4)\Sigma(1:4,1:4)\mathtt{V}(:,1:4)^\top. \tag{3}$$

Matrices $\mathtt{M}$ and $\mathtt{S}$ can then be estimated up to a projective transformation $\mathtt{A} \in GL(4)$, as $\mathtt{M} = \hat{\mathtt{U}}\hat{\Sigma}\mathtt{A}$ and $\mathtt{S} = \mathtt{A}^{-1}\hat{\mathtt{V}}^\top$.

In the case of perspective cameras, the projection model (1) becomes

$$\lambda_{fp}\boldsymbol{x}_{fp} = \Pi_f \boldsymbol{X}_p, \tag{4}$$

where λ_{fp} is the projective depth of $\boldsymbol{x}_{fp} \in \mathbb{P}^2$, which is now the homogenous coordinate of the image point. $\Pi_f \in \mathbb{R}^{3\times 4}$ is the camera matrix for the frame f, which depends on the motion of the object relative to the camera at frame f. Therefore, all point trajectories can stacked into a matrix $\mathtt{W}(\lambda)$, which is now of dimension $3F \times P$ due to the homogeneous representation for the image points. As before, this matrix can be factorized into motion and structure as

$$\begin{array}{ccccc} \overbrace{\begin{bmatrix} \lambda_{11}\boldsymbol{x}_{11} \cdots \lambda_{1P}\boldsymbol{x}_{1P} \\ \vdots \qquad \vdots \\ \lambda_{F1}\boldsymbol{x}_{F1} \cdots \lambda_{FP}\boldsymbol{x}_{FP} \end{bmatrix}}^{3F\times P} & = & \overbrace{\begin{bmatrix} \Pi_1 \\ \vdots \\ \Pi_F \end{bmatrix}}^{3F\times 4} & \overbrace{\begin{bmatrix} \boldsymbol{X}_1 \cdots \boldsymbol{X}_P \end{bmatrix}}^{4\times P} \\ \mathtt{W}(\lambda) & = & \mathtt{M} & \mathtt{S}^\top. \end{array} \tag{5}$$

We can see from (5) that if the depths were known, then the rank of $\mathtt{W}(\lambda)$ would be at most 4. Therefore, motion and structure could be estimated from the SVD of $\mathtt{W}(\lambda)$, similar to the case of affine cameras. In reality, however, the depths are unknown. To overcome this challenge, Sturm and Triggs [10] propose to estimate the depths using principles from 2-view geometry. Treating the first frame f_0 as a reference, they estimate the fundamental matrix $\mathcal{F}_{f_0 f}$ for each frame f using the 8 point algorithm [4]. The epipole $\boldsymbol{e}_{f_0 f}$ is then obtained as the left null vector of $F_{f_0 f}$. Once these quantities are known, the depths are estimated as

$$\lambda_{fp} = \frac{(\boldsymbol{e}_{f_0 f} \times \boldsymbol{x}_{fp})^\top (\mathcal{F}_{f_0 f} \boldsymbol{x}_{f_0 p})}{\|\boldsymbol{e}_{f_0 f} \times \boldsymbol{x}_{fp}\|^2} \lambda_{f_0 p}. \tag{6}$$

Given the depths, $\mathtt{W}(\lambda)$ can be factorized as $\mathtt{MS}^\top$.

A drawback of the ST algorithm is that it relies on a correct estimation of the depths from two-view geometry. This may not be possible in practice, because the image measurements are corrupted by noise. This issue motivated several iterative extensions to the ST algorithm [14, 4, 7, 8]. Given an initial estimate for the depths, these methods iterate between two steps. First, estimating a rank-4 projection $\hat{\mathtt{W}}$ and consequently, the structure and motion. Second, estimating the depths from $\hat{\mathtt{W}}$ in the least square sense as

$$\lambda_{fp} = \frac{\hat{\mathtt{W}}_{fp}^\top \boldsymbol{x}_{fp}}{\boldsymbol{x}_{fp}^\top \boldsymbol{x}_{fp}}. \tag{7}$$

where $\hat{\mathtt{W}}_{fp}$ are the entries in $\hat{\mathtt{W}}$ corresponding to $\lambda_{fp}\boldsymbol{x}_{fp}$. This iterative method can be initialized using 2-view geometry, as in equation (6). Alternatively, one can initialize the algorithm by setting all the depths to 1. It was shown in [4] that this is a good first order approximation to the projective depths when the ratio of true depths of the different 3D points remains approximately constant during a sequence.

3. Projective Factorization of Multiple Motions

In this section, we present a generalization of the ST algorithm to the case of multiple rigid-body motions.

Given F perspective views $\{\boldsymbol{x}_{fp}\}$ of P points lying in n moving objects, our goal is to recover the structure and motion associated with each object and the grouping of the P point trajectories according to the n motions. To that end, let $\mathtt{W}_i(\lambda) \in \mathbb{R}^{3F \times P_i}$ for $i = 1, \dots, n$, be a matrix whose columns are the trajectories of P_i points belonging to the ith moving object. Then, the data matrix containing the trajectories of all points can be written as

$$\mathtt{W}(\lambda) = \left[\mathtt{W}_1(\lambda), \mathtt{W}_2(\lambda), \cdots, \mathtt{W}_n(\lambda)\right] \Gamma \in \mathbb{R}^{3F \times P}, \tag{8}$$

where $P = \sum_{i=1}^{n} P_i$ is the total number of points and $\Gamma^\top \in \mathbb{R}^{P \times P}$ is an unknown matrix permuting the columns of $\mathtt{W}(\lambda)$ according to the n motions.

From the single-body case, we know that each $\mathtt{W}_i(\lambda)$ can be factorized into a motion matrix and a structure matrix as $\mathtt{W}_i(\lambda) = \mathtt{M}_i \mathtt{S}_i^\top$. Thus, the data matrix $\mathtt{W}(\lambda)$ associated with the n motions can be factorized as

$$\begin{aligned} \mathtt{W}(\lambda) &= \left[\mathtt{W}_1(\lambda) \quad \cdots \quad \mathtt{W}_n(\lambda)\right] \Gamma \\ &= \left[\mathtt{M}_1 \quad \cdots \quad \mathtt{M}_n\right] \begin{bmatrix} \mathtt{S}_1^\top & & 0 \\ & \ddots & \\ 0 & & \mathtt{S}_n^\top \end{bmatrix} \Gamma \\ &= \qquad \mathtt{M} \qquad\qquad\qquad \mathtt{S}^\top \qquad \Gamma, \end{aligned} \tag{9}$$

where $\mathtt{M} \in \mathbb{R}^{3F \times D}$ and $\mathtt{S} \in \mathbb{R}^{D \times P}$ for $D \leq 4n$.

Recall that if the depths λ were known, then the trajectories corresponding to a single motion would live in a subspace of $\mathbb{R}^{3F}$ of dimension at most four. In the case of multiple motions, although the permutation matrix Γ is unknown, the columns of $\mathtt{W}(\lambda)$ live in a union of n subspaces of $\mathbb{R}^{3F}$ of dimension at most 4. Therefore, motion segmentation with known depths is equivalent to clustering the point trajectories according to n low-dimensional subspaces. This problem can be solved using techniques of subspace separation, such as [1, 2, 5, 6, 19].

In reality, however, the depths of the points are unknown, and we are faced with a chicken-and-egg problem. Given the depths, we can segment the point trajectories by applying subspace separation to the columns of $\mathtt{W}(\lambda)$. Given the segmentation of the point trajectories, we can apply factorization to each group to estimate the depths of the points using equation (7). This motivates the following iterative algorithm for solving the motion segmentation problem.

Algorithm 1. **(Multibody motion segmentation from multiple perspective views)**
Given the trajectories of P feature points in F frames

1. **Initialization**: Obtain an initial estimate of the depths or an initial segmentation of the point trajectories.

2. **Motion segmentation**: Given the depths, form the matrix $\mathtt{W}(\lambda)$ and segment its columns into different subspaces using subspace separation.

3. **Depth estimation**: Given the segmentation of the point trajectories, apply factorization to each group to obtain an estimate of the depth of each point.

4. **Iterative refinement**: Iterate between steps 2 and 3 until convergence of the depths and segmentation.

5. **Motion estimation**: given the final segmentation, recover motion and structure for each group using factorization.

We describe the main steps of our motion segmentation algorithm in the following subsections.

3.1. Initialization

We can choose to initialize the algorithm with the depths of the points or with the segmentation of the trajectories.

In the first case, we set all the initial depths to be one. This is because there is no existing method that helps estimate the depths without knowledge of the segmentation.

In the second case, we explore two options: (1) assume the data to be captured by an affine camera and segment the trajectories using the algorithms proposed in [16] and [19], or (2) segment the motions between pairs of views using the two-view segmentation algorithm of [18] and then combine the segmentation results from these pairs of views using a voting scheme. Although the latter method might not give a good segmentation if the data in some views is too noisy, it still serves as a good initialization to boostrap the algorithm.

3.2. Motion segmentation

Once the depths are known, we can form the $\mathtt{W}(\lambda)$ matrix and segment the trajectories. To that end, notice that each trajectory is of dimension $3F$, which can be quite high when working with a large number of views. Recalling that the trajectories corresponding to each motion live in a subspace of dimension at most 4, we project the trajectories onto a generic subspace of dimension $D \geq 5$ using the SVD of $\mathtt{W}(\lambda)$. This yields a new matrix of projected trajectories

$$\tilde{\mathtt{W}}(\lambda) = \big[\boldsymbol{w}_1, \cdots, \boldsymbol{w}_P\big] \in \mathbb{R}^{D \times P}. \tag{10}$$

This projection not only reduces the dimension of the feature vectors, but also preserves the dimension and number of the individual subspaces, as shown in [17].

The problem is now reduced to clustering subspaces of dimension at most 4 in $\mathbb{R}^D$. Since our algorithm is iterative, we would like each step of the iteration not only to yield good results, but also to be computationally cheap. As shown in [15], Generalized PCA (GPCA) [17] and Local Subspace Affinity (LSA) [19] are two subspace clustering algorithms satisfying these criteria. The following paragraphs describe each one of these methods.

Generalized PCA (GPCA): GPCA [17] is an algebraic algorithm used for clustering data lying in multiple linear subspaces. In the case of motion segmentation, each motion subspace is of dimension 4, thus the first step of GPCA is to project the data onto a subspace of dimension $D = 5$. Each projected subspace is then a hyperplane that can be represented by its unique normal $\boldsymbol{b}_i \in \mathbb{R}^5$ as $\{\boldsymbol{w} : \boldsymbol{b}_i^\top \boldsymbol{w} = 0\}$. The union of n subspaces is then represented as

$$\{\boldsymbol{w} : q_n(\boldsymbol{w}) = \boldsymbol{c}^\top \nu_n(\boldsymbol{w}) = (\boldsymbol{b}_1^\top \boldsymbol{w}) \cdots (\boldsymbol{b}_n^\top \boldsymbol{w})) = 0\}, \tag{11}$$

where $\nu_n(\boldsymbol{w})$ contains all homogeneous monomials of degree n in $\boldsymbol{w}$, and $\boldsymbol{c}$ is the vector of coefficients of the polynomial q_n representing all the motion subspaces. Given the number of subspaces, $\boldsymbol{c}$ can be estimated from the projected data $\boldsymbol{w}$ in a least squares sense. Note that for the simple case $n = 2$, we have $\nabla q_n(\boldsymbol{w}) = (\boldsymbol{b}_2^\top \boldsymbol{w})\boldsymbol{b}_1 + (\boldsymbol{b}_1^\top \boldsymbol{w})\boldsymbol{b}_2$. It is easy to check that $\nabla q_n(\boldsymbol{w}) \sim \boldsymbol{b}_1$ or $\boldsymbol{b}_2$, if $\boldsymbol{w}$ belongs to the first or second motion subspace, respectively. In general, for any n one can obtain the normal to the hyperplane passing through $\boldsymbol{w}_p$ as $\nabla q_n(\boldsymbol{w}_p)$. This allows one to define a similarity matrix between two points i and j as $\mathtt{S}_{ij} = \cos(\theta_{ij})$, where θ_{ij} is the angle between $\nabla q_n(\boldsymbol{w}_i)$ and $\nabla q_n(\boldsymbol{w}_j)$. Segmentation of the point trajectories is then obtained by applying spectral clustering to this similarity matrix.

Local Subspace Affinity (LSA): LSA [19] is another subspace separation method, which is also based on a linear projection followed by spectral clustering. In LSA, the data is first projected onto a subspace of dimension $D \geq 5$, where D is determined by model selection. The projected data is then further projected onto the hypersphere $\mathcal{S}^{D-1}$ by enforcing the norm of each point to be 1. The method then finds the k nearest neighbors to each point i and locally fits a subspace $\mathcal{W}_i$ to the point and its neighbors. The dimension d_i of this subspace $\mathcal{W}_i$ is dependent upon the kind of motion and the configuration of the points. Subsequently, a similarity matrix $\mathtt{S} \in \mathbb{R}^{P \times P}$ is constructed with its elements being defined as $\mathtt{S}_{ij} = \exp\{-\sum_{m=1}^{d_{ij}} \sin^2(\theta_m)\}$, where $\{\theta_m\}_{m=1}^{d_{ij}}$ are the principal angles between the subspaces $\mathcal{W}_i$ and $\mathcal{W}_j$, and d_{ij} is the minimum between d_i and d_j. The trajectories are then segmented by applying spectral clustering to this similarity matrix.

3.3. Depth estimation

Given the segmentation of the point trajectories into n groups, we may apply factorization to obtain the motion and structure associated with each moving object. However, as our motion segmentation algorithm depends only on the depths of the points, we use a simplified factorization method to obtain the depths of the points as follows:

Algorithm 2. **(Iterative depth estimation algorithm)**
Given the trajectory of P feature points in F frames corresponding to a single motion

1. **Depth initialization**: Set the depths $\{\lambda_{fp}\}_{f=1,\dots,F}^{p=1,\dots,P}$ to be 1, or get initial estimates using equation (6).
2. **Projection**: Compute the rank 4 approximation of $\mathtt{W}(\lambda)$ to get $\hat{\mathtt{W}}(\lambda)$ as in equation (3).
3. **Depth estimation**: Linearly estimate the depths λ_{fp} as in equation (7) to minimize the reprojection error between the entries of $\mathtt{W}(\lambda)$ and $\hat{\mathtt{W}}(\lambda)$.
4. **Iterative refinement**: Iterate between steps 2 and 3 until $\frac{\|\hat{\mathtt{W}}(\lambda) - \mathtt{W}(\lambda)\|^2}{\|\mathtt{W}(\lambda)\|^2}$ is below a specified threshold.

4. Experiments

In §3, we saw that there are different variations of our algorithm, depending on (1) the method used for initialization, (2) the method used for subspace separation, and (3) the initialization of the algorithm for estimating the depths. In this section, we test a few of these variations on real data and analyze the results to conclude which version performs better. More specifically, we test the following variations:

1. *DepthInit-GPCA* and *DepthInit-LSA* refer to variations where we bootstrap Algorithm 1 by initializing all the depths to be equal to 1, and perform subspace separation using using GPCA and LSA, respectively.
2. *SegInit-GPCA* and *SegInit-LSA* refer to variations where we bootstrap Algorithm 1 by giving an initial segmentation. The initialization is done using the same segmentation scheme as that used in [16] or [19].

In all these variations, Algorithm 2 for estimating depths is initialized by setting all depths to 1. Also, recall that before segmentation we perform a rank-D approximation of the $\mathtt{W}(\lambda)$ matrix. Following [15], we use $D = 5$ for GPCA and $D = 4n$ for LSA, where n is the number of motions.

We compare these 4 variations with existing algorithms for motion segmentation from multiple affine views. More specifically, we compare with GPCA [16], Local Subspace Affinity (LSA) [19] and Multi-Stage Learning (MSL) [11].[1] The comparison is made on the *Hopkins155* database [15], which consists of 155 sequences of both indoor and outdoor scenes with degenerate and non-degenerate motions, independent and partially dependent motions, articulated motions, nonrigid motions, etc. The database contains 120 sequences with 2 motions and 35 sequences with 3 motions. The sequences can further be categorized into checkerboard sequences, traffic sequences and scenes with articulated motions. We present results for each category in order to get a better analysis of the algorithms' performance.

For each variant of our algorithm, we investigated the use of certain techniques that affect its numerical performance. In each case, the combination of techniques giving the best performance was chosen. The first technique involves normalizing the columns and rows of the $\mathtt{W}(\lambda)$ matrix to be 1, and is referred to as **balancing** [14]. Balancing prevents the factorization step from being numerically ill-conditioned. From our experiments, we concluded that our algorithm performs better when using balancing in the segmentation step. However, we did not use balancing in the depth estimation step, because it did not always improve the results. The second technique is that of **projecting** the trajectories to have norm 1 before segmenting them. This eliminates any scaling and ensures that the subspaces are well separated. In our experiments, the projection step improved the results when LSA was used for subspace separation. However, when using GPCA, the projection step was useful only for initialization and not in the iterative stage, because GPCA is applied to homogeneous data with balancing. Finally, we considered **normalizing** the image coordinates as suggested in [10, 4]. This was done in order to ensure that the homogeneous coordinates of the points were of the same order of magnitude. All variants of our method performed better with normalized data. However, the initial segmentations by GPCA and LSA were obtained using un-normalized data, because affine methods operate on in-homogeneous coordinates.

We now present a comparison of the statistics of errors and computation times of different segmentation methods. For each sequence in the database, the classification error is computed with respect to the ground truth as the percentage of incorrectly classified points. Tables 1 and 3 give the error statistics for sequences with 2 and 3 motions, respectively. Tables 2 and 4 list the statistics of computation times for the same. Figure 1 gives two histograms of the classification errors over the sequences with 2 and 3 motions, respectively.

Table 1. Classification errors (%) for sequences with 2 motions

Method	DepthInit-GPCA	DepthInit-LSA	SegInit-GPCA	SegInit-LSA	GPCA	LSA	MSL
Checkerboard: 78 sequences							
Mean	4.29	2.86	4.18	2.61	6.09	2.57	4.46
Median	2.09	0.25	2.24	0.42	1.03	0.27	0.00
Traffic: 31 sequences							
Mean	0.52	5.74	0.65	4.54	1.41	5.43	2.23
Median	0.00	1.55	0.00	1.30	0.00	1.48	0.00
Articulated: 11 sequences							
Mean	4.79	7.95	4.55	4.38	2.88	4.10	7.23
Median	0.43	1.39	0.43	1.39	0.00	1.22	0.00
All: 120 sequences							
Mean	3.36	4.07	3.30	3.27	4.59	3.45	4.14
Median	0.59	0.51	0.53	0.53	0.38	0.59	0.00

Table 2. Average computation times for 2 motions.

Method	DepthInit-GPCA	DepthInit-LSA	SegInit-GPCA	SegInit-LSA	GPCA	LSA	MSL
Check.	223.48s	118.46s	160.34s	142.55s	317ms	15.73s	7h 4m
Traffic	174.23s	73.29s	112.29s	124.11s	273ms	9.00s	21h 34m
Articul.	64.57s	41.92s	25.85s	71.06s	165ms	7.34s	9h 47m
All	196.19s	99.78s	135.60s	131.23s	292ms	13.22s	11h 4m

Tables 1 and 3 show that our iterative algorithm often improves the results of affine segmentation by GPCA [16] or LSA [19]. However, the improvement is not significant, as affine methods already give good results. Also, using LSA in our algorithm is better than using GPCA for three motions, though the results are similar for two motions.

[1]Notice that the affine methods (including GPCA and LSA) operate on a $2F \times P$ data matrix $\mathtt{W}$. However, whenever we use GPCA or LSA in step 2 of our method, we use them on a $3F \times P$ matrix $\mathtt{W}(\lambda)$.

Table 3. Classification errors (%) for sequences with 3 motions

Method	DepthInit-GPCA	DepthInit-LSA	SegInit-GPCA	SegInit-LSA	GPCA	LSA	MSL
Checkerboard: 26 sequences							
Mean	20.54	6.67	22.44	5.55	31.95	5.80	10.38
Median	17.30	1.00	23.20	1.21	32.93	1.77	4.61
Traffic: 7 sequences							
Mean	2.46	10.21	8.00	9.51	19.83	25.07	1.80
Median	0.55	4.71	2.06	4.71	19.55	23.79	0.00
Articulated: 2 sequences							
Mean	6.72	2.13	7.05	3.52	16.85	7.25	2.71
Median	6.72	2.13	7.05	3.52	16.85	7.25	2.71
All: 35 sequences							
Mean	16.13	7.12	18.68	6.23	28.66	9.73	8.23
Median	12.77	1.62	15.86	1.62	28.26	2.33	1.76

Table 4. Average computation times for 3 motions.

Method	DepthInit-GPCA	DepthInit-LSA	SegInit-GPCA	SegInit-LSA	GPCA	LSA	MSL
Check.	348.73s	218.81s	354.01s	133.19s	406ms	14.18s	7h 4m
Traffic	184.39s	150.06s	281.79s	159.39s	257ms	9.51s	21h 34m
Articul.	138.20s	57.94s	231.82s	19.39s	187ms	1.12s	9h 47m
All	303.83s	195.87s	332.58s	131.93s	357ms	12.50s	11h 4m

Furthermore, using LSA in our algorithm always outperforms MSL. Tables 2 and 4 show that our algorithm takes about hundreds of seconds to segment a sequence. Thus our method is much faster than MSL that can take hours to segment a sequence.[2] Therefore, our approach performs on par, if not better than other methods, and it does so at low computational expense.

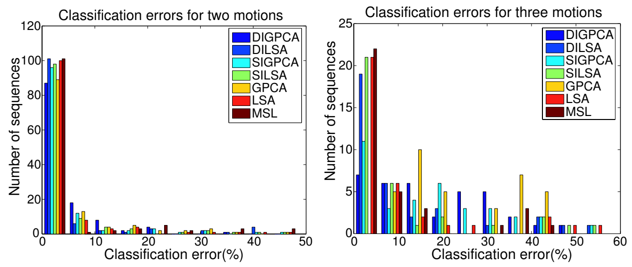

Figure 1. Classification error histograms on Hopkins155 database.

5. Conclusions

We have presented a new algorithm for segmenting multiple rigid motions using multiple perspective views simultaneously. The algorithm uses the trajectories across all frames at once, hence it is preferable over algorithms that combine the segmentation over 2 or 3 views. Our method is iterative in nature and alternates between segmentation of the point trajectories and the estimation of the depths. A detailed analysis of the performance of the different variations of our algorithm on the Hopkins155 database shows that our method compares favorably against existing methods in terms of misclassification error as well as computation time.

[2]The MSL method is run on an Intel Xeon MP with 8 processors at 3.66GHz and 32GB of RAM, while our method is run on an Intel Xeon TM with 2 processors at 3.2GHz with 4GB of RAM (but each algorithm exploits only one processor, without any parallelism).

Acknowledgements

This work is 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] J. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *IJCV*, 29(3):159–179, 1998. 1, 3

[2] C. W. Gear. Multibody grouping from motion images. *IJCV*, 29(2):133–150, 1998. 1, 3

[3] R. Hartley and R. Vidal. The multibody trifocal tensor: Motion segmentation from 3 perspective views. In *CVPR*, volume I, pages 769–775, 2004. 2

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, 2000. 1, 3, 5

[5] K. Kanatani. Motion segmentation by subspace separation and model selection. In *ICCV*, volume 2, pages 586–591, 2001. 1, 2, 3

[6] K. Kanatani and C. Matsunaga. Estimating the number of independent motions for multibody motion segmentation. In *ECCV*, pages 25–31, 2002. 1, 3

[7] S. Mahamud, M. Hebert, Y. Omori, and J. Ponce. Provably-convergent iterative methods for projective structure from motion. In *CVPR*, volume I, pages 1018–1025, 2001. 1, 3

[8] J. Oliensis and R. Hartley. Iterative extensions of the sturm/triggs algorithm: Convergence and nonconvergence. In *ECCV (4)*, pages 214–227, 2006. 1, 3

[9] K. Schindler, J. U, and H. Wang. Perspective n -view multibody structure-and-motion through model selection. In *ECCV (1)*, pages 606–619, 2006. 2

[10] P. Sturm and B. Triggs. A factorization based algorithm for multi–image projective structure and motion. In *ECCV*, pages 709–720, 1996. 1, 2, 3, 5

[11] Y. Sugaya and K. Kanatani. Geometric structure of degeneracy for multi-body motion segmentation. In *Workshop on Statistical Methods in Video Processing*, 2004. 5

[12] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography. *IJCV*, 9(2):137–154, 1992. 1, 2

[13] P. H. S. Torr. Geometric motion segmentation and model selection. *Phil. Trans. Royal Society of London*, 356(1740):1321–1340, 1998. 2

[14] B. Triggs. Factorization methods for projective structure and motion. In *CVPR*, pages 845–51, 1996. 1, 3, 5

[15] R. Tron and R. Vidal. A benchmark for the comparsion of 3-D motion segmentation algorithms. In *CVPR*, 2007. 4, 5

[16] R. Vidal and R. Hartley. Motion segmentation with missing data by PowerFactorization and Generalized PCA. In *CVPR*, volume II, pages 310–316, 2004. 2, 4, 5

[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. 4

[18] R. Vidal, Y. Ma, S. Soatto, and S. Sastry. Two-view multibody structure from motion. *IJCV*, 68(1):7–25, 2006. 2, 4

[19] J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *ECCV*, pages 94–106, 2006. 2, 3, 4, 5