Motion Segmentation via Robust Subspace Separation in the Presence of Outlying, Incomplete, or Corrupted Trajectories *

Shankar R. Rao† Roberto Tron‡ René Vidal‡ Yi Ma†

†Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{srrao,yima}@uiuc.edu

‡Center for Imaging Science
Johns Hopkins University
{tron,rvidal}@cis.jhu.edu

Abstract

We examine the problem of segmenting tracked feature point trajectories of multiple moving objects in an image sequence. Using the affine camera model, this motion segmentation problem can be cast as the problem of segmenting samples drawn from a union of linear subspaces. Due to limitations of the tracker, occlusions and the presence of nonrigid objects in the scene, the obtained motion trajectories may contain grossly mistracked features, missing entries, or not correspond to any valid motion model. In this paper, we develop a robust subspace separation scheme that can deal with all of these practical issues in a unified framework. Our methods draw strong connections between lossy compression, rank minimization, and sparse representation. We test our methods extensively and compare their performance to several extant methods with experiments on the Hopkins 155 database. Our results are on par with state-of-the-art results, and in many cases exceed them. All MATLAB code and segmentation results are publicly available for peer evaluation at `http://perception.csl.uiuc.edu/coding/motion/`.

1. Introduction

A fundamental problem in computer vision is to infer structures and movements of 3D objects from a video sequence. While classical multiple-view geometry typically deals with the situation where the scene is static, recently there has been growing interest in the analysis of dynamic scenes. Such scenes often contain multiple motions, as there could be multiple objects moving independently in a scene, in addition to camera motion. Thus an important initial step in the analysis of video sequences is the *motion segmentation* problem. That is, given a set of feature points that are tracked through a sequence of video frames, one seeks to cluster the trajectories of those points according to different motions.

In the literature, many different camera models have been proposed and studied, such as paraperspective, orthographic, affine and perspective. Among these the affine camera model is arguably the most popular, due largely to its generality and simplicity. Thus, in this paper, we assume the affine camera model, and show how to develop a robust solution to the motion segmentation problem. Before we delve into our problems of interest, we first review the basic mathematical setup.

Basic Formulation of Motion Segmentation. Suppose we are given trajectories of P tracked feature points of a rigid object $\{(x_{fp}, y_{fp})\}_{f=1\ldots F}^{p=1\ldots P}$ from F 2-D image frames of a rigidly moving camera. The affine camera model stipulates that these tracked feature points are related to their 3-D coordinates $\{(X_p, Y_p, Z_p)\}_{p=1}^{P}$ by the matrix equation:

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1P} \\ y_{11} & y_{12} & \cdots & y_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ x_{F1} & x_{F2} & \cdots & x_{FP} \\ y_{F1} & y_{F2} & \cdots & y_{FP} \end{bmatrix}}_{\mathtt{Y}\in\mathbb{R}^{2F\times P}} = \underbrace{\begin{bmatrix} A_1 \\ \vdots \\ A_F \end{bmatrix}}_{\mathtt{A}\in\mathbb{R}^{2F\times 4}} \underbrace{\begin{bmatrix} X_1 & \cdots & X_P \\ Y_1 & \cdots & Y_P \\ Z_1 & \cdots & Z_P \\ 1 & \cdots & 1 \end{bmatrix}}_{\mathtt{X}\in\mathbb{R}^{4\times P}},$$

$$\mathtt{Y} = \mathtt{A}\mathtt{X} \tag{1}$$

where $A_f = K_f \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \end{bmatrix} \begin{bmatrix} R_f & \boldsymbol{t}_f \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{2\times 4}$ is the *affine motion matrix* at frame f. The affine motion matrix is parameterized by the camera calibration matrix $K_f \in \mathbb{R}^{2\times 3}$ and the relative orientation of the rigid object w.r.t. the camera $(R_f, \boldsymbol{t}_f) \in SE(3)$. From this formulation we see that

$$\text{rank}(\mathtt{Y}) = \text{rank}(\mathtt{A}\mathtt{X}) \leq \min(\text{rank}(\mathtt{A}), \text{rank}(\mathtt{X})) \leq 4. \tag{2}$$

Thus the affine camera model postulates that trajectories of feature points from a single rigid motion will all lie in a linear subspace of $\mathbb{R}^{2F}$ of dimension at most four.

A dynamic scene can contain multiple moving objects, in which case the affine camera model for a single rigid motion cannot be directly applied. Now let us assume that the given P trajectories correspond to N moving objects. In this case, the set of all trajectories will lie in a *union of N linear subspaces* in $\mathbb{R}^{2F}$, but we do not know which trajectory belongs to which subspace. Thus, the problem of assigning each trajectory to its corresponding motion reduces to the problem

*This work was partially supported by grants NSF EHS-0509151, NSF CCF-0514955, ONR YIP N00014-05-1-0633, NSF IIS-0703756, NSF CAREER 0447739, NSF EHS-0509101, ONR N00014-05-1083 and WSE/APL Contract: Information Fusion & Localization in Distributed Sensor Systems.

of segmenting data drawn from multiple subspaces, which we refer to as *subspace separation*.

Problem 1 (Motion Segmentation via Subspace Separation). *Given a set of trajectories of P feature points $Y = [\boldsymbol{y}_1 \dots \boldsymbol{y}_P] \in \mathbb{R}^{2F \times P}$ from N rigidly moving objects in a dynamic scene, find a permutation Γ of the columns of the data matrix Y:*

$$Y = [Y_1 \dots Y_N]\Gamma^{-1}, \tag{3}$$

such that the columns of each submatrix Y_n, $n = 1, \dots, N$, are trajectories of a single motion.

Related Work on Motion Segmentation. In the literature, there are many approaches to motion segmentation, that can roughly be grouped into three categories: factorization-based, algebraic, and statistical.

Many early attempts at motion segmentation attempt to directly factor $\mathtt{Y}$ according to (3) [1, 7, 11, 12]. To make such approaches tractable, the motions must be independent of one another, *i.e*. the pairwise intersection of the motion subspaces must be the zero vector. However, for most dynamic scenes with a moving camera or containing articulated objects, the motions are at least partially dependent on each other. This has motivated the development of algorithms designed to deal with dependent motions.

Algebraic methods, such as Generalized Principal Component Analysis (GPCA) [19], are generic subspace separation algorithms that do not place any restriction on the relative orientations of the motion subspaces. However, when a linear solution is used, the complexity of algebraic methods grows *exponentially* with respect to both the dimension of the ambient space and the number of motions in the scene, and so algebraic methods are not scalable in practice.

The statistical methods come in many flavors. Many formulate motion segmentation as a statistical clustering problem that is tackled with Expectation-Maximization (EM) or variations of it [17, 13, 9]. As such, they are iterative methods that require good initialization, and can potentially get stuck in suboptimal local minima. Other statistical methods use local information around each trajectory to create a pairwise similarity matrix that can then be segmented using *spectral clustering* techniques [24, 22, 5].

Robustness Issue and Our Approach. Many of the above approaches assume that all trajectories are good, with perhaps a moderate amount of noise. However, real motion data acquired by a tracker can be much more complicated:

1. A trajectory may correspond to certain nonrigid or random motions that do not obey the affine camera model (an *outlying trajectory*).
2. Some of the features may be missing in some frames, causing a trajectory to have some missing entries (an *incomplete trajectory*).
3. Even worse, some feature points may be mistracked (with the tracker unaware), causing a trajectory to have some entries with gross errors (a *corrupted trajectory*).

While some of the methods can be modified to be robust to *one* of such problems [9, 5, 23, 22, 20], to our knowledge there is no motion segmentation algorithm that can elegantly deal with all of these problems in a unified fashion.

In this paper, we propose a new motion segmentation scheme that draws heavily from the principles of *data compression* and *sparse representation*. We show that the new algorithm naturally handles outlying trajectories, and can be designed to repair incomplete or corrupted trajectories.[1] Our methods use the affine camera model assumption, so we do not make any comparisons with perspective camera-based methods[2]. As most extant methods for motion segmentation assume that the number of motions is known, for fair comparison, we also assume the group count is given.

2. Robust Subspace Separation

In this section, we describe the subspace separation method that we use for motion segmentation and show that by properly exploiting the low rank subspace structure in the data, our method can be made robust to the three kinds of pathological trajectories discussed earlier.

To a large extent, the goal of subspace separation is to find a partition of the data matrix $\mathtt{Y}$ into submatrices $\{\mathtt{Y}_n\}_{n=1}^N$ such that each $\mathtt{Y}_n$ is maximally rank deficient. *Matrix rank minimization* (MRM) is itself a very challenging problem. The rank function is neither smooth nor convex, and so finding a matrix $\mathtt{M}$ that is maximally rank deficient among a convex set of matrices is known to be NP-Hard. Also, the rank function is highly unstable in the presence of noise. For a positive semidefinite matrix $\mathtt{M} \in \mathbb{R}^{D \times D}$, one can deal with both instability and computational intractability by minimizing the following smooth surrogate for $\mathrm{rank}(\mathtt{M})$:

$$J(\mathtt{M}, \delta) \doteq \log_2 \det\left(\delta \mathtt{I} + \mathtt{M}\right), \tag{4}$$

where $\delta > 0$ is a small regularization parameter [6].

As we are not minimizing $\mathrm{rank}(\mathtt{Y}_n)$ over a convex set, subspace separation is not technically an instance of MRM. However, after a slight modification to (4), we can see a connection between MRM and the principle of *lossy minimum description length* (LMDL). Given data $\mathtt{Y}_n \in \mathbb{R}^{D \times P_n}$ drawn from a linear subspace, the number of bits needed to code the data $\mathtt{Y}_n$ up to distortion ε^2 [15] [3] is given by

[1] We make a distinction between incomplete and corrupted trajectories: for incomplete trajectories, we know in which frames the features are missing; for corrupted ones, we do not have that knowledge.

[2] Please refer to [16] for work on robust motion segmentation with a perspective camera model.

[3] It can be shown that as $\varepsilon \to 0$, (5) converges to the optimal coding length for a Gaussian source, and is also an upper bound for the coding length of subspace-like data.

$$L(\mathtt{Y}_n, \varepsilon) \doteq \frac{D+P_n}{2}\left[J\left(\frac{1}{P_n}\mathtt{Y}_n\mathtt{Y}_n^T, \frac{\varepsilon^2}{D}\right) - \log_2\det\left(\frac{\varepsilon^2}{D}\mathtt{I}\right)\right]$$
$$= \frac{D+P_n}{2}\log_2\det\left(\mathtt{I} + \frac{D}{P_n\varepsilon^2}\mathtt{Y}_n\mathtt{Y}_n^T\right). \quad (5)$$

$L(\mathtt{Y}_n, \varepsilon)$ is still a smooth surrogate for rank$(\mathtt{Y}_n)$, as it is obtained by subtracting a constant term from $J(\mathtt{M}, \delta)$, with $\mathtt{M} = \frac{1}{P_n}\mathtt{Y}_n\mathtt{Y}_n^T$ and $\delta = \frac{\varepsilon^2}{D}$, and scaling by a constant factor.

Now suppose the data matrix $\mathtt{Y} \in \mathbb{R}^{D\times P}$, can be partitioned into disjoint subsets $\mathtt{Y} = [\mathtt{Y}_1 \ldots \mathtt{Y}_N]$ of corresponding sizes $P_1 + \cdots + P_N = P$. If we encode each subset separately, the total number of bits required is

$$L^s(\{\mathtt{Y}_1, \ldots, \mathtt{Y}_N\}, \varepsilon) \doteq \sum_{n=1}^{N} L(\mathtt{Y}_n, \varepsilon) - P_n \log_2 \frac{P_n}{P}. \quad (6)$$

The second term in this equation counts the number of bits needed to represent the membership of the P vectors in the N subsets (*i.e.* by Huffman coding). In [15], Ma *et al.* posit that the optimal segmentation of the data minimizes the number of bits needed to encode the segmented data up to distortion ε^2.

Finding a global minimum of (6) is a combinatorial problem. Nevertheless, an agglomerative algorithm, proposed in [15], has been shown to be very effective for minimizing (6). It initially treats each sample as its own group, iteratively merging pairs of groups so that the resulting coding length is maximally reduced at each iteration. The algorithm terminates when it can no longer reduce the coding length. We refer to their algorithm as *Agglomerative Lossy Compression* (ALC). See [15] for more details.

2.1. Outlying Trajectories

Dynamic scenes often contain trajectories that do not correspond to any of the motion models in the scene. Such outlying trajectories can arise from motions not well described by the affine camera model, such as the motion of non-rigid objects. These kinds of trajectories have been referred to as "sample outliers" by [2], suggesting that no subset of the trajectory corresponds to any affine motion model. Fortunately, ALC deals with these outliers in an elegant fashion. In [15], it was observed that in low dimensions (≤ 3), all outliers tend to cluster into a single group. This is because in low dimensions it is very unlikely that outliers live in a lower-dimensional subspace. Hence it is more efficient to code them together with respect to a single basis for the ambient space. Such a group can be easily detected, because the number of bits per vector in that group is very large relative to other groups. However, in higher-dimensional spaces, such as in our motion segmentation problem, outliers are more sparsely distributed. Hence, it is more efficient to code them by representing each outlier as a separate group. Such small groups are also easily detectable.

Figure 1. The motions sequences "1R2RC" (left), "arm" (center), and "cars10"(right) from the Hopkins155 database [18].

Experiments. For all of the experiments in Section 2, we choose three representative sequences from the Hopkins155 motion segmentation database [18] for testing: "1R2RC" (checkerboard), "arm" (articulation), and "cars10" (traffic) (see Figure 1). We compare the robustness to outliers of ALC and Local Subspace Affinity (LSA) [22], a spectral clustering-based motion segmentation algorithm that is reasonably robust to outliers. We add between 0% and 25% outlying trajectories to the dataset of a given motion sequence. Outlying trajectories were generated by choosing a random initial point in the first frame, and then performing a random walk through the following frames. Each increment is generated by taking the difference between the coordinates of a randomly chosen point in two randomly chosen consecutive frames. In this way the outlying trajectories will qualitatively have the same statistical properties as the other trajectories, but will not obey to any particular motion model. We then input these outlier-ridden datasets into LSA and ALC, respectively, and compute the misclassification rate and outlier detection rate for both algorithms.[4] For each experiment we run 100 trials with different randomly generated outlying trajectories. Table 1 shows the average misclassification rates and outlier detection rates for each experiment. As the results show, ALC can easily detect outliers without hindering motion segmentation, whereas for LSA, the outliers tend to interfere with the classification of valid trajectories.

Table 1. Top: Misclassification rates for LSA and ALC as a function of the outlier percentage (from 0% to 25%) for three motion sequences. Bottom: Outlier Detection rates for LSA and ALC as a function of the outlier percentage for three motion sequences.

	1R2RC [%]		arm [%]		cars10 [%]	
[%]	LSA	ALC	LSA	ALC	LSA	ALC
0	2.40	1.09	22.08	0.00	16.84	1.34
7	6.91	1.29	24.17	0.13	31.97	0.40
15	3.09	1.31	15.38	0.06	26.43	0.19
25	2.69	1.16	10.25	0.04	24.59	0.17

	1R2RC [%]		arm [%]		cars10 [%]	
[%]	LSA	ALC	LSA	ALC	LSA	ALC
0	98.04	100	77.9	100	86.87	100
7	94.75	99.99	92.79	100	96.82	99.70
15	98.04	99.98	91.34	100	98.84	99.81
25	98.20	99.97	95.56	100	98.76	99.83

[4] In ALC a trajectory is labeled an outlier if it belongs to a group with less than five samples. In our implementation of LSA, a trajectory is labeled as an outlier if its distance from the nearest motion subspace is greater than a predetermined threshold.

2.2. Incomplete Trajectories

In practice, due to occlusions or limitations of the tracker, some features may be missing in some frames. This can lead to incomplete trajectories. However by harnessing the low rank subspace structure of the data set, it is possible to complete these trajectories prior to subspace separation.

The key observation is that samples drawn from a low-dimensional linear subspace are self-expressive, meaning that a sample can be expressed in terms of a few other complete samples from the same linear subspace. More precisely, if the given incomplete sample is $\boldsymbol{y} \in \mathbb{R}^D$ and $\mathtt{Y} \in \mathbb{R}^{D \times P}$ is the matrix whose columns are all the complete samples in the data set, then there exists a coefficient vector $\boldsymbol{c} \in \mathbb{R}^P$ that satisfies

$$\boldsymbol{y} = \mathtt{Y}\boldsymbol{c}. \tag{7}$$

As the number of samples P is usually much greater than the dimension of the ambient space D, (7) is a highly underdetermined system of linear equations, and so, in general, $\boldsymbol{c}$ is not unique. In fact, any D vectors in the set that span $\mathbb{R}^D$ can serve as a basis for representing $\boldsymbol{y}$. However, since $\boldsymbol{y}$ lies in a low-dimensional linear subspace, it can be represented as a linear combination of only a few vectors from the same subspace. Hence, its coefficient vector should have only a few nonzero entries corresponding to vectors from the same subspace. Thus, what we seek is the *sparsest* $\boldsymbol{c}$:

$$\boldsymbol{c}^* = \underset{\boldsymbol{c}}{\operatorname{argmin}} \|\boldsymbol{c}\|_0 \text{ subject to } \boldsymbol{y} = \mathtt{Y}\boldsymbol{c}, \tag{8}$$

where $\|\cdot\|_0$ is the "ℓ^0 norm", equal to the number of nonzero entries in the vector. The sparsest coefficient vector $\boldsymbol{c}^*$ is unique when $\|\boldsymbol{c}^*\|_0 < D/2$. In the general case, ℓ^0 minimization, like MRM, is known to be NP-Hard [5]. Fortunately, due to the findings of Donoho *et al*. [3], it is known that if $\boldsymbol{c}^*$ is sufficently sparse (*i.e*. $\|\boldsymbol{c}^*\|_0 \lesssim \lfloor \frac{D+1}{3} \rfloor$), then the ℓ^0 minimization in (8) is equivalent to the following ℓ^1 minimization:

$$\boldsymbol{c}^* = \underset{\boldsymbol{c}}{\operatorname{argmin}} \|\boldsymbol{c}\|_1 \text{ subject to } \boldsymbol{y} = \mathtt{Y}\boldsymbol{c}, \tag{9}$$

which is essentially a linear program.

We apply these results to the problem of dealing with incomplete data. We assume that we have a set of samples $\mathtt{Y} \in \mathbb{R}^{D \times P}$ on the N subspaces with *no* missing entries, and we use $\mathtt{Y}$ to complete each sample with missing entries *individually*. Suppose $\boldsymbol{y} \in \mathbb{R}^D$ is a sample with missing entries $\{y_i\}_{i \in I}, I \subset \{1, \ldots, D\}$. Let $\hat{\boldsymbol{y}} \in \mathbb{R}^{D-|I|}$ and $\hat{\mathtt{Y}} \in \mathbb{R}^{(D-|I|) \times P}$ be $\boldsymbol{y}$ and $\mathtt{Y}$ with the rows indexed by I removed, respectively. By removing these rows, we are essentially projecting the data onto the $(D-|I|)$-dimensional subspace orthogonal to span$(\{\boldsymbol{e}_i : i \in I\})$.[6] With probability one, an arbitrary d-dimensional projection preserves the structural relationships between the subspaces, as long as the dimension of each subspace is strictly less than d. Thus if we solve the linear program:[7]

$$\boldsymbol{c}^* = \underset{\boldsymbol{c}}{\operatorname{argmin}} \|\boldsymbol{c}\|_1 \text{ subject to } \hat{\boldsymbol{y}} = \hat{\mathtt{Y}}\boldsymbol{c}, \tag{10}$$

then the completed vector $\boldsymbol{y}^*$ can be recovered as

$$\boldsymbol{y}^* = \mathtt{Y}\boldsymbol{c}^*. \tag{11}$$

In the literature there are many methods for filling in missing entries of a low rank matrix [10, 9, 14]. It is important to note that low rank matrix completion is quite different from our task here. For a matrix with low column rank, the problem of completing missing data is *overdetermined*. Thus algorithms like Power Factorization (PF) [20] essentially solve for the missing entries in a least-squares (minimum ℓ^2 norm) sense to preserve the low rank of the matrix. However, data drawn from a union of subspaces will, in general, be full rank – the matrix $\hat{\mathtt{Y}}$ is often over-complete. As such, the problem becomes instead *underdetermined* so there is no unique solution for the values of the missing entries. Our method then chooses the unique solution with the minimum ℓ^1 (and hence minimum ℓ^0) norm. The vector with missing entries is represented by the fewest possible complete vectors, which will in general be from only one of the subspaces. On the other hand, the least-squares (ℓ^2) solution found with Power Factorization is typically not sparse [3]. Table 2 compares our method to Power Factorization suggested in [20] for motion segmentation with missing data. As we see, in the case when the problem is underdetermined, the ℓ^1 solution indeed gives a much more accurate completion for the missing entries.

Table 2. Average errors over 100 trials in pixels per missing entry for Power Factorization (with rank 5) [20] and our ℓ^1-based feature completion method on the same three motion sequences used in the previous experiment (Figure 1). For each trial, 10% of the entries of the data matrix are removed and we use 75% of the complete trajectories to fill in the missing entries.

1R2RC [pixel]		arm [pixel]		cars10 [pixel]	
PF	ℓ^1	PF	ℓ^1	PF	ℓ^1
0.177	0.033	8.568	0.070	0.694	0.212

Experiments. We now test the limits of our ℓ^1-based method for entry completion. In each trial, we randomly select a trajectory $\boldsymbol{y}_p$ from the dataset for a given sequence, and remove between 1 and $D-1 = 2F-1$ of its entries. We then apply (10) and (11) to recover the missing entries[8]. In order to simulate many trajectories with missing entries in the dataset, we perform 5 different experiments. In each experiment, we use a portion (from 20% to 100%) of the remaining dataset to complete $\boldsymbol{y}_p$. Figure 2 (top) shows the results for 200 trials. For each sequence, we plot

[5] In fact, when MRM is applied to a set of *diagonal* matrices, it reduces to ℓ^0 minimization.

[6] $\boldsymbol{e}_i$ is the i-th vector in the canonical basis for $\mathbb{R}^D$.

[7] As suggested in [21], one can deal with noisy data by replacing the equality constraint in (10) with $\|\hat{\boldsymbol{y}} - \hat{\mathtt{Y}}\boldsymbol{c}\|_2 \leq \epsilon$.

[8] For all of our experiments that use ℓ^1-minimization, we use the freely available CVX toolbox for MATLAB [8].

the average per-entry error of the recovered trajectory w.r.t. the ground truth versus the percentage of missing entries in each incomplete trajectory. The different colored plots are for the experiments with varying percentage of the dataset used for completion. We see that for all motion sequences, our method is able to reconstruct trajectories to within subpixel accuracy even with over 80% of the entries missing! We also see that the performance remains consistent even when the entries are completed with small subsets of the remaining data. This suggests that our method can work well even if a large number of trajectories have missing features.

2.3. Corrupted Trajectories

Corrupted entries can be present in a trajectory when the tracker unknowingly loses track of feature points.[9] Such entries contain gross errors. One could treat corrupted trajectories as outliers.[10] However, in a corrupted trajectory, a portion of the entries still correspond to a motion in the scene, hence it seems wasteful to simply discard such information.

Repairing a vector with corrupted entries is much more difficult than the entry completion problem in Section 2.2, as now both the number and location of the corrupted entries in the vector are *not known*. Once again, by taking advantage of the low rank subspace structure of the dataset, we can both detect and repair vectors with corrupted entries *prior* to subspace separation. Our approach is similar to one proposed in [21] for robust face recognition.

A corrupted vector $\hat{\boldsymbol{y}}$ can be modeled as

$$\hat{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{e}, \tag{12}$$

where $\boldsymbol{y}$ is the uncorrupted vector, and $\boldsymbol{e} \in \mathbb{R}^D$ is a vector that contains all of the gross errors. We assume that there are only a few gross errors, so $\boldsymbol{e}$ will only have a few nonzero entries, and thus be sparse[11]. As long as there are enough uncorrupted vectors in the dataset, we can express $\boldsymbol{y}$ as a linear combination of the other vectors in the dataset as in Section 2.2. If $\mathtt{Y} \in \mathbb{R}^{P \times D}$ is a matrix whose columns are the other vectors in the dataset, and $\mathtt{I} \in \mathbb{R}^{D \times D}$ is an identity matrix, then (12) becomes

$$\hat{\boldsymbol{y}} = \mathtt{Y}\boldsymbol{c} + \boldsymbol{e} = \begin{bmatrix} \mathtt{Y} & \mathtt{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{e} \end{bmatrix} \doteq \mathtt{B}\boldsymbol{w}. \tag{13}$$

We would like both the coefficient vector $\boldsymbol{c}$ and the error vector $\boldsymbol{e}$ to be sparse[12]. If the true $\boldsymbol{c}$ and $\boldsymbol{e}$ are sufficiently sparse, we can simultaneously find the sparsest $\boldsymbol{c}$ and $\boldsymbol{e}$ by solving the linear program:[13]

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \|\boldsymbol{w}\|_1 \quad \text{subject to} \quad \hat{\boldsymbol{y}} = \mathtt{B}\boldsymbol{w}. \tag{14}$$

Once $\boldsymbol{w}^*$ is computed, we decompose it into $\boldsymbol{w}^* = \begin{bmatrix} \boldsymbol{c}^* & \boldsymbol{e}^* \end{bmatrix}^T$, where $\boldsymbol{c}^* \in \mathbb{R}^P$ is the recovered coefficient vector and $\boldsymbol{e}^* \in \mathbb{R}^D$ is the recovered error vector. The repaired vector $\boldsymbol{y}^*$ is simply

$$\boldsymbol{y}^* = \mathtt{Y}\boldsymbol{c}^*. \tag{15}$$

The error vector $\boldsymbol{e}^*$ also provides useful information. The nonzero entries of $\boldsymbol{e}^*$ are precisely the gross errors in $\hat{\boldsymbol{y}}$.

Experiments. We now test the limits of our ℓ^1-based method for repairing corrupted trajectories. For each trial in the experiments, we randomly select a trajectory $\boldsymbol{y}_p$ from the given dataset, and randomly *select and corrupt* between 1 and $D - 1 = 2F - 1$ entries in the vector. To corrupt the selected entries, we replace them with random values drawn from a uniform distribution. We then apply (14) and (15) to both detect the locations of corrupted entries, as well as repair them. In each experiment we run 200 trials and average the errors. We perform five experiments of this type, each with a portion (from 0% to 80%) of the remaining dataset $\mathtt{Y}$ being corrupted in the same way as $\boldsymbol{y}_p$. The results of these experiments are shown in Figure 2 (bottom). For each sequence, we plot the the average per-entry error of the repaired vector w.r.t. the ground truth versus the percentage of corrupted entries in each vector. The different colors represent experiments with varying portions of corrupted $\mathtt{Y}$. As Figure 2 (bottom) shows, this method is able to reconstruct vectors to within subpixel accuracy even with roughly 1/3 of the entries corrupted. This is in line with the bound $\|\boldsymbol{c}^*\|_0 < \lfloor \frac{D+1}{3} \rfloor$ given by [3]. We also see that the performance remains consistent even if 80% of the entire dataset is corrupted!

3. Large Scale Experiments

In this section, we perform experiments on the entire Hopkins155 database. We first discuss what modifications are needed to tailor ALC to the motion segmentation problem. We then compare our performance on the entire database versus some other motion segmentation algorithms. Finally, we do experiments on a set of motion sequences with real incomplete or corrupted trajectories.

3.1. Applying ALC to Motion Segmentation

ALC requires only a single parameter ε, the variance of the noise. However, the performance is also affected by the dimension that the original data is projected onto. Here we describe some methods for choosing these parameters.

Choosing ε. In principle, ε could be determined in some heuristic fashion from the statistics of the data. However,

[9] These kind of trajectories are called "intra-sample outliers" in [2].

[10] Indeed, if a dataset with some corrupted trajectories is input to ALC, the algorithm will classify those trajectories as outliers, as the gross errors will greatly increase the coding length of their ground-truth motion group.

[11] We realize that, in practice, trajectories may be corrupted by a large number of gross errors. However, it is unlikely that *any* method can repair such trajectories, and so it is best to treat them as outliers.

[12] The columns of $\mathtt{Y}$ should be scaled to have unit ℓ^2 norm to ensure that no vector is preferred in the sparse representation of $\boldsymbol{w}$.

[13] The presence of the identity submatrix $\mathtt{I}$ in $\mathtt{B}$ already renders the linear program stable to moderate noise.

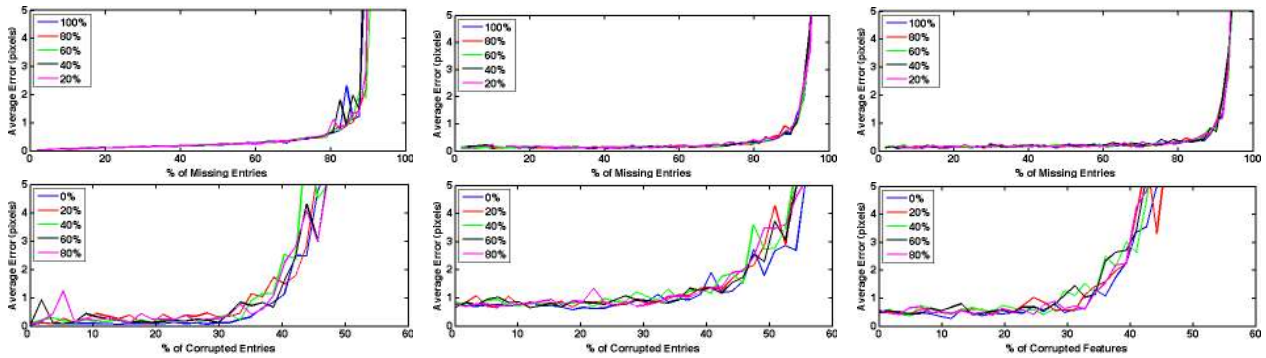

Figure 2. Errors of recovered trajectories for the sequences: "1R2RC" (left), "arm" (center), and "cars10" (right). Top: Results for our ℓ^1-based trajectory completion. The different colored plots are for experiments with varying percentage of the dataset used for completion. Bottom: Results for our ℓ^1-based detection and repair of corrupted trajectories. The different colors represent experiments with varying percentage of corrupted trajectories in the dataset.

most extant motion segmentation algorithms require the number of motions as a parameter. Thus, in order to make a fair comparison with other methods, we assume that the number of motions is given, and use it to determine ε.

Figure 3 shows an example sequence from the database. We run ALC on this sequence for several choices of ε. On the right we plot the misclassification rate and estimated group count as a function of ε. We see that the correct segmentation is stable over a fairly large interval. Using this observation, we developed the following voting scheme:

1. For a given motion sequence, run the algorithm multiple times over a number of choices of ε.[14]
2. Discard any ε that does not give rise to a segmentation with the correct number of groups.[15]
3. With the remaining choices of ε, find all the distinct segmentations that are produced.
4. Choose the ε that minimizes the coding length for the most segmentations, relative to the other choices of ε.

This scheme is quite simple, and by no means optimal, but as our experiments will show it works very well in practice.

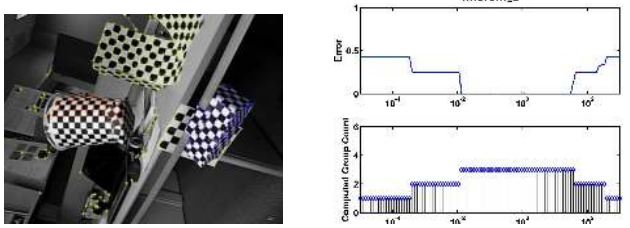

Figure 3. Left: The "1RT2TCRT_B" sequence from the Hopkins155 database. Right: The misclassification rate and estimated group count as a function of ε.

Choosing the Dimension of the Projection d. In general, Dimension Reduction improves the computational tractability of a problem. For example, for segmenting affine motions, [20] suggests projecting the trajectories onto a 5-dimensional subspace. However, for more complicated scenes (*e.g.* scenes with articulated motion), five dimensions may not be sufficient.

ALC scales roughly cubic with the dimension, so, in theory, we can leave our data in a relatively high-dimensional space. However, due to the greedy nature of the algorithm, a local minimum segmentation can be found if the samples do not adequately cover each subspace. Thus, Dimension Reduction can improve the results of ALC by increasing the density of samples within each subspace.

A balance needs to be struck between expressiveness and sample density. One choice, recently proposed in the sparse representation community [4], is the dimension d_{sp}:

$$d_{\text{sp}} = \min d \quad \text{subject to} \quad d \geq 2k \log(D/d),$$

where D is the dimension of the ambient space and k is the true low dimension of the data. It has been shown, that, asymptotically, as $D \to \infty$, this d is the smallest projection dimension such that the ℓ^1 minimization is still able to recover the correct sparse solutions. For our problem, using the affine camera model, we can assume that $k = 4$ and obtain a conservative estimate for a projection dimension d.

In our experiments, we test ALC with projection dimensions[16] $d = 5$ (as suggested in [20]), and the *sparsity-preserving* d stated above. We refer to the two versions of the algorithm as ALC_5 and ALC_{sp}, respectively.

3.2. Results on the Hopkins155 Database

The Hopkins155 database consists of 155 motion sequences categorized as checkerboard, traffic, or articulated. The motion sequences were obtained using an automatic tracker, and errors in tracking were manually corrected for each sequence. Thus in this experiment, there is no attempt to deal with incomplete or corrupted trajectories. See [18] for more details on the Hopkins155 database.

[14] Our experiments use 101 steps of ε logarithmically spaced in the interval $[10^{-5}, 10^{3}]$.

[15] If none of the choices of ε produce the right number of groups, we select the ε that minimizes the "penalized" coding length proposed in [15].

[16] We used Principal Component Analysis (PCA) as our method of Dimension Reduction.

We run ALC_5 and ALC_{sp} on the checkerboard, traffic, and articulated sequences using the voting scheme described earlier to determine ε. For each category of sequences, we compute the average and median misclassification rates, and the average computation times. We list these results in Tables 3-6 along with the reported results for Multi-Stage Learning (MSL) [13] and Local Subspace Affinity (LSA)[17] on the same database. Figure 4 gives two histograms of the misclassification rates over the sequences with two and three motions, respectively. There are several other algorithms that have been tested on the Hopkins155 database (*e.g.*, GPCA, RANSAC), but we list these two algorithms because they have the best reported misclassification rates in many categories of sequences.

As these results show, ALC performs well compared to the state-of-the-art. It has the best overall misclassification rate as well as for the checkerboard sequences. In categories where ALC is not the best, its performance is still competitive. The one notable exception is for the set of articulated sequences. In articulated sequences, it is difficult to track a lot of trajectories in each limb, but these trajectories live in a relatively high-dimensional space. Though in theory one only needs as many trajectories as the dimension of the subspace, we have observed experimentally that ALC can make suboptimal groupings when the ambient space is high-dimensional and the density of the data within a subspace is low. Finally, with regard to the projection dimension, our results indicate that, overall, ALC_{sp} performs better than ALC_5.

Table 3. Misclassification rates for sequences of two motions.

Checkerboard	MSL	LSA	ALC_5	ALC_{sp}
Average	4.46%	2.57%	2.66%	**1.55%**
Median	0.00%	0.27%	0.00%	0.29%
Traffic	MSL	LSA	ALC_5	ALC_{sp}
Average	2.23%	5.43%	2.58%	**1.59%**
Median	0.00%	1.48%	0.25%	1.17%
Articulated	MSL	LSA	ALC_5	ALC_{sp}
Average	7.23%	**4.10%**	6.90%	10.70%
Median	0.00%	1.22%	0.88%	0.95%
All Sequences	MSL	LSA	ALC_5	ALC_{sp}
Average	4.14%	3.45%	3.03%	**2.40%**
Median	0.00%	0.59%	0.00%	0.43%

3.3. Experimental Results on Robustness

We now test our robust subspace separation method on real motion sequences with incomplete or corrupted trajectories. We use the three motion sequences shown in Figure 5. These sequences are taken from [20] and are similar to the checkerboard sequences in Hopkins155. Each sequence contains three different motions and was split into three new sequences containing only trajectories from the first and second groups, first and third groups, and second and third groups, respectively. Thus, in total, we have twelve motion sequences, nine with two motions, and three with three motions. For these sequences, between 4% and 35% of the entries in the data matrix of trajectories are corrupted. These entries were manually located and labeled.

[17] For LSA we report the results for the version that projects the data onto a $4N$-dimensional space.

Table 4. Misclassification Rates for sequences of three motions.

Checkerboard	MSL	LSA	ALC_5	ALC_{sp}
Average	10.38%	5.80%	7.05%	**5.20%**
Median	4.61%	1.77%	1.02%	0.67%
Traffic	MSL	LSA	ALC_5	ALC_{sp}
Average	**1.80%**	25.07%	3.52%	7.75%
Median	0.00%	23.79%	1.15%	0.49%
Articulated	MSL	LSA	ALC_5	ALC_{sp}
Average	**2.71%**	7.25%	7.25%	21.08%
Median	2.71%	7.25%	7.25%	21.08%
All Sequences	MSL	LSA	ALC_5	ALC_{sp}
Average	8.23%	9.73%	**6.26%**	6.69%
Median	1.76%	2.33%	1.02%	0.67%

Table 5. Misclassification Rates over entire Hopkins 155 Database.

Checkerboard	MSL	LSA	ALC_5	ALC_{sp}
Average	5.94%	3.38%	3.76%	**2.47%**
Median	0.00%	0.57%	0.00%	0.31%
Traffic	MSL	LSA	ALC_5	ALC_{sp}
Average	**2.15%**	9.05%	2.76%	2.77%
Median	0.00%	1.96%	0.41%	1.10%
Articulated	MSL	LSA	ALC_5	ALC_{sp}
Average	6.53%	**4.58%**	7.58%	13.71%
Median	0.00%	1.22%	0.92%	3.46%
All Sequences	MSL	LSA	ALC_5	ALC_{sp}
Average	5.06%	4.87%	3.83%	**3.56%**
Median	0.00%	0.90%	0.27%	0.50%

Table 6. Average computation times for various algorithms.

Method	MSL	LSA	ALC_5	ALC_{sp}
Checkerboard	17h 40m	10.423s	12m 6s	24m 4s
Traffic	12h 42m	8.433s	8m 42s	17m 19s
Articulated	7h 35m	3.551s	4m 51s	10m 43s
All Sequences	19h 11m	9.474s	10m 32s	21m 3s

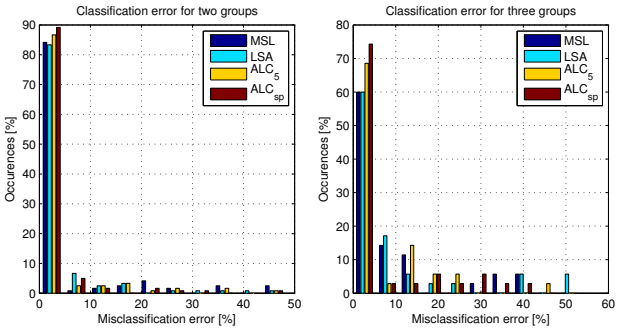

Figure 4. Misclassification rate histograms for various algorithms on the Hopkins155 database.

Incomplete Data. To see how ℓ^1-based entry completion affects the quality of segmentation, we remove the entries of

Figure 5. Example frames from three motion sequences with incomplete or corrupted trajectories. Sequences taken from [20].

trajectories that were marked as corrupted so that we may treat them as missing entries. We apply our ℓ^1-based entry completion method to this data, and input the completed data into ALC_5 and ALC_{sp}, respectively. For comparison, we also use Power Factorization to complete the data before segmentation. The misclassification rate for each sequence is listed in Table 7. The best overall results are for our ℓ^1-based method combined with ALC_{sp}. However, while Power Factorization combined with ALC_5 also performs competitively, its performance becomes much worse when combined with ALC_{sp}. These results give some empirical justification to our assertion that Power Factorization relies on the low rank of a matrix to recover missing entries.

Table 7. Misclassifications rates for Power Factorization and our ℓ^1-based approach on 12 real motion sequences with missing data.

Method	PF+ALC_5	PF+ALC_{sp}	ℓ^1+ALC_5	ℓ^1+ALC_{sp}
Average	1.89%	10.81%	3.81%	1.28%
Median	0.39%	7.85%	0.17%	1.07%

Corrupted Data. We also test our ability to repair corrupted trajectories, and observe the effects of the repair on segmentation. We simply apply our ℓ^1-based repair and detection method to the raw motion sequences, and then input the *repaired* data to ALC_5 and ALC_{sp}, respectively. The misclassification rate for each sequence is listed in Table 8. As the results show, our ℓ^1-based approach can repair corrupted trajectories to achieve reasonable segmentations.

Table 8. Misclassifications rates for our ℓ^1-based approach on 12 real motion sequences with corrupted trajectories.

Method	ℓ^1+ ALC_5	ℓ^1+ ALC_{sp}
Average	4.15%	3.02%
Median	0.21%	0.89%

4. Conclusion

In this paper we have developed a robust subspace separation method that applies Agglomerative Lossy Compression to the problem of motion segmentation. We showed that by properly exploiting the low rank nature of motion data, we can effectively deal with practical pathologies such as incomplete or corrupted trajectories. These techniques are in fact generic to subspace separation, and can conceivably be used in other application domains with little modification.

References

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

[2] F. De la Torre and M. J. Black. Robust principal component analysis for computer vision. In *ICCV*, pages 362–369, 2001.

[3] D. L. Donoho. For most large underdetermined systems of linear equations the minimal ℓ^1-norm solution is also the sparsest solution. preprint, *http://www-stat.stanford.edu/∼donoho/reports.html*, September 2004.

[4] D. L. Donoho and J. Tanner. Counting faces of randomly projected polytopes when the projection radically lowers dimension. preprint, *http://www.math.utah.edu/∼tanner/*, 2007.

[5] Z. Fan, J. Zhou, and Y. Wu. Multibody Grouping by Inference of Multiple Subspaces from High Dimensional Data Using Oriented-Frames. *IEEE TPAMI*, 28(1):90–105, Jan 2006.

[6] M. Fazel, H. Hindi, and S. Boyd. Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices. In *Proceedings of the American Control Conference*, pages 2156–2162, Jun 2003.

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

[8] M. Grant and S. Boyd. `CVX`: MATLAB software for disciplined convex programming [web page and software]. *http://www.stanford.edu/∼boyd/cvx/*, November 2007.

[9] A. Gruber and Y. Weiss. Multibody factorization with uncertainty and missing data using the EM algorithm. In *CVPR*, volume 1, pages 769–775, 2004.

[10] R. Hartley and F. Schaffalitzky. PowerFactorization: an approach to affine reconstruction with missing and uncertain data. *Australia-Japan Advanced Workshop on Computer Vision*, 2003.

[11] N. Ichimura. Motion segmentation based on factorization and discriminant criterion. In *ICCV*, pages 600–605, 1999.

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

[13] K. Kanatani and Y. Sugaya. Multi-state optimization for multi-body motion segmentation. *Australia-Japan Advanced Workshop on Computer Vision*, 2003.

[14] Q. Ke and T. Kanade. Robust ℓ^1-norm factorization in the presence of outliers and missing data by alternative convex programming. In *CVPR*, pages 739–746, 2005.

[15] Y. Ma, H. Derksen, W. Hong, and J. Wright. Segmentation of multivariate mixed data via lossy coding and compression. *IEEE TPAMI*, 29(9):1546–1562, September 2007.

[16] K. Schindler, D. Suter, and H. Wang. A model-selection framework for multibody structure-and-motion of image sequences. *IJCV*, 2008.

[17] P. Torr. Geometric motion segmentation and model selection. *Phil. Trans. Royal Society of London*, pages 1321–1340, 1998.

[18] R. Tron and R. Vidal. A benchmark for the comparison of 3-D motion segmentation algorithms. In *CVPR*, pages 1–8, 2007.

[19] R. Vidal, Y. Ma, and S. Sastry. Generalized Principal Component Analysis (GPCA). *IEEE TPAMI*, 27(12):1–15, 2005.

[20] R. Vidal, R. Tron, and R. Hartley. Multiframe motion segmentation with missing data using PowerFactorization and GPCA. *IJCV*, 2007.

[21] J. Wright, A. Y. Yang, A. Ganesh, Y. Ma, and S. S. Sastry. Robust Face Recognition via Sparse Representation. *to appear in IEEE TPAMI*, 2008.

[22] 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.

[23] A. Y. Yang, S. Rao, and Y. Ma. Robust statistical estimation and segmentation of multiple subspaces. In *CVPR Workshop on 25 Years of RANSAC*, 2006.

[24] L. Zelnik-Manor and M. Irani. Degeneracies, dependencies and their implications in multi-body and multi-sequence factorization. In *CVPR*, volume 2, pages 287–293, 2003.