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

Shankar 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. In practice, due to limitations of the tracker, occlusions and the presence of nonrigid objects in the scene, obtained motion trajectories may contain grossly mistracked features, missing entries, or not correspond to any valid motion model. 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.csi.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 more robust solution to the motion segmentation problem. But 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 $\{(x_{fp}, y_{fp})\}_{f=1}^{P}$ 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:

$$
Y = AX
$$

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

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

Thus the affine camera model postulates that trajectories of feature points from a single rigid motion will all lie in a linear subspace of $\mathbb{R}^{2P}$ 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 $\mathbb{R}^{2P}$, 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 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 $Y = [y_1, \ldots, y_P] \in \mathbb{R}^{D \times P}$ from $N$ rigidly moving objects in a dynamic scene, find a permutation $\Gamma$ of the columns of the data matrix $Y$:

$$Y = [Y_1 \ldots Y_N] \Gamma^{-1}, \quad (3)$$

such that the columns of each submatrix $Y_n$, $n = 1, \ldots, N$, 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 $Y$ 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) [18], are generic subspace separation algorithms that do not place any restriction on the relative orientations of the motion subspaces. However, the complexity of these methods grow exponentially with respect to both the dimension of the ambient space and the number of motions in the scene, and so they 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 [16, 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 [23, 21, 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, 22, 21, 19], 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.

Our methods use the affine camera model assumption, so we do not make any comparisons with perspective camera-based methods. 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 $Y$ into submatrices $\{Y_n\}_{n=1}^N$ such that each $Y_n$ 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 $\mathcal{M}$ 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 $\mathcal{M} \in \mathbb{R}^{D \times D}$, one can deal with both instability and computational intractability by minimizing the following smooth surrogate for rank($\mathcal{M}$):

$$J(\mathcal{M}, \delta) \triangleq \log_2 \det (\delta I + \mathcal{M}), \quad (4)$$

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

As we are not minimizing rank($Y_n$) 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 $Y_n \in \mathbb{R}^{D \times P_n}$ drawn from a linear subspace, the number of bits needed to code the data $Y_n$ up to distortion $\varepsilon^2$ [15] is given by

\[ \text{length of subspace-like data.} \]
\[ L(\gamma_n, \varepsilon) = \frac{D + P_n}{2} \left[ J \left( \frac{1}{P_n} \gamma_n \gamma_n^T, \frac{\varepsilon^2}{D} \right) - \log_2 \det \left( \frac{\varepsilon^2}{D} \right) \right] = \frac{D + P_n}{2} \log_2 \det \left( I + \frac{D}{P_n \varepsilon^2} \gamma_n \gamma_n^T \right). \]  

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

Now suppose the data matrix \( Y \in \mathbb{R}^{D \times P} \) can be partitioned into disjoint subsets \( Y = [Y_1 \ldots Y_N] \) of corresponding sizes \( P_1 + \cdots + P_N = P \). If we encode each subset separately, the total number of bits required is

\[ L^*(\{Y_1, \ldots, Y_N\}, \varepsilon) \doteq \sum_{n=1}^{N} L(\gamma_n, \varepsilon) - P_n \log_2 \frac{P_n}{P}. \]  

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 \( \varepsilon^2 \).

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 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 outliers in an elegant fashion. In [15], it was observed that in low dimensions (\( \leq 3 \)), a sufficient number of outliers will cover the entire space, and so the algorithm tends to group all outliers into a single group. Such a group can be easily detected, because the number of bits per vector in that group will be very large relative to other groups. However, in higher-dimensional spaces, such as in our motion segmentation problem, it would require an enormous number of outliers to fill the space. If outliers are thinly scattered in the ambient space, they will be most efficiently encoded when each outlier is its own group. Such small groups are also easily detectable.

**Experiments.** For all of the experiments in Section 2, we choose three representative sequences from the Hopkins155 motion segmentation database [17] for testing: “IR2RC” (checkerboard), “arm” (articulation), and “cars10” (traffic) (see Figure 1). We compare the robustness to outliers of ALC and Local Subspace Affinity (LSA) [21], 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. 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.

![Figure 1](image_url)  
Figure 1. The motions sequences “IR2RC” (left), “arm” (center), and “cars10” (right) from the Hopkins155 database [17].

### 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.

<table>
<thead>
<tr>
<th>IR2RC [%]</th>
<th>arm [%]</th>
<th>cars10 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LSA</td>
<td>ALC</td>
</tr>
<tr>
<td>7</td>
<td>6.91</td>
<td>1.29</td>
</tr>
<tr>
<td>15</td>
<td>3.09</td>
<td>1.31</td>
</tr>
<tr>
<td>25</td>
<td>2.69</td>
<td>1.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IR2RC [%]</th>
<th>arm [%]</th>
<th>cars10 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LSA</td>
<td>ALC</td>
</tr>
<tr>
<td>7</td>
<td>94.75</td>
<td>99.99</td>
</tr>
<tr>
<td>15</td>
<td>98.04</td>
<td>99.98</td>
</tr>
<tr>
<td>25</td>
<td>98.20</td>
<td>99.97</td>
</tr>
</tbody>
</table>

---

3In 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 image frames and 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 samples from the same linear subspace. More precisely, if the given sample is \( y \in \mathbb{R}^D \) and \( Y \in \mathbb{R}^{D \times P} \) is the data matrix whose columns are all of the other samples in the dataset, then there exists a coefficient vector \( c \in \mathbb{R}^P \) that satisfies

\[
y = Yc.
\]

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, \( c \) is not unique. In fact, any \( D \) vectors in the set that span \( \mathbb{R}^D \) can serve as a basis for representing \( y \). However, since \( y \) 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 \( c \):

\[
e^* = \arg\min_c \|c\|_0 \text{ subject to } y = Yc,
\]

where \( \|\cdot\|_0 \) is the "\( \ell^0 \) norm", equal to the number of nonzero entries in the vector. The sparsest coefficient vector \( e^* \) is unique when \( \|e^*\|_0 < D/2 \). In the general case, \( \ell^0 \) minimization, like MRM, is known to be NP-Hard \(^4\). Fortunately, due to the findings of Donoho et al. \([3]\), it is known that if \( e^* \) is sufficiently sparse (i.e., \( \|e^*\|_0 \lesssim \left\lfloor \frac{D+1}{3} \right\rfloor \)), then the \( \ell^0 \) minimization in (8) is equivalent to the following \( \ell^1 \) minimization:

\[
e^* = \arg\min_c \|c\|_1 \text{ subject to } y = Yc,
\]

which is essentially a linear program.

We apply these results to the problem of dealing with incomplete data. Suppose we have a sample \( y \in \mathbb{R}^D \) with missing entries \( \{y_i\}_{i \in I} \subseteq \{1, \ldots, D\} \) and a dataset \( Y \in \mathbb{R}^{D \times P} \) with no missing entries. Let \( \hat{y} \in \mathbb{R}^{D-|I|} \) and \( \hat{Y} \in \mathbb{R}^{(D-|I|) \times P} \) be \( y \) and \( Y \) 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 \( \text{span}(\{e_i: i \in I\}) \).\(^5\) As long as the dimension of each subspace is strictly less than \( d \), then an arbitrary \( d \)-dimensional projection preserves the structural relationships between the subspaces with probability one. Thus if we solve the linear program:\(^6\)

\[
e^* = \arg\min_c \|c\|_1 \text{ subject to } \hat{y} = \hat{Y}c,
\]

then the completed vector \( y^* \) can be recovered as

\[
y^* = Yc^*.
\]

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) \([19]\) essentially solve for the missing entries in a least-squares (minimum \( \ell^2 \) 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 \( Y \) is often overcomplete. 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 \( \ell^1 \) (and hence minimum \( \ell^0 \)) norm. The vector with missing entries is represented by fewest possible complete vectors, which will in general be from only one of the subspaces. On the other hand, the least-squares (\( \ell^2 \)) solution found with Power Factorization is typically not sparse \([3]\). Table 2 compares our method to Power Factorization suggested in \([19]\) for motion segmentation with missing data. As we see, in the case when the problem is underdetermined, the \( \ell^1 \) solution indeed gives a much more accurate completion for the missing entries.

<table>
<thead>
<tr>
<th>1R2RC [pixel]</th>
<th>arm [pixel]</th>
<th>cars10 [pixel]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{PF} ) | ( \ell^1 ) | ( \ell^2 ) | ( \text{PF} ) | ( \ell^1 ) | ( \ell^2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.177</td>
<td>0.033</td>
<td>8.568</td>
</tr>
</tbody>
</table>

Experiments. We now test the limits of our \( \ell^1 \)-based method for entry completion. In each trial, we randomly select a trajectory \( y_p \) 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\(^7\). 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 \( y_p \). Figure 2 (top) shows the results for 200 trials. For each sequence, we plot

\(^4\) In fact, when MRM is applied to a set of diagonal matrices, it reduces to \( \ell^0 \) minimization.

\(^5\) \( e_i \) is the \( i \)-th vector in the canonical basis for \( \mathbb{R}^D \).

\(^6\) For noisy trajectories, one may replace the equality constraint in (10) with \( \|\hat{y} - \hat{Y}c\|_2 \leq \epsilon \).

\(^7\) For all of our experiments that use \( \ell^1 \)-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 sub-pixel 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. Such entries are gross errors that could have arbitrary magnitude. One could treat corrupted trajectories as outliers. However, in a corrupted trajectory, as a portion of the entries still correspond to a motion in the scene, 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 [20] for robust face recognition.

A corrupted vector \( \hat{y} \) can be modeled as

\[
\hat{y} = y + e, \tag{12}
\]

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

\[
\hat{y} = Yc + e = [Y, I][c \mid e] = Bw. \tag{13}
\]

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

\[
u^* = \arg\min_{w} \|w\|_1 \quad \text{subject to} \quad \hat{y} = Bw. \tag{14}
\]

Once \( u^* \) is computed, we decompose it into \( u^* = [e^*, c^*]^T \), where \( c^* \in \mathbb{R}^p \) is the recovered coefficient vector and \( e^* \in \mathbb{R}^D \) is the recovered error vector. The repaired vector \( \hat{y}^* \) is simply

\[
y^* = \gamma c^*. \tag{15}
\]

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

Experiments. We now test the limits of our \( \ell^1 \)-based method for repair of corrupted trajectories. For each trial in the experiments, we randomly select a trajectory \( y_p \), 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 \( Y \) being corrupted in the same way as \( y_p \). 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 \( Y \). 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 \( ||c^*||_0 < (\frac{DP}{3}) \) 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 \( \varepsilon \), 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.

\(^{12}\) The presence of the identity submatrix \( I \) in \( B \) already renders the linear program stable to moderate noise.
Choosing $\varepsilon$. In principle, $\varepsilon$ could be determined in some heuristic fashion from the statistics of the data. However, 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 $\varepsilon$.

Figure 3 shows an example sequence from the database. We run ALC on this sequence for several choices of $\varepsilon$. On the right we plot the misclassification rate and estimated group count as a function of $\varepsilon$. 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 $\varepsilon$.
2. Discard any $\varepsilon$ that does not give rise to a segmentation with the correct number of groups.
3. With the remaining choices of $\varepsilon$, find all the distinct segmentations that are produced.
4. Choose the $\varepsilon$ that minimizes the coding length for the most segmentations, relative to the other choices of $\varepsilon$.

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

Choosing the Dimension of the Projection $d$. In general, Dimension Reduction improves the computational tractability of a problem. For example, for segmenting affine motions, [19] 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 making the subspaces more dense with samples.

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

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

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 $\ell^1$ 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 $d = 5$ (as suggested in [19]), and the sparsity-preserving $d$ stated above. We refer to the two versions of the algorithm as ALC$^5$ and ALC$^\text{sp}$, 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

---

13Our experiments use 101 steps of $\varepsilon$ in the interval $[10^{-5}, 10^5]$.

14If none of the choices of $\varepsilon$ produce the right number of groups, we select the $\varepsilon$ that minimizes the “penalized” coding length proposed in [15].

15We used Principal Component Analysis (PCA) as our method of Dimension Reduction.
each sequence. Thus in this experiment, there is no attempt
to deal with incomplete or corrupted trajectories. See [17]
for more details on the Hopkins155 database.

We run ALC$_5$ and ALC$_{sp}$ on the checkerboard, traf-
cic, and articulated sequences using the voting scheme
described earlier to determine $\varepsilon$. For each category of se-
quencies, we compute the average and median misclassifica-
tion 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)$^{16}$ 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 (GPCA, RANSAC etc.), but we list these two al-
gorithms because they have the best reported misclassifica-
tion 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 misclassifica-
tion rate as well as for the checkerboard sequences. In categories
where ALC is not the best, its performance is still com-
petitive. 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 sub-
space, we have observed experimentally that ALC can make
suboptimal groupings when dealing with high dimensional
subspaces that are not well sampled. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>MSL</th>
<th>LSA</th>
<th>ALC$_5$</th>
<th>ALC$_{sp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.46%</td>
<td>2.57%</td>
<td>2.56%</td>
<td>1.49%</td>
</tr>
<tr>
<td>Median</td>
<td>0.00%</td>
<td>0.27%</td>
<td>0.00%</td>
<td>0.27%</td>
</tr>
<tr>
<td>Traffic</td>
<td>2.23%</td>
<td>5.43%</td>
<td>2.83%</td>
<td>1.75%</td>
</tr>
<tr>
<td>Median</td>
<td>0.00%</td>
<td>1.48%</td>
<td>0.30%</td>
<td>1.51%</td>
</tr>
<tr>
<td>Articulated</td>
<td>7.23%</td>
<td>4.10%</td>
<td>6.90%</td>
<td>10.70%</td>
</tr>
<tr>
<td>Median</td>
<td>0.00%</td>
<td>1.22%</td>
<td>0.89%</td>
<td>0.95%</td>
</tr>
<tr>
<td>All Sequences</td>
<td>4.14%</td>
<td>3.45%</td>
<td>3.03%</td>
<td>2.40%</td>
</tr>
</tbody>
</table>

3.3. Experimental Results on Robustness

We now test our robust subspace separation method on
real motion sequences with incomplete or corrupted trajec-
tories. We use the three motion sequences shown in Fig-
ure 5. These sequences are taken from [19] and are simi-
lar to the checkerboard sequences in Hopkins155. Each

$^{16}$For LSA we report the results for the version that projects the data
onto a 4N-dimensional space.
ruptured. These entries were manually located and labeled.

Figure 5. Example frames from three motion sequences with incomplete or corrupted trajectories. Sequences taken from [19].

Incomplete Data. To see how \( \ell^1 \)-based entry completion affects the quality of segmentation, we remove the entries of trajectories that were marked as corrupted so that we may treat them as missing entries. We apply our \( \ell^1 \)-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 \( \ell^1 \)-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 \( \ell^1 \)-based approach on 12 real motion sequences with missing data.

<table>
<thead>
<tr>
<th>Method</th>
<th>PF+ALC(_5)</th>
<th>PF+ALC(_{Sp})</th>
<th>( \ell^1 )+ALC(_5)</th>
<th>( \ell^1 )+ALC(_{Sp})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>1.89%</td>
<td>10.81%</td>
<td>3.81%</td>
<td>1.28%</td>
</tr>
<tr>
<td>Median</td>
<td>0.39%</td>
<td>7.85%</td>
<td>0.17%</td>
<td>1.07%</td>
</tr>
</tbody>
</table>

Corrupted Data. We also test our ability to repair corrupted trajectories, and observe the effects of the repair on segmentation. We simply apply our \( \ell^1 \)-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 \( \ell^1 \)-based approach can repair corrupted trajectories to achieve reasonable segmentations.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>( \ell^1 )+ALC(_5)</th>
<th>( \ell^1 )+ALC(_{Sp})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.15%</td>
<td>3.02%</td>
</tr>
<tr>
<td>Median</td>
<td>0.21%</td>
<td>0.89%</td>
</tr>
</tbody>
</table>

4. Conclusion

In this paper we develop a robust subspace separation method that applies Agglomerative Lossy Compression to the problem of motion segmentation. We show 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