Towards A Framework For Simultaneous Feature Tracking And Segmentation

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Bryan Poling

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Gilad Lerman

May, 2016

Acknowledgements

I would like to thank my adviser Gilad Lerman for his instruction, guidance, and enduring support. This work was supported by NSF CAREER award of Professor Lerman: DMS-09-56072 and Doctoral Dissertation Fellowship of Bryan Poling. Professor Lerman contributed significantly to this work. I would like to thank my instructors and fellow students for their instruction and support, especially Samantha Schumacher, for entertaining my endless skepticism and ultimately deepening my understanding of mathematics.

Dedication

To my parents.

Abstract

This is a collection of several works that I have done during my PhD research as a graduate student at the University of Minnesota. There are three parts, each focusing on a different topic in machine learning and computer vision. The common theme underlying these works is the tracking of feature points in motion video and their segmentation by object. Abstracts for each part are included below.

Abstract for Part I

We present a novel approach to rigid-body motion segmentation from two views. We use a previously developed nonlinear embedding of two-view point correspondences into a 9-dimensional space and identify the different motions by segmenting lower-dimensional subspaces. In order to overcome mixed and unknown dimensions of subspaces and nonuniform distributions along them we suggest the novel concept of global dimension and its minimization for clustering subspaces with some theoretical motivation. We propose a fast projected gradient algorithm for minimizing global dimension and thus segmenting motions from 2-views. We develop an outlier detection framework around the proposed method, and we present state-of-the-art results on outlier-free and outlier-corrupted two-view data for segmenting motion.

Abstract for Part II

In this part, we present a framework for jointly tracking a collection of features in motion video, which enables sharing information between the different features in the scene. Our method exploits the fact that trajectories of features from locally rigid and semi-rigid scenes are approximately confined to low-dimensional subspaces, and it uses this fact to aid the tracking of poor-quality and non-corner-like features by filling in missing information from other, better feature points. Our method significantly improves tracking performance in real-world, poorly lit scenes, does not require explicit modeling of the structure or motion of the scene, and runs in real time on a single CPU core.

Abstract for Part III

In this part, we employ low-cost gyroscopes to improve general purpose feature tracking. We use some of the same ideas from Part II, except instead of borrowing information from other features to aid the tracking of poor-quality features, we rely on independent estimates of optical flow from external inertial sensors. Most related previous methods use gyroscopes to initialize and bound the search for features. In contrast, we use them to regularize the tracking energy function so that they can directly assist in the tracking of ambiguous and poor-quality features. We demonstrate that our simple technique offers significant improvements in performance over conventional template-based tracking methods, and is in fact competitive with more complex and computationally expensive state-of-the-art trackers, but at a fraction of the computational cost. Additionally, we show that the practice of initializing template-based feature trackers like KLT (Kanade-Lucas-Tomasi) using gyro-predicted optical flow offers no advantage over using a careful optical-only initialization method, suggesting that some deeper level of integration, like the method we propose, is needed in order to realize a genuine improvement in tracking performance from these inertial sensors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Feature tracking is the task of determining and maintaining the location of one or more visually interesting points as they move about in motion video. This task is crucial in computer vision, where it is frequently used as a first step in solutions to important problems like simultaneous localization and mapping (SLAM) and structure from motion (SFM). Feature-based motion segmentation, on the other hand, is the task of segmenting a set of tracked features into different groups corresponding to different moving objects in a scene.

In feature tracking, the input is raw video and the output is a collection of *feature trajectories*, each recording the position of a single feature in a sequence of video frames. In feature-based motion segmentation, one takes a set of feature trajectories as the input, and the output is a collection of labels identifying which trajectories correspond to common objects in the scene (two features belong to the same object if they have the same label). Traditionally, these are treated as distinct problems, and they are handled in a pipeline; feature tracking is performed first, and the resulting trajectories are then segmented.

One of the dominant approaches to feature-based motion segmentation is to exploit the fact that trajectories of features corresponding to a single rigid body are confined to a low-dimensional subspace. Moving objects can therefore be identified by clustering trajectories into low-dimensional subspaces. This process is called subspace clustering, or hybrid linear modeling, and it is at the heart of part I of this thesis. It is possible, however, to use the same observation in a different way. Instead of using the subspace

property to segment trajectories, you can use it to improve feature tracking when you know that a collection of features belongs to a rigid or semi-rigid object. In this situation, you know that the trajectories should live in a subspace, so you can use that fact to help resolve trajectories when some of them may otherwise be difficult to measure. This is the focus of part II of this thesis, and we present a rather powerful technique for accomplishing this goal. In fact, we find that by weakly encouraging trajectories to fit into a subspace, the method can even improve tracking when the assumption that the features belong to a rigid or semi-rigid object is violated.

The fact that the subspace property can be used to improve feature tracking, especially when features are known to belong to a common object, or it can be used to segment features by object affiliation if tracking is taken for granted, suggests value in combining the tracking and segmentation problems and handling them together in a single framework. The subspace property can be used to segment features, and this segmentation can be used to exploit the subspace property again to improve the tracking of feature cohorts belonging to each object in a scene. This is the ultimate goal of my work, and each part of this thesis focuses on some aspect of feature tracking or segmentation, with an eye towards ultimately developing a unified approach to both problems.

- Part I presents a new subspace clustering algorithm, specifically tailored to feature-based motion segmentation, called Global Dimension Minimization (GDM).
- Part II covers an approach to using the subspace property for feature trajectories to improve general-purpose feature tracking.
- Part III presents an alternative approach to improving feature tracking by bringing in additional information from external inertial sensors. This part uses some of the same ideas as part II, but it exploits a different source of information.
- We conclude with a discussion about developing a common framework for simultaneous feature tracking and segmentation. We discuss some of the lessons learned in parts I - III and what they can tell us about the development of such a framework.

Chapter 2

Part I: Two-View Motion Segmentation Using Global Dimension Minimization

Based on work done with Gilad Lerman

2.1 Introduction

A classic problem in computer vision is that of feature-based motion segmentation from two views. In this problem one has two images, taken at different times, of a 3D scene. The scene is assumed to consist of multiple, independently moving rigid bodies. The goal is to identify the different moving objects and estimate a motion model for each one of them. For this purpose, one automatically tracks the locations of visually-interesting "features" in the scene, which are visible in both views (e.g., via Lucas Kanade type algorithm [1]). Each feature is represented as a pair of 2-vectors, holding the image coordinates of the feature in the two different views; such a pair is referred to as a *point correspondence*. The mathematical problem of feature-based, two-view motion segmentation is to both segment the point correspondences according to the rigid objects to which they belong, and estimate a motion model for each object.

A basic strategy to solve the feature-based two-view motion segmentation problem

is to first cluster point correspondences and then estimate the single-body motions within clusters (well-known methods for single-body motion estimation are described in [2, 3]). This procedure was suggested in [4], while clustering point correspondences with K-means or spectral clustering, and in [5], while alternating between clustering and motion segmentation via an EM procedure. Both clustering strategies of [4] and [5] are based primarily on spatial separation between the clusters, however, different clusters in this setting may intersect each other (e.g., when motions share a symmetry).

Due to this problem, some algebraic methods have been developed for directly solving for the motion parameters, while eliminating the clustering of point correspondences [6, 7]. Another solution is to segment feature trajectories by taking into account their geometric structures, which may be different than spatial separation (the feature trajectories in 2-views are the 4-dimensional vectors concatenating the 2 point correspondences of the same feature from 2 views). Costeira and Kanade [8] showed that under the affine camera model, feature trajectories in n-views (for $n \geq 2$ these are vectors of length $2n$) within each rigid body lie on an affine subspace of dimension at most 3. This observation has given rise to several feature-based motion segmentation schemes, which are based on clustering subspaces [9]; we refer to such clustering as Hybrid Linear Modeling (HLM). For the more general and realistic model of the perspective camera, it can be shown that feature trajectories from two-views lie on quadratic surfaces of dimension at most 3 (in $\mathbb{R}^4$) (see §2).

Arias-Castro et al. [10] suggested clustering the quadratic surfaces of point correspondences (in $\mathbb{R}^4$) using Higher Order Spectral Clustering (HOSC) for manifold clustering. They demonstrated competitive results on the outlier-free database of [7], when assuming that the clusters are of dimension 2. However, their results are not competitive for incorporating dimension 3 and they did not provide any numerical evidence that the dimension of the surfaces was 2 and not 3.

A different approach for clustering these particular quadratic surfaces can be obtained by embedding point correspondences into "quadratic coordinates" and then clustering subspaces. More precisely, if a point correspondence $((x, y), (x', y'))$ is mapped into $(x, y, 1) \otimes (x', y', 1) \in \mathbb{R}^9$, where $\otimes$ denotes the Kronecker product, then these quadratic surfaces are mapped into linear subspaces of dimensions at most 8, which are determined by the fundamental matrices [2, 3, 11] of the different motions. Chen

et al. [11] have used this idea for clustering such quadratic mappings of point correspondences by the Spectral Curvature Clustering (SCC) algorithm [12] (they showed that instead of performing the actual mapping, one can apply the kernel trick). They claimed that other HLM algorithms (at that time) did not work well for such embedded data.

The drawback of applying SCC to this quadratic mapping of point correspondences in $\mathbb{R}^9$ is that SCC does not work well with subspaces of mixed dimensions, and the subspace dimensions must be known a-priori. Unfortunately, the subspaces in this application have mixed and unknown dimensions (see §2.2). What makes SCC successful for this application is the fact that it takes into account some global information of the subspaces (i.e., for d-dimensional subspaces it uses affinities based on arbitrary $d+2$ points, and in particular, far-away points). This helps SCC deal with nonuniform sampling along subspaces with local structure very different than the global one (see §2.2). On the other hand, local methods (e.g., [13, 14]) often do not work well in this setting.

The purpose of this paper is to develop an HLM algorithm that can successfully cluster the quadratically-embedded point correspondences in $\mathbb{R}^9$. In particular, it exploits the global structure of the underlying subspaces, i.e., their "dimensions".

For this purpose, we propose a class of empirical dimension estimators, and a corresponding notion of global dimension for a mixture of subspaces (a function of the estimated dimensions of its constituent parts). We propose the global dimension minimization (GDM) algorithm, which is a fast projected gradient method aiming to minimize the global dimension among all data partitions. We also build an outlier detection framework into this development to allow for corrupted data sets. We demonstrate state-of-the-art results for two-view motion segmentation (via quadratic embedding), both in the outlier-free and outlier-corrupted cases. We even show that these results are competitive with the state-of-the-art results for multiple-views, i.e., using all frames of a video sequence (obtained under the affine camera model). To motivate the use of global dimension, we prove that for special settings and choice of parameters, the global dimension is minimized by the correct partition of the data (representing the underlying subspaces). We then discuss what to do in more general settings.

The paper is organized as follows: §2.2 briefly explains how the problem of 2-view

motion segmentation can be formulated as a problem in HLM; §2.3 introduces global dimension and explains why its minimization can solve the HLM problem under some conditions; §2.4 develops a fast projected gradient method for minimizing global dimension; §2.5 develops an outlier detection/rejection framework for global dimension minimization; §2.6 demonstrates numerical results on real-world 2-view data sets for both outlier-removed and outlier-corrupted data; finally, §2.7 concludes this work. The appendix contains proofs of the key results in the paper.

2.2 Formulating 2-View Motion Segmentation as a Problem in HLM

One way of formulating the motion segmentation problem in terms of HLM is by exploiting the *Affine Motion Subspace*. Costeira and Kanade [8] demonstrated that when a set of features all come from a single rigid body, then under the assumptions of the affine camera model, the corresponding feature trajectories lie in an affine subspace of dimension 3 or less. One can use this fact to partition the set of features by clustering their trajectories into subspaces. This is a popular formulation of the segmentation problem, even when dealing with only two views of a scene.

The formulation involving the affine motion subspace has the advantage that the feature trajectories tend to be nicely distributed in their respective subspaces, and the different subspaces all have nearly the same dimensions. This formulation has the drawback that it requires an affine camera model. The affine camera assumption breaks down when viewing objects close to the camera, or when looking at objects at significantly different ranges. The consequence of this is that the trajectories from a rigid body do not lie within a subspace, but rather in a manifold which is only locally approximated by a subspace of dimension at most 3.

When dealing with 2-view segmentation, there is another approach, based on a more general camera model, which avoids this problem of distortion. This approach assumes a perspective camera, and relies on the fundamental matrix [2] for a rigid body.

Indeed, if $\mathbf{F} = (F_{i,j})_{i,j=1}^{3}$ is the fundamental matrix for a rigid body, and $\mathbf{x}_h =$

$(x, y, 1)^T$ and $\mathbf{x}'_h = (x', y', 1)^T$ together form a point correspondence (in standard homogenous coordinates) from that body, then

$$\mathbf{x}_h'^T \mathbf{F} \mathbf{x}_h = 0,$$

which is algebraically equivalent to

$$\text{vec}(\vec{F}) \cdot \boldsymbol{v} = 0, \tag{2.1}$$

where

$$\text{vec}(\vec{F}) = (F_{11}, F_{12}, F_{13}, F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33})^T$$

and

$$\boldsymbol{v} = (xx', x'y, x', xy', yy', y', x, y, 1)^T = (x, y, 1)^T \otimes (x', y', 1)^T.$$

We refer to the vector $\boldsymbol{v}$ as the *nonlinear* or *Kronecker* embedding of a point correspondence (recall that $\otimes$ is the Kronecker product). The vectors obtained through this nonlinear embedding for feature points on the same rigid object lie in a linear subspace of $\mathbb{R}^9$ of dimension at most 8. Indeed, (2.1) says that there is a vector $\text{vec}(\vec{F}) \in \mathbb{R}^9$ orthogonal to all of the feature trajectories in this set (it also shows that the linear embedding (x, y, x', y') lies on a 3-dimensional quadratic manifold). However, the subspace dimension can decrease due to two different reasons. First of all, if there are very few points (per motion), then they may span a lower-dimensional subspace. The second cause is degeneracy in the 3D configuration of the features. If all world points and both camera centers live on a ruled quadratic surface[1] , then their corresponding subspace has dimension 7 or less. In particular, if all world points (but not necessarily the camera centres) are coplanar, the corresponding subspace will have dimension no larger than 6 (see [2, pg. 296]). Therefore, to make use of this embedding, the hybrid-linear modeling algorithm being employed must be tolerant of subspaces of mixed dimension.

Since the perspective camera assumption is accurate in a much broader range of situations than the affine camera model, subspaces are more apparent with the nonlinear embedding than with the linear embedding. However, the nonlinear embedding distorts the original sampling and results in lower-dimensional structures (of dimension

[1] A surface S is ruled if through every point of S there exists a straight line that lies on S.

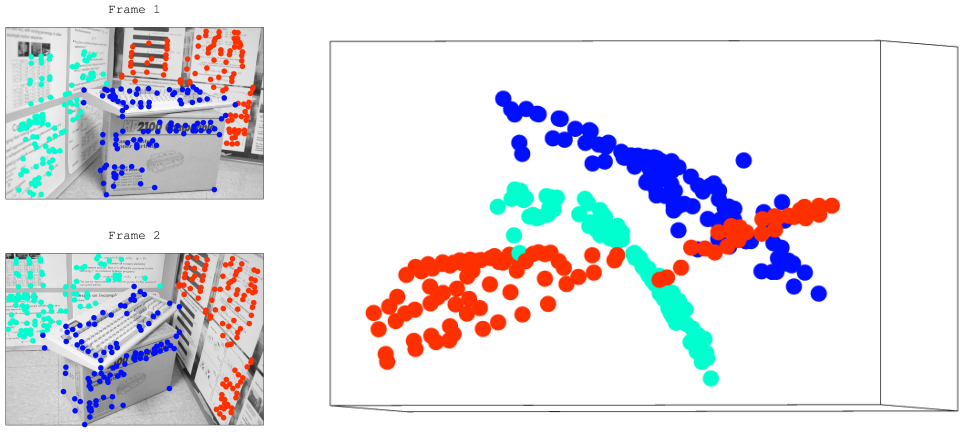


Figure 2.1: Two views of a 3D scene with features overlayed (left), and the non-linearly embedded point correspondences in $\mathbb{R}^9$, projected onto the 3-dimensional subspace spanned by their 3rd, 4th and 5th principal components (right).

at most 3) within the higher dimensional subspaces (of typical dimensions 6, 7 or 8), which is a serious obstacle for many HLM algorithms, especially ones using local spatial information.

2.3 Global Dimension

From here on, we will be considering 2-view motion segmentation under the perspective camera model, i.e. using the the Kronecker embedding. We will present a global HLM method, which is well-suited for handling the data which results from this embedding. We begin our development by providing some intuitive motivation for our approach.

Imagine that we have access to an oracle, who for any set of vectors in $\mathbb{R}^D$, can provide for us a good, robust estimate of the dimensionality of the set[2] . Now, suppose we have a set of vectors which are sampled from a hybrid-linear distribution. Consider a general partition of the data set, and define the "vector of set dimensions" for that partition to be the vector of oracle-provided approximate dimensions of each respective set in the partition. Our inspiration is the observation that for most partitions one may happen upon, each set in the partition will typically contain points from many of the

[2] In a noiseless case this would return the dimension of the linear span of the set of vectors

underlying subspaces. The associated vector of set dimensions will contain relatively large numbers, and the p-norm of this vector will be large. The p-norm of the vector of set dimensions will be referred to as the *global dimension* of the partition. The best way to make the global dimension small, it would seem, is to try and decrease all of the elements of the vector of set dimensions by grouping together vectors that come from common subspaces. This notion will be made precise and we will show, in fact, that under certain conditions, the *natural partition* of the data set (the one where point assignment agrees with subspace affiliation) is a global minimizer of the global dimension function.

Our approach to HLM will be to find the partition of a data set that yields the lowest possible global dimension. In this section, we develop the global dimension objective function in two parts. In §2.3.1 we suggest a new class of dimension estimators that can perform the role of the oracle in our discussion above. In §2.3.2 we define *global dimension* and explain why we expect its minimizer to reveal the clusters corresponding to the underlying subspaces. A fast algorithm for this minimization will be later described in §2.4.

2.3.1 On Empirical Dimension

We present here a class of dimension estimators depending on a parameter $\epsilon \in (0, 1]$. For $\boldsymbol{u} = (u_1, \ldots, u_k)$ and any $p > 0$, we use the notation $\|\boldsymbol{u}\|_p$ to mean $(u_1^p + \ldots + u_k^p)^{1/p}$ (even for $p = \epsilon < 1$, where $\|\cdot\|_p$ is not a norm). For a given set of N vectors in $\mathbb{R}^D$, $\{\boldsymbol{v}_i\}_{i=1}^N$, we denote by $\boldsymbol{\sigma} = (\sigma_1\ \sigma_2\ \ldots\ \sigma_{N\wedge D})^T$ the vector of singular values of the $D \times N$ data matrix $\boldsymbol{A}$ (the matrix whose columns are the data vectors).

For $\epsilon \in (0, 1]$ the *empirical dimension*, denoted by $\hat{d}_\epsilon(\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_N)$ (or simply $\hat{d}_\epsilon$) is defined by

$$\hat{d}_\epsilon(\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_N) := \frac{\|\boldsymbol{\sigma}\|_\epsilon}{\|\boldsymbol{\sigma}\|_{\left(\frac{\epsilon}{1-\epsilon}\right)}}. \tag{2.2}$$

When $\epsilon = 1$, this is sometimes called the "effective rank"[3] of the data matrix [16].

The following theorem explains why $\hat{d}_\epsilon$ is a good estimator for dimension. Put simply, it says two things. First, if we rotate and/or uniformly scale our set of vectors

[3] "Effective rank" is sometimes defined differently. See [15].

by a non-zero amount, then the empirical dimension of the set does not change. Second, in the absence of noise, empirical dimension never exceeds true dimension, but it approaches true dimension in the limit (as the number of measurements goes to infinity) for spherically symmetric distributions. From now on we refer to d-dimensional subspaces as d-subspaces.

Theorem 1 *For $\epsilon \in (0, 1]$, $\hat{d}_\epsilon$ possesses the following properties:*

1. *$\hat{d}_\epsilon$ is invariant under dilations (i.e., scaling).*
2. *$\hat{d}_\epsilon$ is invariant under orthogonal transformations.*
3. *If $\{\boldsymbol{v}_i\}_{i=1}^N$ are contained in a d-subspace of $\mathbb{R}^D$, then $\hat{d}_\epsilon \leq d$.*
4. *If $\{\boldsymbol{v}_i\}_{i=1}^N$ are i.i.d. samples from a sub-Gaussian probability measure, which is spherically symmetric within a d-subspace[4] and non-degenerate[5], then $\lim_{N\to\infty} \hat{d}_\epsilon(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = d$ with probability 1.*

To gain some intuition into the definition of empirical dimension, consider taking a large set of samples from a spherically symmetric distribution supported by a d-subspace. Call the covariance matrix for this distribution $\mathbf{Q}$. As the number of samples becomes large, the empirical covariance matrix approaches $\mathbf{Q}$, which has the first d elements on the main diagonal all equal (call the value α^2), and 0's everywhere else. The empirical dimension of the set of vectors involves the singular values of the data matrix, which are approaching the square roots of the eigenvalues of $\mathbf{Q}$. Hence, as the number of samples increases, we get:

$$\hat{d}_\epsilon(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) \to \frac{\|(\alpha, \alpha, ..., \alpha, 0, ..., 0)\|_\epsilon}{\|(\alpha, \alpha, ..., \alpha, 0, ..., 0)\|_{(\frac{\epsilon}{1-\epsilon})}} = \frac{d^{1/\epsilon}\alpha}{d^{(1-\epsilon)/\epsilon}\alpha} = d. \tag{2.3}$$

Thus, for any value of ϵ in $(0, 1]$, the empirical dimension approaches the true dimension of the set as the number of measurements increases.

If we look at a distribution that is not spherically symmetric, but still supported by a d-subspace, then empirical dimension tends to under-estimate the true dimension

[4] *A measure is spherically symmetric within a d-subspace if it is supported on this subspace and invariant to rotations within this subspace.*

[5] *A measure is non-degenerate on a subspace if it does not concentrate mass on any proper subspace. In our setting the measure is also assumed to be spherically symmetric, and this assumption is equivalent to assuming the measure does not concentrate at the origin.*

of the distribution, even as the number of samples approaches infinity. This is actually desirable behavior. If we take a spherically symmetric distribution in a d-subspace and imagine the process of collapsing it in one direction until it lies in a $(d-1)$-subspace, then true dimension behaves discontinuously. The true dimension of a large set of samples will equal d until the collapsing is complete; at that point the dimension will instantly drop to $d-1$. Empirical dimension smoothly drops from d to $d-1$ during this collapsing process. It is in this setting that we see the necessity of the parameter ϵ. This parameter controls how quickly the empirical dimension drops from d to $d-1$ in this process. More generally, a low value of ϵ results in a "strict" dimension estimator (meaning that it will not under-estimate dimension easily, even when distributions are asymmetric). When ϵ is large (approaching 1), empirical dimension is a lenient dimension estimator. It is much more tolerant of noise, but it may consequently under-estimate the dimension of highly asymmetric distributions. The trade-off is that when dealing with noisy data or distributions only approximately supported by linear subspaces, a stricter estimator can mistakenly interpret noise or distortion as energy in new directions, thereby causing an over-estimate of dimension. Numerical experiments (e.g., Fig. 2.2) have shown that values of ϵ between 0.3 and 0.7 seem to provide reasonable estimators, which tend to agree with our intuitive notion of dimension.

In our application, since the perspective camera model is reasonably accurate (as opposed to the affine model), the nonlinear embedding of point correspondences results in subspaces with rather negligible distortion. We can thus afford a low value of ϵ. In fact, this is needed because the data vectors are frequently distributed in very non-isotropic ways with this embedding. Thus, to avoid underestimating dimension, we choose $\epsilon = 0.35$, which lies just slightly above the lowest value we confirmed for ϵ (0.3). Notice that this value is not "tuned" to individual data sets, but is chosen based on the properties of the application as a whole and the nature of the embedding.

2.3.2 On Global Dimension

Assume we are provided a data set X in $\mathbb{R}^D$ (in our application $D = 9$ with the nonlinear embedding) and a partition of it $\Pi = (\Pi_1, \Pi_2, ..., \Pi_k)$ for some $k \in \mathbb{N}$ (i.e., $\{\Pi_i\}_{i=1}^k$ are disjoint subsets of X whose union is X). We also assume that X lies on a union of K subspaces and denote the "correct" (or natural) partition of the data

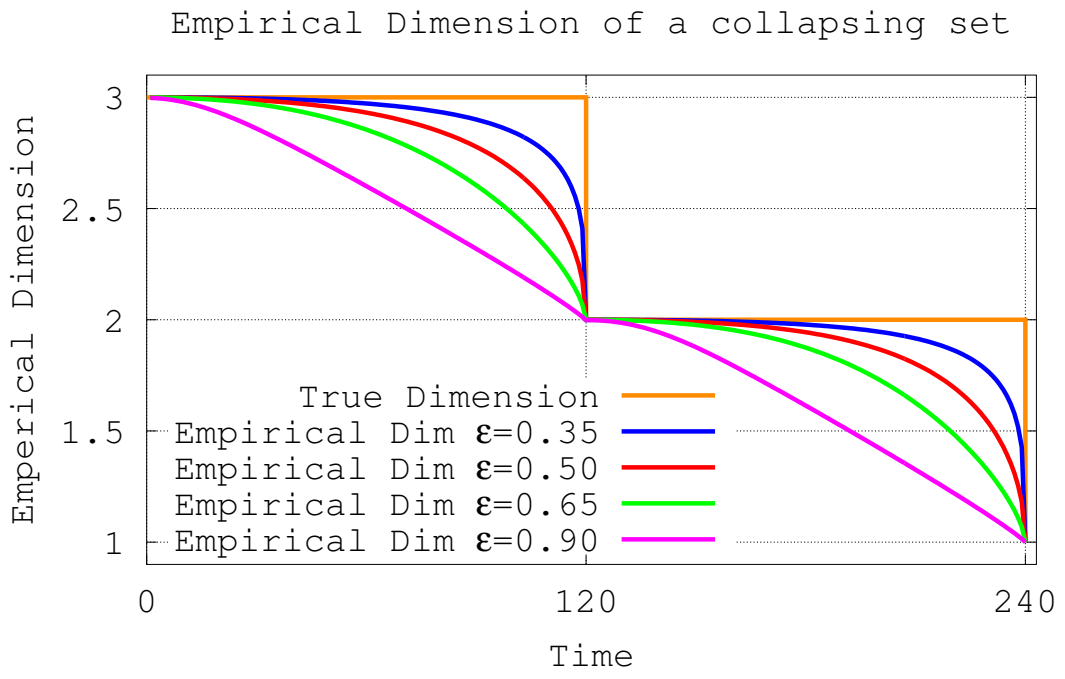


Figure 2.2: The experiment mentioned above is illustrated. A normally-distributed point cloud is created in $\mathbb{R}^3$ and is slowly collapsed into a plane and then a line. One can see that if ϵ is close to 0, empirical dimension more closely tracks true dimension, resulting in a strict dimension estimator. If ϵ is close to 1, empirical dimension changes more smoothly, resulting in a lenient estimator.

(where each subset contains only points from a single underlying subspace) by Π_{Nat}. For a fixed $\epsilon \in (0, 1]$, $\{\hat{d}_{\epsilon,i}\}_{i=1}^{k}$ are the empirical dimensions of the sets $\{\Pi_i\}_{i=1}^{k}$. We seek to minimize a function based on these dimensions to recover Π_{Nat}. To this end we define *global dimension* (GD). When thinking of this function, we take the set of data vectors to be fixed and given, and we view GD as a function of partitions, Π, of the set of data vectors. For a fixed $p \in (0, \infty)$ (we discuss the meaning of p later) we define GD as follows:

$$\mathrm{GD}(\Pi) = \|(\hat{d}_{\epsilon,1}, \hat{d}_{\epsilon,2}, ..., \hat{d}_{\epsilon,K})^T\|_p = \left(\sum_{i=1}^{K} \hat{d}_{\epsilon,i}^{p}\right)^{1/p}. \tag{2.4}$$

Our strategy for recovering Π_{Nat} will be to try and find the partition of the data

set that minimizes GD(Π). Intuitively, by trying to minimize the p-norm of the vector of set dimensions, we are looking for a partition where all of the set dimensions are small. Imagine trying to minimize this objective function by hand, and starting with a partition close to, but not equal to Π_{Nat}. If there is a point assigned to the wrong cluster, then removing it from the set it is currently assigned to should result in a significant drop in the dimension of that particular set. Re-assigning that point to the correct set, on the other hand, will have little impact on the dimension of the target set because the point will lie approximately in the span of other points already in the set.

Thus, such a change would cause a significant drop in one of the set dimensions, without disturbing the other sets, and the global dimension will decrease. This would suggest that amongst partitions that are close to it, Π_{Nat} yields the lowest global dimension. Additionally, if one considers a "random", or usual partition, then each set in that partition will tend to contain vectors from many different subspaces. Each set will have a large dimension, and the global dimension will exceed that of Π_{Nat}. This would suggest that minimizing global dimension may be a reasonable objective if we want to recover Π_{Nat}.

Unfortunately, there can exist certain special partitions of a data set that result in low global dimension (in some cases even lower than that of Π_{Nat}). For example, let us choose $p = 1$, so that the global dimension of a partition is simply the sum of the dimensions of its constituent parts. Now consider 3 lines in the plane, and a data set consisting of many points sampled from each line. In this case Π_{Nat} will consist of three sets. Each set will contain only points from a single line. The dimension of each set in Π_{Nat} is 1. Hence, GD(Π_{Nat}) = 3. On the other hand, if we consider the "degenerate" partition, that simply puts all points in a single set, then since we are in $\mathbb{R}^2$, the dimension of that set, and hence the global dimension of the partition, is 2.

The above example is actually rather special. Consider the same data set, but set p to a large value instead of 1. When p is large, the global dimension approximately returns the largest value from $\{\hat{d}_i\}_{i=1}^K$. Now consider minimizing this quantity, subject to the constraint that the partition contains no more than 3 sets. Minimizing global dimension in this setting penalizes partitions consisting of fewer, higher-dimensional sets instead of multiple, more balanced sets. Specifically, the global dimension of the degenerate partition is again approximately 2, while the global dimension of Π_{Nat} is

approximately 1, since that is the maximum dimension of its constituent sets. In fact, as shown in the next theorem, using large p effectively resolves the issue of special partitions yielding lower global dimension than Π_{Nat}.

We will consider the setting where we have K distinct[6] linear subspaces of $\mathbb{R}^D$, each of dimension $d < D$. Call these subspaces $\{L_k\}_{k=1}^K$. Assume we have a collection of non-degenerate measures $\{\mu_k\}_{k=1}^K$ supported by these subspaces (so that μ_k is supported by L_k, $k = 1, 2, ..., K$). Let $\{\boldsymbol{v}_n\}_{n=1}^N$ be a set consisting of N_k i.i.d. points from each μ_k (so $N = N_1 + N_2 + ... + N_K$). We require that $N_k > d$ for each k so that each subspace is adequately represented in the data set. Let GD_{True} be global dimension for a fixed parameter p, defined using true dimension as the "dimension estimator" for a set. That is, $GD_{True}(\Pi) = \|(d_{True}(\Pi_1), ..., d_{True}(\Pi_K))\|_p$ where the sets Π_k are the constituent sets of the partition Π and $d_{True}(\bullet)$ returns the true dimension of its parameter set. Then, we get the following result:

Theorem 2 *Let $\{L_k\}_{k=1}^K$, $\{\mu_k\}_{k=1}^K$, $\{\boldsymbol{v}_n\}_{n=1}^N$ satisfy the conditions above. If $p > ln(K)/(ln(d+1) - ln(d))$, then amongst all partitions of $\{\boldsymbol{v}_n\}_{n=1}^N$ into K or fewer sets, the natural partition is almost surely (w.r.t $\{\mu_k\}_{k=1}^K$) the unique minimizer of GD_{True}.*

The weakness of the above theorem is that it requires all of the intrinsic subspaces to have the same dimension. In practice, the global dimension objective function appears to be rather robust to subspaces with mixed dimensions. If there is a large difference in dimension between two subspaces in a dataset, then the minimum of global dimension tends to be be very near Π_{Nat}, the only difference being that a few points from the higher-dimensional set are re-assigned to lower-dimensional sets to balance out the set dimensions.

Theorem 2 gives us a quantitative way of selecting an appropriate value of p for our application. Specifically, if we identify the largest number of clusters we will need to address (K) and an upper bound for d, then this theorem gives us a minimum value of p to ensure that the natural partition minimizes global dimension. Table 2.1 shows these minimum values for different values of d and K. We do not want to choose p extravagantly large because of the potential for numerical issues when taking large

[6] "Distinct" means that no 2 subspaces are identical. We do not require independence of subspaces (that assumption would be violated in this application due to low ambient dimension).

powers. In our setting, we want to be able to handle up to 4 sets and we will use one less than the ambient dimension as an upper bound for the intrinsic dimension ($d = 8$). According to Theorem 2 we should select $p \geq 11.77$. In all of our experiments we set $p = 15$ to give us a safety margin. We did some tests on one of our motion segmentation databases (outlier-free RAS) to see how sensitive the minimizer of global dimension is to p in practice. We found that values as low as $p = 10$ result in nearly identical performance to $p = 15$, and we don't start to see significant degradation in results in the other direction until $p = 25$.

Table 2.1: Minimum values of p to ensure that Π_{Nat} is the unique minimizer of G_p, for various values of K and d.

		d				
		8	7	6	5	4
K	2	5.89	5.19	4.50	3.80	3.11
	3	9.33	8.23	7.13	6.03	4.92
	4	11.77	10.38	8.99	7.60	6.21

2.4 A Fast Algorithm for Minimizing Global Dimension

Global dimension is defined on the set of partitions of a data set. With a discrete domain, finding ways of quickly minimizing the objective function is non-trivial. In this section we briefly introduce a method, which we will call *Global Dimension Minimization* (GDM) for doing exactly this.

GDM is based on the gradient projection method [17, §2.3]. In order to apply a gradient-based method, we need to re-formulate the problem so that we have a smooth objective function over a convex domain. To do this we employ the notion of *fuzzy assignment*. Rather than trying to assign each data point a label, identifying it with a single cluster, we allow each point to be associated with every cluster simultaneously, in varying amounts. Specifically, we assign each data point $\boldsymbol{v}_j$ a probability vector where the i'th coordinate holds the strength of $\boldsymbol{v}_j$'s affiliation with cluster i. Assuming we have a data set of N points in $\mathbb{R}^D$, and we seek K clusters, we need N probability vectors of length K to encode the *soft partition* of the data. This membership information will be stored in a *membership matrix*, $\boldsymbol{M}$, where each column is a probability vector. Element

(i, j) of the matrix $\boldsymbol{M}$ holds the strength of $\boldsymbol{v}_j$'s affiliation with cluster i.

The next step is to extend the definition of global dimension so that it is defined on soft partitions in a meaningful way. In its original formulation, to evaluate the global dimension of a partition, we would break up the data set into parts, based on the partition, and estimate the dimension of each part using empirical dimension. To extend this to soft partitions, we estimate the dimension of the k'th set in a partition by scaling each data point by its respective affiliation strength to set k ($\boldsymbol{v}_n$ is multiplied by $\boldsymbol{M}_{(k,n)}$). We then use empirical dimension to estimate the dimension of the scaled set. In essence, each point is now included in each dimension estimate. However, if a point is scaled so that it lays near the origin when considering a given set, it has little impact on the estimated dimension of that set. In fact, if we look at the global dimension of a soft partition that assigns each data point entirely to a single set ($\boldsymbol{M}$ has only 1's and 0's in it), then the global dimension of that soft partition, using our new definition, agrees with the global dimension of the corresponding "hard partition", using our original definition. Thus, this change is a reasonable extension of the original definition to soft partitions. Our extended definition of global dimension is:

$$\begin{aligned} &GD = \|(\hat{d}_\epsilon^1, \hat{d}_\epsilon^2, ..., \hat{d}_\epsilon^K)\|_p \\ &\text{where } \ \hat{d}_\epsilon^k = \hat{d}_\epsilon(\boldsymbol{M}_{(k,1)}\boldsymbol{v}_1, \boldsymbol{M}_{(k,2)}\boldsymbol{v}_2, ..., \boldsymbol{M}_{(k,N)}\boldsymbol{v}_N). \end{aligned} \tag{2.5}$$

With this modified formulation, global dimension is an almost-everywhere differentiable function defined over the Cartesian product of N K-dimensional probability simplexes. One can check that this is a convex domain (the product of convex sets is convex). A natural approach to minimizing a problem of this sort is the gradient projection method [17, §2.3]. In this method, we begin at some initial state, compute the gradient of the objective function, take a step in the direction opposite the gradient, and then project our new state back into the domain of optimization. This is repeated until our state converges.

The gradient of global dimension can be computed, but we need some notation first. For $i = 1, \ldots, K$, we denote by $\boldsymbol{A}_k$ the D-by-N matrix whose j'th column equals $\boldsymbol{M}_{(k,j)}\boldsymbol{v}_j$ for $j = 1, 2, ..., N$ (i.e., $\boldsymbol{A}_k$ is the data matrix scaled according to weights for cluster k). Let $\boldsymbol{A}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k(\boldsymbol{V}_k)^T$ be the thin SVD of $\boldsymbol{A}_k$, and $\boldsymbol{\sigma}_k$ denote the vector of

elements from the diagonal of $\boldsymbol{\Sigma}_k$. Let $\delta = \epsilon/(1-\epsilon)$. Define

$$\boldsymbol{D}_k = \left(\frac{\|\boldsymbol{\sigma}_k\|_\epsilon^{1-\epsilon} \|\boldsymbol{\sigma}_k\|_\delta}{\|\boldsymbol{\sigma}_k\|_\delta^2} \right) \cdot (\boldsymbol{\Sigma}_k)^{\epsilon-1} - \left(\frac{\|\boldsymbol{\sigma}_k\|_\epsilon \|\boldsymbol{\sigma}_k\|_\delta^{1-\delta}}{\|\boldsymbol{\sigma}_k\|_\delta^2} \right) \cdot (\boldsymbol{\Sigma}_k)^{\delta-1}. \tag{2.6}$$

Theorem 3 *The derivative of global dimension w.r.t. an arbitrary element of the membership matrix $\boldsymbol{M}$ is given by:*

$$\frac{\partial GD}{\partial \boldsymbol{M}_{(k,n)}} = \boldsymbol{V}_{k\,(n,:)} \left((\hat{d}_\epsilon^k)^{p-1} \, \|(\hat{d}_\epsilon^1, \hat{d}_\epsilon^2, ..., \hat{d}_\epsilon^K)\|_p^{1-p} \, \boldsymbol{D}_k \, (\boldsymbol{U}_k)^T \right) \boldsymbol{A}_{(:,n)}. \tag{2.7}$$

A proof of Theorem 3 is included in the appendix. This theorem allows us to evaluate the gradient vector of global dimension. As was mentioned before, in an iteration of the gradient projection method we take a step in the direction opposite the gradient. Computing a good step size is frequently a challenging task, but here we are fortunate. Our domain has a meaningful natural scale, since it is formed as a product of probability simplexes. Intuitively, our step size should be large enough to move us across the entire space in a reasonable number of steps, but small enough that any individual membership vector can move only a fraction of the way across its own simplex in one step. In practice, we scale each step so that the membership vectors most affected by the step move a distance of .3 on average. This seems to work well in general.

Finally, one can check that projecting onto the domain of optimization can be accomplished by individually projecting each column of $\boldsymbol{M}$ onto the standard K-dimensional probability simplex.

We have outlined a projected gradient descent method for minimizing global dimension. The above method forms the core of the GDM algorithm. However, since the global dimension function is non-convex (and hence may contain multiple local minimums), it is important to achieve reasonably good initialization. Our initialization strategy is inspired by ALC [18]. We start with a "trivial" partition where each point is in its own set, and we randomly select many pairs of sets in the partition. For each pair, we hypothetically merge the two sets and measure the resulting global dimension. We select the pair that results in the lowest global dimension when merged and we effect that merge. We then repeat the process iteratively until we have the desired number of sets in our partition (in each step the number of sets in the partition decreases by 1).

After initialization, the projected gradient descent algorithm is run until convergence (or for a fixed, but large number of iterations). Thresholding is performed to recover a "hard partition" from our soft partition (point j is assigned to cluster i if $\boldsymbol{M}_{(i,j)}$ is the largest element from column j of $\boldsymbol{M}$). After this is done, we perform a final genetic stage to clean up small errors which may have occurred in any of the previous stages. This is done by taking each point and hypothetically re-assigning it to each different cluster (while all other point assignments are kept fixed) and retaining the assignment that results in the lowest global dimension. This is repeated a few times or until no single-point re-assignments reduce global dimension. This primarily helps with placing points that lie near the intersections of different subspaces (their fuzzy assignments may associate them almost equally to 2 different subspaces, making them difficult to place). Finally, we run this entire process several times and return the best partition of all runs (as measured by global dimension).

Algorithm 1 GDM Algorithm for HLM

Input: $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\} \subseteq \mathbb{R}^D$: data, K: number of clusters, p: global dimension parameter, ϵ: empirical dimension parameter, n_1, n_2, n_3: number of iterations (default: $n_1 = n_3 = 10$, $n_2 = 30$)

Output: A partition, Π, of X into K disjoint clusters

for $i = 1 : n_1$ **do**

- $\Pi :=$ Partition of X where each point is in its own set.

while number of sets in Π greater than K **do**

- Randomly choose several pairs of sets.
- For each pair, measure the effect on global dimension if the pair is merged.
- Merge the pair of sets which results in the lowest global dimension.

- Convert Π to a soft partition, encoded in membership matrix M.

for $j = 1 : n_2$ **do**

- Compute gradient of global dimension, ∇GD.
- $\rho =$ average magnitude of largest 10% of columns of ∇GD.
- Take a step in direction $-1 * \nabla GD$ of length $.3/\rho$.
- Project each column of M onto the standard k-dim. probability simplex.

- Convert M back to a "hard partition", Π, by thresholding.

for $j = 1 : n_3$ **do**

for $n = 1 : N$ **do**

- Check if re-assigning point n to another cluster drops global dimension.
- If so, re-assign point n to that cluster.

- Return partition from all runs with lowest global dimension.

2.4.1 Complexity of GDM

A thorough analysis of the computational complexity is not included here; this is a short summary of the computational aspects involved. The main numerical component of GDM is computing ∇GD. For a single iteration its complexity is $O(K \cdot N \cdot D^2)$. Our choice of ρ requires a sorting procedure and is thus of order $O(N \cdot \log(N))$ operations for a single iteration. The initialization of the algorithm via ALC-type procedure [18] requires $O(n_1 \cdot N \cdot \log(N) \cdot D^2)$ operations. Also, the last genetic step has the following

complexity $O(n_1 \cdot n_3 \cdot K \cdot D^2 \cdot N^2)$ (without taking advantage of incremental SVD). In theory, we can make the algorithm linear in the number of points N, by randomly initializing it, removing the genetic "clean-up" step and changing how we select our step size[7] . We have good numerical evidence, even with large N, that this can result in good accuracy and speed for artificial data. Regardless, for the values of N in our application the algorithm is sufficiently fast and these additional steps help improving accuracy, especially for points which are nearby several clusters (whose percentage is not negligible when N is small).

2.5 Detecting and Rejecting Outliers with GDM

In practice, it turns out that the GDM algorithm described above is naturally robust to a small number of outliers (in that they do not tend to affect the classification of inliers), but no instruments were put in place for explicitly detecting or rejecting these outlying points. In this section, we introduce a modification to GDM that allows for explicit outlier detection and rejection. The guiding intuition is that an outlier has the property that if the true hybrid-linear structure is reflected in a partition, then no matter which group we assign the outlier to, it causes a significant increase in the empirical dimension of that group. This, in turn, results in a significant increase in global dimension. In other words, if we have a partition that reflects the true hybrid-linear structure of the data set, then there is no good place to put an outlier. If the algorithm was given the option of paying a fixed, low price for the right to ignore a given point, it would make sense for it to exercise this option on outliers, and only segment inliers.

We propose modifying the global dimension objective function, and the accompanying variational development in the following way:

$$\mathrm{GD}(\boldsymbol{M}) = \alpha \|\boldsymbol{M}_{1,:}\|_1 + \|(\hat{d}_{\epsilon,2}, \hat{d}_{\epsilon,3}, ..., \hat{d}_{\epsilon,K+1})^T\|_p \tag{2.8}$$

where:

$$\hat{d}_{\epsilon,k} = \hat{d}_\epsilon \left(\boldsymbol{M}_{k,1}\boldsymbol{v}_1, \boldsymbol{M}_{k,2}\boldsymbol{v}_2, ..., \boldsymbol{M}_{k,N}\boldsymbol{v}_N\right). \tag{2.9}$$

[7] This is assuming that we will not require more iterations to get close enough to the minimum that we can apply thresholding. In our experiments the number of needed iterations does not appear to grow with N, but we do not have any results to guarantee this.

This modification adds an additional "cluster" to the problem (call it cluster 1), and we treat it differently than the others. Clusters 2 through $K+1$ contribute to the global dimension in the same way that they did in the original development. Cluster 1 contributes to the cost function the sum of the membership strengths of all data points to this cluster. This is the "fuzzy assignment" version of the following notion: we allow the algorithm to pay a fixed price, α, for the right to ignore any particular data point (not assign it to any true cluster).

2.5.1 Modification to GDM

The proposed modification to the objective function only trivially changes the state space (now it is the product of N $K+1$-dimensional probability simplexes, as opposed to K-dimensional simplexes). Thus, our method of projecting states onto the convex domain is effectively the same. The change to the objective function does mean that we must re-evaluate the gradient of global dimension. The computation is very similar to the unmodified version, and the result is:

$$\frac{\partial GD}{\partial m_1^n} = \alpha m_1^n \tag{2.10}$$

and, for all $k > 1$

$$\frac{\partial GD}{\partial m_k^n} = \boldsymbol{V}_{k(n,:)} \hat{d}_{\epsilon,k}^{p-1} \left(\hat{d}_{\epsilon,2}^{p} + ... + \hat{d}_{\epsilon,K+1}^{p} \right)^{\frac{1}{p}-1} \boldsymbol{D}_k \boldsymbol{U}_k^T \boldsymbol{v}_n, \tag{2.11}$$

where the notation and constants are as defined in §2.4. Thus, the necessary modifications to GDM are:

1. Update the evaluation of the objective function GD according to (2.8).
2. Update the initialization of the state vector to include an outlier group.
3. Update the state projection routine to accommodate additional dimensions in domain.
4. Update the evaluation of ∇GD according to 2.10 and 2.11.

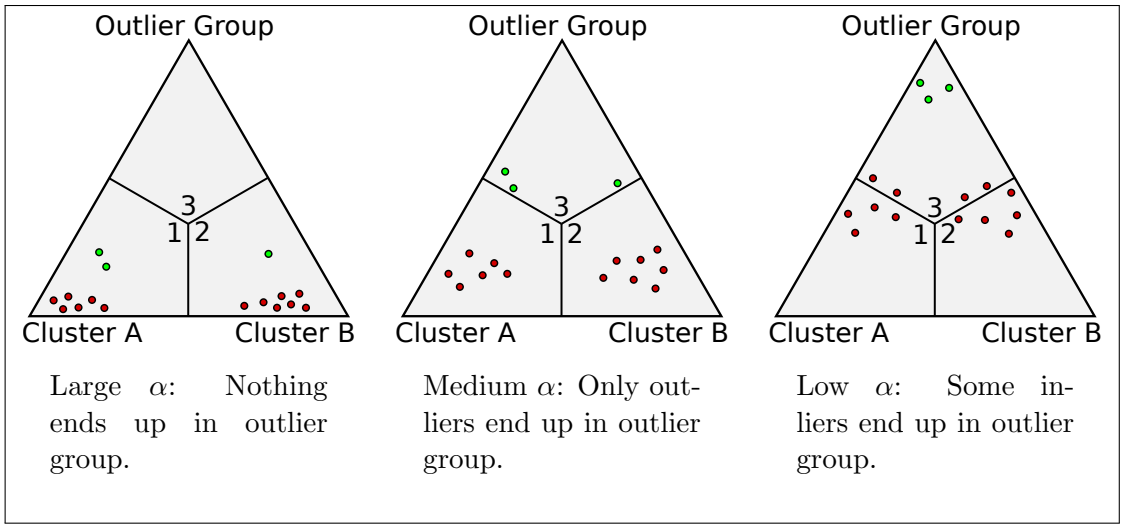


Figure 2.3: The three images here illustrate the problem with GDM-Naive. Each triangle represents the probability simplex containing the fuzzy assignment vectors for a fictitious data set. The fuzzy assignment for each point is plotted after many iterations of GDM. Points in red are inliers and points in green are outliers. The quantization regions (for the threshold step) are numbered 1-3. One can see that if α is not chosen correctly, points can be quantized into the wrong cluster. On the other hand, the outlier ranking of a point (the # of points closer the the outlier corner) is a more stable quantity.

2.5.2 Practical Implementations of Outlier Rejection

We have described an idea for how to handle outliers, but it introduces a new parameter, α. It is not immediately clear how one should choose this parameter, and how sensitive the results will be to it. In theory one would need to choose an outlier cost, α, that is not so high that nothing is ever assigned to the outlier group, but not so low that large quantities of inliers are assigned to this group. The appropriate values would likely depend on multiple quantities, like intrinsic dimension, noise level, and distortion of the underlying subspaces. These are quantities that can vary not just between applications, but also from data set to data set for a single application. Applying the suggested modification exactly as proposed (and trying to "tune" this parameter) would therefore lead to an unreliable and unpredictable algorithm. We refer to this approach as GDM-Naive, and Figure 2.3 illustrates why this method is unsound. Instead, we propose two variations of this method, which lead to more reliable solutions.

1. **GDM Known-Fraction:** Run the proposed algorithm with a fixed, low value of α (we use $\alpha = 0.01$) but stop before the threshold step. Rank the data points according to their membership strengths to the outlier group. Remove a pre-set fraction of the data set (the part that most strongly affiliates with the outlier group). Continue with the classic (non-outlier version) of the variational algorithm on the surviving points only[8] - this provides the inlier segmentation. The points that were removed are labelled outliers.

2. **GDM Model-Reassign:** Run method 1 above (GDM Known-Fraction). Fit subspaces of appropriate dimension (round the empirical dimension) to each set in the resulting partition. Re-assign all points (including those that were decided to be outliers) according to their distances from each subspace. Call a point an outlier if it is more than some fixed distance, κ, from all of the subspaces.

Each of the proposed methods handles the task of selecting α, but introduces a new parameter. For method 1, this is the percentage of the data set to throw out. For method 2, the new parameter is the maximum distance a point can be from a subspace to be considered an inlier. Both of these parameters are more natural than selecting α. In a noisy environment, one may have an idea, based on experiments, of what percentage of the data set will be outliers, or what the inlier modelling error tends to be. Additionally, when using the "Model-Reassign" method, one could find the average and variance of the residuals, μ, and σ^2 respectively, when fitting subspaces to the inlier clusters. These quantities can be used to come up with a reasonable value of κ for a given application ($\mu + r\sigma$ for some r). One could also find these values on a per-cluster basis and have a different outlier threshold for each cluster.

[8] We could skip this step and segment directly from the fuzzy assignment that we already have. Refining the membership matrix after removing the outliers is done to repair whatever damage the outliers may have done to the membership matrix before thresholding.

2.6 Results on Real-World Data

2.6.1 Performance in the Absence of Outliers

We tested the GDM algorithm on 2 motion segmentation databases. First, we used the outlier-free RAS database [7, 11] and compared with many leading methods in 2-view segmentation. We noticed that some of the HLM methods performed better when using the linearly embedded point correspondences than with the nonlinear embedding. Therefore, in Table 2.2 we present each of the competing HLM algorithms twice. Where "Linear" appears, the algorithm was run on the feature trajectories in $\mathbb{R}^4$. Where "Nonlinear" appears, the algorithm was run on the Kronecker products (in $\mathbb{R}^9$) of the standard homogeneous coordinates of each feature correspondence. Figure 2.5 presents more details on the performance of the HLM methods with the nonlinear embedding, and Table 2.3 gives the average runtimes of these methods. The other HLM methods we included are SCC [12], MAPA [14], SSC [19], SLBF [13], and LRR [20]. We also included two other successful methods for two-views (for which there was a code available online): RAS [7] and HOSC [10]. Algorithm parameters and our experiment procedure are detailed in §2.8.4.

(a) SCC **(b)** GDM **(c)** MAPA

Figure 2.4: Clustering by SCC, GDM, and MAPA on file 6 of the outlier-free RAS database.

Table 2.2: Misclassification Rates (given as % Error) on the outlier-free RAS database.

		File Number													Average	Average w/o File #8
		1	2	3	4	5	6	7	8	9	10	11	12	13		
Method/Embedding	GDM Nonlin	**0.85**	**0.00**	1.57	0.65	**0.00**	**0.00**	**0.00**	12.76	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.22	**0.26**
	SCC Linear	**0.85**	**0.00**	1.18	0.65	**0.00**	1.37	**0.00**	1.42	0.39	**0.00**	**0.00**	1.01	**0.00**	**0.53**	0.45
	SCC Nonlin	**0.85**	**0.00**	24.41	**0.00**	**0.00**	19.18	**0.00**	**0.00**	**0.00**	13.97	5.36	0.84	1.10	5.05	5.48
	MAPA Linear	**0.85**	3.65	1.18	0.65	**0.00**	13.70	15.97	1.29	**0.00**	**0.00**	**0.00**	0.67	3.30	3.17	3.33
	MAPA Nonlin	**0.85**	20.55	21.65	0.65	**0.00**	21.92	6.25	7.73	**0.00**	13.97	1.43	0.34	3.30	7.59	7.57
	SSC Linear	1.69	18.26	**0.79**	1.94	**0.00**	**0.00**	6.25	32.22	**0.00**	**0.00**	14.64	1.35	4.40	6.27	4.11
	SSC Nonlin	1.27	**0.00**	22.44	0.65	**0.00**	21.92	**0.00**	9.02	**0.00**	13.97	9.29	12.12	6.59	7.48	7.35
	SLBF Linear	**0.85**	0.46	1.18	0.65	**0.00**	**0.00**	**0.00**	0.26	**0.00**	**0.00**	**0.00**	0.67	4.40	0.65	0.68
	SLBF Nonlin	**0.85**	**0.00**	5.12	1.94	**0.00**	19.18	**0.00**	10.57	**0.00**	13.97	**0.00**	1.68	14.29	5.20	4.75
	LRR Linear	5.08	24.66	1.18	2.58	2.38	2.74	**0.00**	29.12	**0.00**	**0.00**	8.93	14.81	18.68	8.47	6.75
	LRR Nonlin	1.27	9.13	2.76	1.94	**0.00**	**0.00**	3.47	3.61	**0.00**	**0.00**	9.64	18.18	2.20	4.02	4.05
	RAS	11.65	**0.00**	2.56	9.68	16.19	26.03	26.74	11.21	3.28	13.97	3.21	2.36	6.59	10.27	10.19
	HOSC d=2	**0.85**	**0.00**	24.41	1.61	**0.00**	**0.00**	**0.00**	22.94	**0.00**	**0.00**	**0.00**	**0.00**	2.20	4.00	2.42
	HOSC d=3	1.27	23.74	24.41	3.23	**0.00**	19.18	12.15	19.59	23.75	**0.00**	1.43	1.01	17.58	11.33	10.65

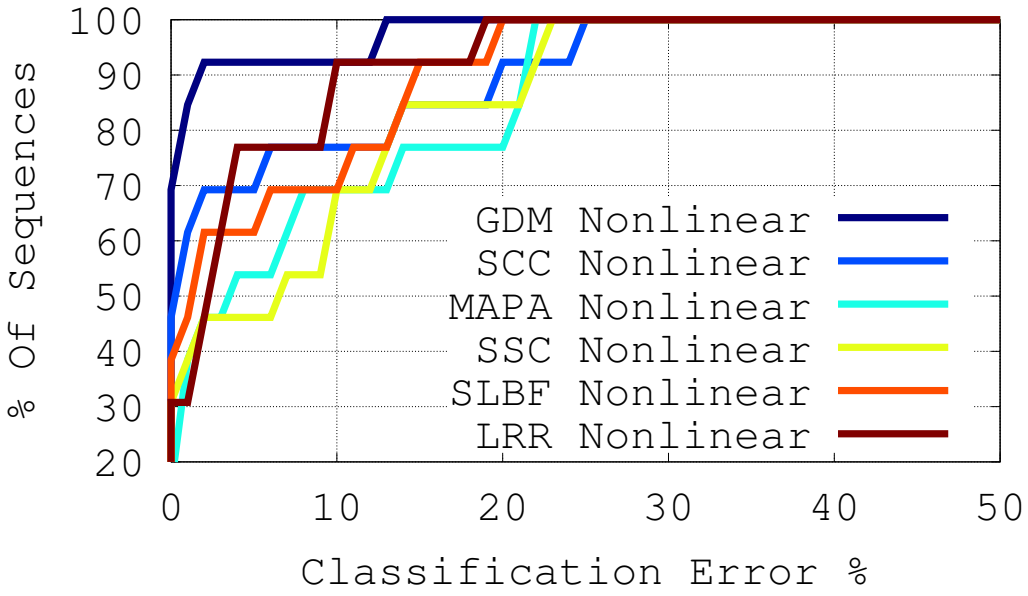


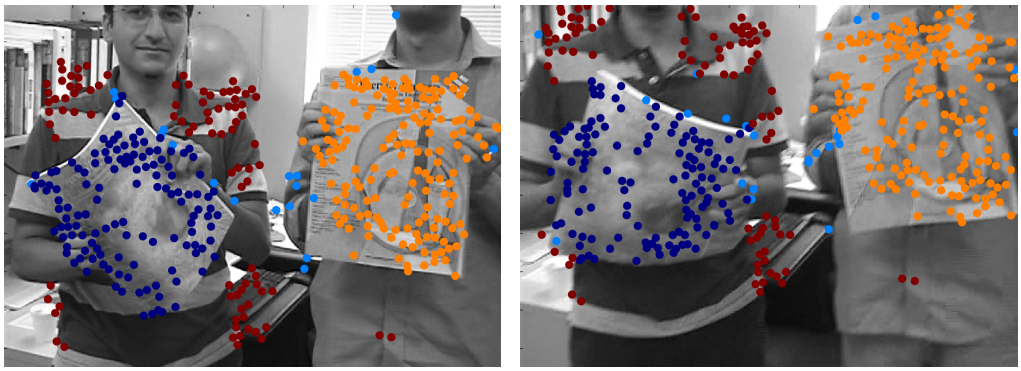
Figure 2.5: GDM is compared against other HLM methods on the nonlinear 2-view embedding of the outlier-free RAS database.

Table 2.3: Average runtimes (per file) of HLM-based methods on non-linearly embedded (outlier-free) RAS data.

		Runtime (seconds)
Method	GDM	12.7
	SCC	2.3
	MAPA	5.6
	SSC	89.5
	SLBF	4.0
	LRR	0.8

From Table 2.2 and Figure 2.5, we can see that GDM performs very competitively on this database. There is only a single file (#8) on which GDM exhibits significant error. This file contains features from two bent magazines as well as a rigid background. Since the bent magazines are clearly non-rigid, our model assumptions are not met (see Fig. 2.6). There were two methods in the comparison that had a lower average misclassification error than GDM ("SCC Linear" and "SLBF Linear"). This is because they

perform significantly better on file (#8). Both of these are spectral methods, accompanied by the linear embedding, and are therefore better able to handle the manifold structure that results from the non-rigidity of the objects in this file. Amongst the other files however, GDM performs better on average than both of these two methods (see the last column of Table 2.2). Comparing just the HLM-based methods on the nonlinearly-embedded data, GDM performs better than any other method, with the most perfect classifications and the fewest number of files with significant errors. Figure 2.5 more clearly emphasizes this superb performance amongst methods using the nonlinear embedding.



(a) Frame 1 **(b)** Frame 2

Figure 2.6: File 8 in the RAS database. This is a problematic file because the two magazines in the scene appear to undergo a non-rigid transformation between the two frames. Point correspondences are colored according to ground-truth segmentation.

We also performed experiments on the Hopkins155 database [21]. For 2-view segmentation we extracted the first and last frame of each sequence and performed 2-view segmentation on the nonlinear embedding (in $\mathbb{R}^9$) of the data. For comparison, we demonstrate the results of some other HLM algorithms on this embedded data: MAPA [14], SCC-MS [12, 13] and SLBF-MS [13]. We also supply results for a few state-of-the-art HLM methods on the full n-view feature trajectories. For these n-view results we chose in this table the best methods on Hopkins155 we are aware of, which do not require careful tuning with parameters: SSC [19] and SLBF-MS [13]. We also include the reference (REF) results [21]. REF finds the best linear models (via least

squares approximation) for each cluster of embedded points (given the ground truth segmentation), and then finds new clusters by assigning points to the models they best agree with. For GDM on this database, it was necessary to increase the number of random initializations (n_1 in Algorithm 1) to achieve reliable convergence (we changed it from 10 to 30). From Table 2.4 we see that GDM outperforms the other 2-view methods (although SCC matches or nearly matches its performance in some categories). We remark that we also tested a genetic algorithm for minimizing the global dimension and it achieved even more accurate results, however, we do not include it here since it is not as fast as GDM.

It is also interesting to note that our results for 2-views are comparable to the reference results with n-views. That is, the results of GDM are the best one can expect with pure linear modeling given many views and assuming an affine camera model. GDM for n-views gave comparable results and we thus did not include it. On the other hand, both SLBF-MS and SSC-N are able to obtain better results with n-views and this may be because their machinery of spectral clustering (together with good choices of spectral weights) allows them to take into account some of the manifold structure and nearness of points (information beyond linear modeling).

Table 2.4: The mean and median percentage of misclassified points for two-motions and three-motions in Hopkins 155 database with comparisons to state-of-the-art n-views. Winning results amongst the 2-view methods are bold-faced in each category.

	2-motion	Checker		Traffic		Articulated		All	
		Mean	Median	Mean	Median	Mean	Median	Mean	Median
2-view	GDM	**2.79**	**0.00**	**1.78**	**0.00**	**2.66**	**0.00**	**2.51**	**0.00**
	MAPA	12.85	14.07	6.49	6.93	7.15	5.33	10.69	10.03
	SCC (d=7)	**2.79**	**0.00**	1.97	**0.00**	3.42	**0.00**	2.64	**0.00**
	SLBF (d=6)	8.18	1.39	3.98	0.53	4.73	0.40	6.78	1.11
n-view	SLBF-MS ($2F$,3)	1.28	0.00	0.21	0.00	0.94	0.00	0.98	0.00
	SSC-N ($4K$,3)	1.29	0.00	0.29	0.00	0.97	0.00	1.00	0.00
	REF	2.76	0.49	0.30	0.00	1.71	0.00	2.03	0.00

	3-motion	Checker		Traffic		Articulated		All	
		Mean	Median	Mean	Median	Mean	Median	Mean	Median
2-view	GDM	**5.37**	**3.23**	**4.23**	**2.69**	5.32	5.32	**5.14**	**3.13**
	MAPA	21.89	19.49	13.15	13.04	9.04	9.04	19.41	18.09
	SCC (d=7)	8.05	5.85	4.67	5.45	5.85	5.85	7.25	5.45
	SLBF (d=6)	14.08	12.80	7.93	6.75	**4.79**	**4.79**	12.32	9.57
n-view	SLBF-MS ($2F$,3)	3.33	0.39	0.24	0.00	2.13	2.13	2.64	0.22
	SSC-N ($4K$,3)	3.22	0.29	0.53	0.00	2.13	2.13	2.62	0.22
	REF	6.28	5.06	1.30	0.00	2.66	2.66	5.08	2.40

2.6.2 Performance in the Presence of Outliers

We tested the methods suggested in §2.5.2 on the outlier-corrupted RAS database [7]. The performance of classic GDM (no outlier rejection machinery) is also presented on this database, as is the performance of GDM on the corresponding outlier-free database (for comparison purposes). We also show results from three competing methods for segmenting motion with outliers: RAS [7], HOSC [10], and LRR [20, 22] with outlier rejection performed by identifying the largest columns of $\mathbf{E}$, as suggested in [22, pg. 9]. The details of this experiment, including parameter values, are given in §2.8.4.

It is non-trivial to fairly compare different algorithms in the presence of outliers. Each method generally has at least one parameter for controlling how it handles outliers. This parameter balances the desire for a high outlier detection rate with a desire for a low false alarm rate (these two quantities are invariably correlated). Using any popular metric for evaluating segmentation accuracy (like misclassification rate for true inliers[9]), the performance of each algorithm will depend substantially on its outlier handling parameter. In general terms, if an algorithm is allowed to discard points as outliers more freely, then the accuracy on the surviving points will improve. Thus, if one method is more conservative than another in discarding points as outliers, the results will likely be skewed in favor of one method over the other. It is therefore important when looking at segmentation accuracy to think in terms of accuracy for a given *true positive rate* (TPR) and *false positive rate* (FPR):

$$\text{TPR} = \frac{\#\text{ of outliers that were identified as outliers}}{\#\text{ of outliers in dataset}} * 100,$$

$$\text{FPR} = \frac{\#\text{ of inliers that were identified as outliers}}{\#\text{ of inliers in dataset}} * 100.$$

There are two aspects of these algorithms we wish to compare. The first is outlier detection performance (how good is each method at distinguishing between inliers and outliers). The second is segmentation performance, where we evaluate how good each

[9] "True inliers" are points that are inliers according to ground truth.

method is at segmenting motions in the presence of outliers.

To compare the outlier detection performance of multiple methods, a common tool is the ROC curve, which parametrically plots the TPR vs. FPR as a function of the outlier parameter for a method. A "random classifier" that randomly labels points as inliers or outliers will have an ROC curve lying along the line TPR = FPR. An ideal classifier will follow the line TPR = 1. Hence, methods can be compared by seeing which ROC curve is highest over the broadest range of FPRs (or over the FPRs one is interested in). The ROC curves for GDM (using the Model-Reassign outlier detection method and varying κ), LRR (by varying λ), RAS (by varying "outlierFraction"), and HOSC (by varying α), are presented in Fig. 2.7.

Table 2.5: Misclassification Rates (given as % Error) of inliers on the RAS database. All but 'Classic GDM - clean' are misclassification rates when run on the outlier-corrupted datasets. 'GDM - clean' gives the performance of the unmodified GDM algorithm, when run on the outlier-removed datasets (included as a reference).

		File Number													**Average**	**Average w/o File #8**
		1	2	3	4	5	6	7	8	9	10	11	12	13		
Method	GDM - Model-Reassign	2.97	**0.00**	4.33	**1.29**	0.95	**0.00**	**0.00**	12.63	**0.00**	6.62	**0.00**	**2.02**	17.58	**3.72**	**2.98**
	GDM - Classic	**0.85**	**0.00**	**1.57**	32.26	**0.00**	**0.00**	**0.00**	22.94	**0.00**	**0.00**	22.14	8.75	16.48	8.08	6.84
	RAS	19.49	5.02	1.97	5.81	15.71	23.29	25.00	11.86	2.32	13.97	12.14	18.18	21.98	13.60	13.74
	LRR	4.24	20.55	22.83	7.10	7.14	8.22	18.75	34.54	2.32	27.21	8.57	11.78	25.27	15.27	13.67
	HOSC (d=2)	11.02	22.37	16.54	33.55	10.95	2.74	11.11	**11.34**	3.09	13.97	36.79	66.67	**8.79**	19.15	19.80
	GDM - clean	0.85	0.00	1.57	0.65	0.00	0.00	0.00	12.76	0.00	0.00	0.00	0.00	0.00	1.22	0.26

GDM was again run using the nonlinear embedding of the data. HOSC was run with the linear embedding and LRR was run with the nonlinear embedding since these were the cases that yielded the best performance in the outlier-free tests for each algorithm (see §2.8.4 for more details). From Fig. 2.7 we can see that GDM is very competitive at detecting outliers on this database. At low FPRs GDM yields comparatively excellent performance. At higher FPRs HOSC has a moderate advantage at outlier detection v.s. GDM. However, it will be seen later (Table 2.5) that HOSC is not competitive at segmentation in the presence of outliers. Furthermore, the presented HOSC results were prepared using $d = 2$ (see §2.8.4), instead of $d = 3$ as argued for by its authors. Using $d = 3$ gave worse results and made the algorithm take an extremely long time to execute.

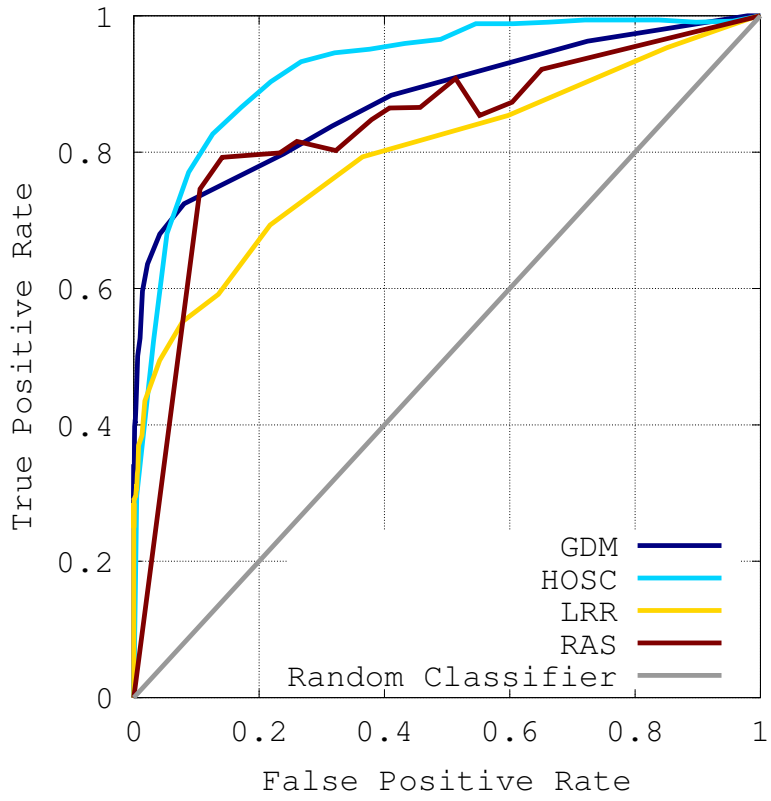


Figure 2.7: The outlier detection performance of GDM (Model-Reassign) is compared against other motion segmentation methods on the outlier-free RAS database.

The TPR and FPR for a robust segmentation algorithm cannot generally be controlled independently or arbitrarily. Thus, for a comparison of segmentation accuracy, one must select "reasonable" parameters for each method, which correspond to the same general region of ROC space. It should be understood that since the TPR and FPR cannot be controlled exactly for each method, any such comparison is inherently unfair, and by manipulating outlier parameters the results can be skewed somewhat in any direction.

For the purpose of fairly comparing GDM with other methods, we must select only one of the suggested outlier detection schemes for GDM ("GDM - Known Fraction" or "GDM Model-Reassign"). To effectively use "GDM - Known Fraction", one must either know roughly what fraction of his or her data are going to be outliers, or be in a situation where over-rejecting points as outliers is acceptable (you can then overestimate the outlier fraction). Since this is not usually the case, we will consider the results of "GDM Model-Reassign" when comparing with other methods.

In Table 2.5 we present a file-by-file comparison of segmentation accuracy for the aforementioned methods using parameters that place the FPR of each method in the range of 0.01 to 0.08. Table 2.6 reports the average TPR and FPR for each of these methods.

Table 2.6: True Positive Rate (TPR) and False Positive Rate (FPR) for each method in our segmentation comparison in Table 2.5. GDM - Model Reassign, RAS, LRR, and HOSC were each tuned to achieve a false positive rate in the range of 0.01 to 0.08.

		TPR	FPR
Method	GDM - Model-Reassign	0.56	0.01
	GDM - Classic	NA	NA
	RAS	0.74	0.08
	LRR	0.49	0.04
	HOSC	0.71	0.06
	GDM - clean	NA	NA

One can see from Table 2.5 that "GDM Model-Reassign" causes an overall improvement in segmentation accuracy (vs "GDM - Classic") in the presence of outliers. There

were several files where the outliers cause the classic GDM method to misclassify large fractions of the data sets (files 4, 8, and 11 have inlier misclassification rates over 20%). On these files the error rates of "GDM Model-Reassign" are dramatically lower. There are some files where the outlier detection framework appears to hurt performance, but in most of these cases the degradation is slight. The results for GDM are better in most cases (and on average) than the competing methods, although there are a few files where GDM is outperformed by a small margin. Unlike the strong outlier detection performance of HOSC discussed earlier, the segmentation capabilities of HOSC appear very intolerant to outliers (if even a few outliers slip through, segmentation performance suffers).

2.7 Conclusions

We presented a new approach to 2-view motion segmentation, which is also a general method for HLM. Its development was motivated by the main obstacles of recovering multiple subspaces within the nonlinear embedding of point correspondences into $\mathbb{R}^9$. The first obstacle is due to nonuniform distributions along subspaces and the second one is due to unknown dimensions of subspaces. The idea was to minimize a global quantity, i.e., global dimension, which does not make an a-priori assumption on the dimensions of the underlying subspaces. We formulated a fast method to minimize this global dimension, which we referred to as GDM. We demonstrated state-of-the-art results of GDM for 2-view motion segmentation.

We carefully explained the meaning of the two main parameters in our algorithm, p and ϵ, and the trade-offs they express. We gave a theoretical basis for selecting an appropriate value of p. Needless to say that these parameters are fixed throughout the paper. We described a preliminary theory which motivated the notion of global dimension, and we justified why it makes sense as an objective function in our application.

Finally, we presented an outlier detection/rejection framework for GDM. We explored two complimentary implementations of this framework, and we presented results demonstrating that it is competitive at handling outliers in this application.

2.8 Appendix For Part I

2.8.1 Proof of Theorem 1

We prove the four properties of the statement of the theorem. For simplicity we assume that $D < N$. That is, the number of data points is greater than the dimension of the ambient space. This is the usual case in many applications.

Proof of Property 1: Clearly, scaling all data vectors by $\alpha \neq 0$ results in scaling all the singular values of the corresponding data matrix by α. Furthermore, this results in scaling by α both the numerator and denominator of the expression for the empirical dimension for any $\epsilon > 0$. Therefore, the empirical dimension is invariant to this scaling.

Proof of Property 2: The singular values of a matrix (in particular the data matrix) are invariant to any orthogonal transformation of this matrix and thus the empirical dimension is invariant to such transformation.

Proof of Property 3: If $\{\boldsymbol{v}_i\}_{i=1}^N$ are contained in a d-subspace, then since these form the columns of $\boldsymbol{A}$, $rank(\boldsymbol{A}) \leq d$. Since $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal, $rank(\boldsymbol{A}) = rank(\boldsymbol{\Sigma})$. In particular, $\boldsymbol{A}$ has at most d singular values. Let $\boldsymbol{\sigma}$ be the vector of singular values of $\boldsymbol{A}$, and let $1_{\boldsymbol{\sigma}}$ be the indicator vector of $\boldsymbol{\sigma}$[10] .

The generalized Hölders Inequality [23, pg. 10] states that if:

$$p_1, p_2 \in (0, \infty] \quad \text{and} \quad \frac{1}{p_1} + \frac{1}{p_2} = \frac{1}{r} \tag{2.12}$$

then

$$\|f_1 f_2\|_r \leq \|f_1\|_{p_1} \|f_2\|_{p_2} \text{ for any functions } f_1 \text{ and } f_2. \tag{2.13}$$

To apply this result to vectors, we view them as functions over the set $\{1, 2, ..., D\}$ with counting measure.

Let $p_1 = 1$, $p_2 = \frac{\epsilon}{1-\epsilon}$, $r = \epsilon$. Also let $f_1 = 1_{\boldsymbol{\sigma}}$, $f_2 = \boldsymbol{\sigma}$. These values satisfy (2.12).

[10] $1_{\boldsymbol{\sigma}}$ has a 1 in each coordinate where $\boldsymbol{\sigma}$ has a non-zero element, and 0's in all other coordinates.

We therefore get:

$$\frac{\|\boldsymbol{\sigma}\|_\epsilon}{\|\boldsymbol{\sigma}\|_{\frac{\epsilon}{1-\epsilon}}} \leq \|1_{\boldsymbol{\sigma}}\|_1 = (\#\text{ of non-zero sing. values of } \boldsymbol{A}) \leq d. \tag{2.14}$$

Proof of Property 4: By hypothesis, the data vectors $\{\boldsymbol{v}_i\}_{i=1}^N$ are i.i.d. and sampled according to probability measure μ, where μ is sub-Gaussian, non-degenerate, and spherically symmetric in a d-subspace of $\mathbb{R}^D$. We define the nth data matrix:

$$\boldsymbol{A}_n = \begin{bmatrix} \uparrow & \uparrow & \uparrow & & \uparrow \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 & \cdots & \boldsymbol{v}_n \\ \downarrow & \downarrow & \downarrow & & \downarrow \end{bmatrix}.$$

Then $\boldsymbol{\Sigma}_n := (\frac{1}{n})\boldsymbol{A}_n\boldsymbol{A}_n^T$ is the nth sample covariance matrix of our data set. Also, let $\boldsymbol{v}$ be a random variable with probability measure μ. Then $\boldsymbol{\Sigma} := E[\boldsymbol{v}\boldsymbol{v}^T]$ is the covariance matrix of the distribution. A consequence of μ being spherically symmetric in a d-subspace is that after an appropriate rotation of space, $\boldsymbol{\Sigma}$ is diagonal with a fixed constant in d of its diagonal entries and 0 in all other locations. We are trying to prove a result about empirical dimension, which is scale invariant and invariant under rotations of space. Because of these two properties we can assume that the appropriate rotation and scaling has been done so that $\boldsymbol{\Sigma}$ is diagonal with value 1 in d diagonal entries and 0 in all others. Without any loss of generality, we assume that the first d diagonal entries are the non-zero ones.

Let $\boldsymbol{\sigma}_n = (\sigma_{n,1}, \sigma_{n,2}, ..., \sigma_{n,D})^T$, $n \geq D$, denote the vector of singular values of the matrix $\boldsymbol{A}_n$. Our first task will be to show that $\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}$ converges in probability (as $n \to \infty$) to the vector:

$$(\underbrace{1, 1, ..., 1}_{d}, 0, ..., 0)^T. \tag{2.15}$$

To accomplish our task, we will first relate $\boldsymbol{\sigma}_n$ to the vector of singular values of $\boldsymbol{\Sigma}_n$, and then use a result showing that $\boldsymbol{\Sigma}_n$ converges to $\boldsymbol{\Sigma}$ as $n \to \infty$.

It is clear that the vector of singular values of $\boldsymbol{\Sigma}_n$, which we will denote by $\boldsymbol{\psi}$, is given by:

$$\boldsymbol{\psi} = \frac{1}{n}\left(\sigma_{n,1}^2, \sigma_{n,2}^2, ..., \sigma_{n,D}^2\right)^T. \tag{2.16}$$

Next, we will need the following result regarding covariance estimation. This is Corollary 5.50 of [16], adapted to be consistent with our notation.

Lemma 1 (Covariance Estimation): Consider a sub-Gaussian distribution in $\mathbb{R}^D$ with covariance matrix $\mathbf{\Sigma}$. Let $\gamma \in (0,1)$, and $t \geq 1$. If $n > C(t/\gamma)^2 D$, then with probability at least $1 - 2e^{-t^2 D}$, $\|\mathbf{\Sigma}_n - \mathbf{\Sigma}\|_2 \leq \gamma$, where $\|\cdot\|_2$ denotes the spectral norm (i.e., largest singular value of the matrix). The constant C depends only on the sub-Gaussian norm of the distribution.

In our problem, we are applying this lemma to the distribution μ. Let $\gamma \in (0,1)$ be given. If

$$n > C(t/\gamma)^2 D, \tag{2.17}$$

then $\|\mathbf{\Sigma}_n - \mathbf{\Sigma}\|_2 \leq \gamma$ with probability at least $1 - 2e^{-t^2 D}$. The 2-norm of the difference of two matrices bounds the differences of their individual singular values. We will use the following result to make this precise:

Lemma 2 [24]: Let $\sigma_i(\bullet)$ denote the ith largest singular value of an arbitrary m-by-n matrix. Then: $|\sigma_i(\mathbf{B} + \mathbf{E}) - \sigma_i(\mathbf{B})| \leq \|\mathbf{E}\|_2$, for each i.

Because $\mathbf{\Sigma}$ is diagonal with only values 1 and 0 on the diagonal, the singular values of $\mathbf{\Sigma}$ are simply these diagonal values. We will use $\mathbf{1}_{i \in 1:d}$ to denote the i'th singular value of $\mathbf{\Sigma}$.

Setting $\mathbf{B} = \mathbf{\Sigma}_n$ and $\mathbf{E} = \mathbf{\Sigma} - \mathbf{\Sigma}_n$, in lemma 2 we get: $\|\mathbf{\Sigma}_n - \mathbf{\Sigma}\|_2 \leq \gamma \Rightarrow |(1/n)\sigma_{n,i}^2 - \mathbf{1}_{i \in 1:d}| \leq \|\mathbf{\Sigma}_n - \mathbf{\Sigma}\|_2 \leq \gamma$, for each i. This implies that:

$$\frac{\sigma_{n,i}}{\sqrt{n}} \in \begin{cases} \left[\sqrt{1-\gamma}, \sqrt{1+\gamma}\right], & \text{if } i \leq d; \\ \left[0, \sqrt{\gamma}\right], & \text{if } i > d. \end{cases} \tag{2.18}$$

Notice that as $\gamma \to 0$, $\frac{\sigma_{n,i}}{\sqrt{n}}$ approaches $\mathbf{1}_{i \in 1:d}$. Specifically, for any desired tolerance, $\eta > 0$, and any desired certainty, ξ, n can be chosen large enough that with probability greater than ξ, $\left|\mathbf{1}_{i \in 1:d} - \frac{\sigma_{n,i}}{\sqrt{n}}\right| < \eta$, simultaneously for each i. It follows from this that the vector $\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}$ converges in probability to (2.15) as $n \to \infty$.

Finally, $\hat{d}_{\epsilon,n} = \frac{\|\boldsymbol{\sigma}_n\|_\epsilon}{\|\boldsymbol{\sigma}_n\|_{\frac{\epsilon}{1-\epsilon}}} = \frac{\left(\frac{1}{\sqrt{n}}\right)\|\boldsymbol{\sigma}_n\|_\epsilon}{\left(\frac{1}{\sqrt{n}}\right)\|\boldsymbol{\sigma}_n\|_{\frac{\epsilon}{1-\epsilon}}} = \frac{\|\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}\|_\epsilon}{\|\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}\|_{\frac{\epsilon}{1-\epsilon}}}$. Thus, $\hat{d}_{\epsilon,n}$ is a continuous

function of the vector $\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}$. Hence, since $\frac{\boldsymbol{\sigma}_n}{\sqrt{n}}$ converges to $\mathbf{1}_{i\in 1:d}$ as $n \to \infty$, $\hat{d}_{\epsilon,n}$ converges in probability to

$$\left(\frac{\|(1,1,...,1,0,...,0)\|_\epsilon}{\|(1,1,...,1,0,...,0)\|_{(\frac{\epsilon}{1-\epsilon})}} \right) = \frac{d^{\frac{1}{\epsilon}}}{d^{\frac{1-\epsilon}{\epsilon}}} = d^{\frac{1}{\epsilon}-\frac{1-\epsilon}{\epsilon}} = d. \tag{2.19}$$

2.8.2 Proof of Theorem 2

Recall that Π_{Nat} denotes the natural partition of the data set. First, we notice that $GD(\Pi_{Nat}) = \|(d_1, d_2, ..., d_K)\|_p$, where d_k is the true dimension of set k of the partition. Notice that d_k cannot exceed d since μ_k is supported by L_k, a d-subspace. Furthermore, since μ_k does not concentrate mass on subspaces it is a probability 0 event that all N_k points from L_k exist in a proper subspace of L_k. Thus, for the natural partition, d_k is almost surely d, for each k. Hence, $GD(\Pi_{Nat})$ is almost surely $\|(d, d, ..., d)\|_p = (Kd^p)^{1/p} = K^{1/p}d$.

Next, we will find a lower bound for the global dimension of any non-natural partition of the data, and show that if p meets the hypothesis criteria, the lower bound we get is greater than $K^{1/p}d = GD(\Pi_{Nat})$. To accomplish this we need the following lemma.

Lemma 1 *If* $\Pi \neq \Pi_{Nat}$ *then* Π *almost surely has one set with dimension at least* $d+1$.

Before proving the lemma, observe that a consequence is that if $\Pi \neq \Pi_{Nat}$, then with probability 1:

$$GD(\Pi) \geq \|(?, ..., ?, d+1, ?, ..., ?)\|_p \geq d+1. \tag{2.20}$$

Then, from our hypothesis:

$$\begin{aligned} & p > ln(K)/(ln(d+1) - ln(d)) \\ & \implies \left(\frac{d+1}{d}\right)^p > K \\ & \implies d+1 > K^{1/p}d. \end{aligned} \tag{2.21}$$

Hence,

$$GD(\Pi) \geq d+1 > K^{1/p}d = GD(\Pi_{Nat}). \tag{2.22}$$

Thus, if we show Lemma 1, the proof of the theorem follows. To prove Lemma 1 we

require an a simpler lemma:

Lemma 2 *If a set Q in Π has fewer than d points from a subspace L_i, then either Q has dimension at least $d+1$ or adding another point from L_i to Q (an R.V. X with probability measure μ_i, independent from all other samples) will almost surely increase the dimension of Q by 1.*

proof 1 *If* $\dim(Q) \leq d$ *then Q has dimension strictly less than the ambient space ($\mathbb{R}^D$). Observe that* $\text{span}(Q)$ *is a linear subspace of $\mathbb{R}^D$, which a.s. does not contain L_i. We cannot have proper containment since* $\dim(L_i) = d \geq \dim(Q)$*. Also, we have fewer than d points from L_i in Q, and each other point in Q lies in L_i with probability 0 (All μ_i do not concentrate mass on subspaces). Thus,* $\text{span}(Q)$ *a.s. does not equal L_i.*

Therefore, if we intersect L_i with $\text{span}(Q)$ *we get a proper subspace of L_i; call it $\bar{L}$. We note that $\mu_i(\bar{L}) = 0$ since μ_i does not concentrate on subspaces. Thus, since X has probability measure μ_i, X a.s. lies outside the intersection of L_i and* $\text{span}(Q)$*. It follows that if we add X to Q, the dimension of Q a.s. increases by 1.*

□

Now we prove Lemma 1. We will assume all sets in Π have dimension less than $d+1$ and pursue a contradiction. By hypothesis, our set $\{\boldsymbol{v}_n\}_{n=1}^N$ contains at least $d+1$ points from each subspace L_i. Since $\Pi \neq \Pi_{Nat}$, there is some subspace L^* whose points are assigned to 2 or more distinct sets in Π. Let $\boldsymbol{v}^*$ be a point from L^*. Now, choose d points from each L_i and denote this collection of Kd points $\{y_1, y_2, ..., y_{Kd}\}$. When making this selection, ensure that v^* is not chosen and that of the points selected from L^*, not all of them are assigned to the same set in Π as $\boldsymbol{v}^*$. Notice that Π induces a partition on $\{y_1, y_2, ..., y_{Kd}\}$.

Select any point y_i and remove it from the set $\{y_1, y_2, ..., y_{Kd}\}$. Since we are assuming that each set in Π has dimension less than $d+1$, Lemma 2 implies that the set in Π to which y_i belongs will have its dimension decrease by 1. Now select another point y_j and remove it. Lemma 2 still applies and so the set to which y_j belonged will have its dimension decrease by 1. We can repeat this until all Kd points have been removed. Since each removal decreases the dimension of some set in Π by 1 it follows that before any removals the sum of the dimensions of all sets in Π was at least Kd. Since each of

the K sets in Π had dimension d or less, we conclude that in fact each set must have had dimension exactly d.

Now, consider our set $\{y_1, y_2, ..., y_{Kd}\}$ and add in v^*. By our choice of v^*, Lemma 2 implies that its addition a.s. increases the dimension of its target set in Π by 1 (to $d+1$). Adding in all remaining points from $\{\boldsymbol{v}_n\}_{n=1}^N$ will only increase the dimensions of the sets in Π. Thus, we almost surely have a set of dimension at least $d+1$ in Π, contradicting our hypothesis.

□

2.8.3 Proof of Theorem 3

Recall that the soft partition is stored in a membership matrix $\boldsymbol{M}$. Specifically, the (k,n)'th element of $\boldsymbol{M}$, denoted m_k^n, holds the "probability" that vector $\boldsymbol{v}_n$ belongs to cluster k. Thus, each column of $\boldsymbol{M}$ forms a probability vector.

Hence, global dimension is a real-valued function of the matrix $\boldsymbol{M}$. We will think of the membership matrix as being vectorized, so that the domain of optimization can be thought of as a subset of $\mathbb{R}^{NK}$. However, we will not explicitly vectorize the membership matrix. Thus, when we talk about the *gradient* of global dimension, we are referring to another K-by-N matrix, where the (k,n)'th element is the derivative of global dimension w.r.t. m_k^n.

To differentiate global dimension we must be able to differentiate the singular values of a matrix w.r.t. each element of that matrix. A treatment of this is available in [25].

To begin, recall the definition of GD:

$$GD = \left\| \begin{array}{c} \hat{d}_\epsilon^1 \\ \hat{d}_\epsilon^2 \\ \vdots \\ \hat{d}_\epsilon^K \end{array} \right\|_p = \left((\hat{d}_\epsilon^1)^p + (\hat{d}_\epsilon^2)^p + ... + (\hat{d}_\epsilon^K)^p \right)^{1/p}. \tag{2.23}$$

We will denote the thin SVD (only D columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are used) of $\boldsymbol{A}_k$:

$$\boldsymbol{A}_k = \boldsymbol{U}_k \boldsymbol{\Sigma}_k {\boldsymbol{V}_k}^T. \tag{2.24}$$

Also, we will let σ_j^i refer to the (j,j)'th element of $\mathbf{\Sigma}_i$. Then, using the chain rule:

$$\frac{\partial GD}{\partial m_k^n} = \frac{\partial GD}{\partial \hat{d}_\epsilon^1}\frac{\partial \hat{d}_\epsilon^1}{\partial m_k^n} + \frac{\partial GD}{\partial \hat{d}_\epsilon^2}\frac{\partial \hat{d}_\epsilon^2}{\partial m_k^n} + ... + \frac{\partial GD}{\partial \hat{d}_\epsilon^K}\frac{\partial \hat{d}_\epsilon^K}{\partial m_k^n}. \tag{2.25}$$

From (2.23) we can compute $\frac{\partial GD}{\partial \hat{d}_\epsilon^i}$ rather easily:

$$\begin{aligned}\frac{\partial GD}{\partial \hat{d}_\epsilon^i} =& \frac{1}{p}\left((\hat{d}_\epsilon^1)^p + (\hat{d}_\epsilon^2)^p + ... + (\hat{d}_\epsilon^K)^p\right)^{\frac{1}{p}-1} p(\hat{d}_\epsilon^i)^{p-1} \\ =& (\hat{d}_\epsilon^i)^{p-1}\left((\hat{d}_\epsilon^1)^p + (\hat{d}_\epsilon^2)^p + ... + (\hat{d}_\epsilon^K)^p\right)^{\frac{1}{p}-1}.\end{aligned} \tag{2.26}$$

Next, we expand the other components of (2.25):

$$\frac{\partial \hat{d}_\epsilon^i}{\partial m_k^n} = \sum_{j=1}^{D}\frac{\partial \hat{d}_\epsilon^i}{\partial \sigma_j^i}\frac{\partial \sigma_j^i}{\partial m_k^n}. \tag{2.27}$$

We now use the definition of $\hat{d}_\epsilon^i$ to compute the first factor of each term as follows:

$$\begin{aligned}\frac{\partial \hat{d}_\epsilon^i}{\partial \sigma_j^i} =& \frac{\|\boldsymbol{\sigma}_i\|_\delta \frac{\partial}{\partial \sigma_j^i}\left(\left((\sigma_1^i)^\epsilon + ... + (\sigma_D^i)^\epsilon\right)^{1/\epsilon}\right)}{\|\boldsymbol{\sigma}_i\|_\delta^2} - \\ & \frac{\|\boldsymbol{\sigma}_i\|_\epsilon \frac{\partial}{\partial \sigma_j^i}\left(\left((\sigma_1^i)^\delta + ... + (\sigma_D^i)^\delta\right)^{1/\delta}\right)}{\|\boldsymbol{\sigma}_i\|_\delta^2} \\ =& \frac{\|\boldsymbol{\sigma}_i\|_\delta \left((\sigma_1^i)^\epsilon + ... + (\sigma_D^i)^\epsilon\right)^{\frac{1-\epsilon}{\epsilon}}\left(\sigma_j^i\right)^{\epsilon-1}}{\|\boldsymbol{\sigma}_i\|_\delta^2} - \\ & \frac{\|\boldsymbol{\sigma}_i\|_\epsilon \left((\sigma_1^i)^\delta + ... + (\sigma_D^i)^\delta\right)^{\frac{1-\delta}{\delta}}\left(\sigma_j^i\right)^{\delta-1}}{\|\boldsymbol{\sigma}_i\|_\delta^2} \\ =& \left(\frac{1}{\|\boldsymbol{\sigma}_i\|_\delta^2}\right)\|\boldsymbol{\sigma}_i\|_\delta \|\boldsymbol{\sigma}_i\|_\epsilon^{1-\epsilon}\left(\sigma_j^i\right)^{\epsilon-1} - \\ & \left(\frac{1}{\|\boldsymbol{\sigma}_i\|_\delta^2}\right)\|\boldsymbol{\sigma}_i\|_\epsilon \|\boldsymbol{\sigma}_i\|_\delta^{1-\delta}\left(\sigma_j^i\right)^{\delta-1} \\ =& C_1^i\left(\sigma_j^i\right)^{\epsilon-1} - C_2^i\left(\sigma_j^i\right)^{\delta-1},\end{aligned} \tag{2.28}$$

where

$$C_1^i = \left(\frac{\|\boldsymbol{\sigma}_i\|_\epsilon^{1-\epsilon} \|\boldsymbol{\sigma}_i\|_\delta}{\|\boldsymbol{\sigma}_i\|_\delta^2} \right), \qquad C_2^i = \left(\frac{\|\boldsymbol{\sigma}_i\|_\epsilon \|\boldsymbol{\sigma}_i\|_\delta^{1-\delta}}{\|\boldsymbol{\sigma}_i\|_\delta^2} \right). \tag{2.29}$$

Next, we must evaluate the second factor in each term of (2.27). Recall that σ_j^i is the j'th largest singular value of the matrix $\boldsymbol{A}_i$. To achieve the next step, we must observe that each singular value of $\boldsymbol{A}_i$ depends, in general, on each element of the matrix $\boldsymbol{A}_i$. We can then compute the derivative of each element of the matrix $\boldsymbol{A}_i$ w.r.t. each membership variable, m_k^n. We will denote the (α, β)'th element of the matrix $\boldsymbol{A}_i$ by $\boldsymbol{A}_{i(\alpha,\beta)}$. Using the chain rule:

$$\frac{\partial \sigma_j^i}{\partial m_k^n} = \sum_{\beta=1}^{N} \sum_{\alpha=1}^{D} \frac{\partial \sigma_j^i}{\partial \boldsymbol{A}_{i(\alpha,\beta)}} \frac{\partial \boldsymbol{A}_{i(\alpha,\beta)}}{m_k^n}. \tag{2.30}$$

A powerful result [25, eqn. 7] allows us to express the partial derivative of each singular value, σ_j^i, w.r.t. a given matrix element in terms of the already-known SVD of $\boldsymbol{A}_i$:

$$\frac{\partial \sigma_j^i}{\partial \boldsymbol{A}_{i(\alpha,\beta)}} = \boldsymbol{U}_{i(\alpha,j)} \boldsymbol{V}_{i(\beta,j)}. \tag{2.31}$$

The second factor in each term of (2.30) can be evaluated directly from the definition of $\boldsymbol{A}_k$:

$$\frac{\partial \boldsymbol{A}_{i(\alpha,\beta)}}{\partial m_k^n} = \begin{cases} 0, & \text{if } n \neq \beta \text{ or if } i \neq k; \\ \boldsymbol{v}_n \cdot \hat{\mathbf{e}_\alpha}, & \text{if } n = \beta \text{ and } i = k, \end{cases} \tag{2.32}$$

where $\hat{\mathbf{e}_\alpha}$ denotes the α'th standard basis vector (1 in position α and 0's everywhere else).

We are now in a position to work backwards and construct the partial derivative of GD w.r.t. m_k^n. In what follows, δ_{ik} is equal to 1 if $i = k$ and is 0 otherwise (this is not to be confused with the un-subscripted δ, which is shorthand for $\epsilon/(1-\epsilon)$). Also, for notational convenience, we use Matlab notation to represent a row or column of a

matrix ($B_{(w,:)}$ and $B_{(:,w)}$, respectively). We fist compute $\partial\sigma_j^i/\partial m_k^n$ as follows:

$$\begin{aligned}
\frac{\partial\sigma_j^i}{\partial m_k^n} =& \sum_{\alpha=1}^{D} \frac{\partial\sigma_j^i}{\partial \boldsymbol{A}_{i(\alpha,n)}} \frac{\partial \boldsymbol{A}_{i(\alpha,n)}}{m_k^n} \\
=& \sum_{\alpha=1}^{D} \boldsymbol{U}_{i(\alpha,j)} \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{v}_n \cdot \hat{\mathbf{e}_\alpha}\right) \delta_{ik} \\
=& \begin{bmatrix} \boldsymbol{U}_{i(1,j)}\boldsymbol{V}_{i(n,j)} \\ \boldsymbol{U}_{i(2,j)}\boldsymbol{V}_{i(n,j)} \\ \vdots \\ \boldsymbol{U}_{i(D,j)}\boldsymbol{V}_{i(n,j)} \end{bmatrix} \cdot \boldsymbol{v}_n \delta_{ik} = \boldsymbol{V}_{i(n,j)} \begin{bmatrix} \boldsymbol{U}_{i(1,j)} \\ \boldsymbol{U}_{i(2,j)} \\ \vdots \\ \boldsymbol{U}_{i(D,j)} \end{bmatrix} \cdot \boldsymbol{v}_n \delta_{ik} \\
=& \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) \delta_{ik}.
\end{aligned} \tag{2.33}$$

Then from (2.27), we get

$$\begin{aligned}
\frac{\partial\hat{d}_\epsilon^i}{\partial m_k^n} =& \sum_{j=1}^{D} \frac{\partial\hat{d}_\epsilon^i}{\partial\sigma_j^i} \frac{\partial\sigma_j^i}{\partial m_k^n} \\
=& \sum_{j=1}^{D} \left(C_1^i \left(\sigma_j^i\right)^{\epsilon-1} - C_2^i \left(\sigma_j^i\right)^{\delta-1}\right) \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) \delta_{ik}.
\end{aligned} \tag{2.34}$$

Now we can write:

$$\begin{aligned}
\frac{\partial\hat{d}_\epsilon^i}{\partial m_k^n} =& C_1^i \left(\sum_{j=1}^{D} \left(\sigma_j^i\right)^{\epsilon-1} \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) \delta_{ik}\right) - \\
& C_2^i \left(\sum_{j=1}^{D} \left(\sigma_j^i\right)^{\delta-1} \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) \delta_{ik}\right).
\end{aligned} \tag{2.35}$$

We now simplify the components of (2.35). After some manipulation, and using the notation

$$\left(\boldsymbol{\Sigma}_i\right)^{\epsilon-1} = \begin{bmatrix} (\sigma_1^i)^{\epsilon-1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & (\sigma_D^i)^{\epsilon-1} \end{bmatrix}, \tag{2.36}$$

we can write

$$\begin{aligned}
&\sum_{j=1}^{D} \left(\sigma_j^i\right)^{\epsilon-1} \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) = \\
&\qquad \left[\left(\sigma_1^i\right)^{\epsilon-1} \boldsymbol{V}_{i(n,1)}, ..., \left(\sigma_D^i\right)^{\epsilon-1} \boldsymbol{V}_{i(n,D)}\right] \begin{bmatrix} \boldsymbol{U}_{i(:,1)} \cdot \boldsymbol{v}_n \\ \boldsymbol{U}_{i(:,2)} \cdot \boldsymbol{v}_n \\ \vdots \\ \boldsymbol{U}_{i(:,D)} \cdot \boldsymbol{v}_n \end{bmatrix} \\
&\qquad = \boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\epsilon-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n.
\end{aligned} \tag{2.37}$$

Similarly, we can simplify part of the second term of (2.35):

$$\sum_{j=1}^{D} \left(\sigma_j^i\right)^{\delta-1} \boldsymbol{V}_{i(n,j)} \left(\boldsymbol{U}_{i(:,j)} \cdot \boldsymbol{v}_n\right) = \boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\delta-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n. \tag{2.38}$$

Substituting (2.37) and (2.38) into (2.35) we get

$$\begin{aligned}
\frac{\partial \hat{d}_\epsilon^i}{\partial m_k^n} = &\Big[C_1^i \left(\boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\epsilon-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n\right) - \\
&\qquad C_2^i \left(\boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\delta-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n\right)\Big] \delta_{ik}.
\end{aligned}$$

With this expression we are ready to evaluate (2.25) as follows:

$$\begin{aligned}
&\frac{\partial GD}{\partial m_k^n} = \sum_{i=1}^{K} \frac{\partial GD}{\partial \hat{d}_\epsilon^i} \frac{\partial \hat{d}_\epsilon^i}{\partial m_k^n} \\
=&\sum_{i=1}^{K} (\hat{d}_\epsilon^i)^{p-1} \left((\hat{d}_\epsilon^1)^p + ... + (\hat{d}_\epsilon^K)^p\right)^{\frac{1}{p}-1} \delta_{ik} \cdot \\
&\Big[C_1^i \left(\boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\epsilon-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n\right) - C_2^i \left(\boldsymbol{V}_{i(n,:)} \left(\boldsymbol{\Sigma}_i\right)^{\delta-1} \left(\boldsymbol{U}_i\right)^T \boldsymbol{v}_n\right)\Big] \\
=&(\hat{d}_\epsilon^k)^{p-1} \left((\hat{d}_\epsilon^1)^p + ... + (\hat{d}_\epsilon^K)^p\right)^{\frac{1}{p}-1} \cdot \\
&\Big[C_1^k \boldsymbol{V}_{k(n,:)} \left(\boldsymbol{\Sigma}_k\right)^{\epsilon-1} \left(\boldsymbol{U}_k\right)^T \boldsymbol{v}_n - C_2^k \boldsymbol{V}_{k(n,:)} \left(\boldsymbol{\Sigma}_k\right)^{\delta-1} \left(\boldsymbol{U}_k\right)^T \boldsymbol{v}_n\Big] \\
=&(\hat{d}_\epsilon^k)^{p-1} \| \left(\hat{d}_\epsilon^1, ..., \hat{d}_\epsilon^K\right) \|_p^{1-p} \cdot \\
&\Big[\boldsymbol{V}_{k(n,:)} \left(C_1^k \left(\boldsymbol{\Sigma}_k\right)^{\epsilon-1} - C_2^k \left(\boldsymbol{\Sigma}_k\right)^{\delta-1}\right) \left(\boldsymbol{U}_k\right)^T \boldsymbol{v}_n\Big] \\
=&(\hat{d}_\epsilon^k)^{p-1} \| \left(\hat{d}_\epsilon^1, ..., \hat{d}_\epsilon^K\right) \|_p^{1-p} \boldsymbol{V}_{k(n,:)} \boldsymbol{D}_k \left(\boldsymbol{U}_k\right)^T \boldsymbol{v}_n,
\end{aligned} \tag{2.39}$$

where

$$\boldsymbol{D}_k = \left(C_1^k \left(\boldsymbol{\Sigma}_k \right)^{\epsilon-1} - C_2^k \left(\boldsymbol{\Sigma}_k \right)^{\delta-1} \right). \tag{2.40}$$

We re-write (2.39) as follows:

$$\frac{\partial GD}{\partial m_k^n} = \boldsymbol{V}_{k(n,:)} \left((\hat{d}_\epsilon^k)^{p-1} \| \left(\hat{d}_\epsilon^1, ..., \hat{d}_\epsilon^K \right) \|_p^{1-p} \boldsymbol{D}_k \left(\boldsymbol{U}_k \right)^T \right) \boldsymbol{A}_{(:,n)}. \tag{2.41}$$

2.8.4 Experiment Setup

For our comparison on the outlier-free RAS database, we include the following methods: GDM, SCC [12], MAPA [14], Sparse Subspace Clustering (SSC) [19], Spectral Local Best-fit Flats (SLBF) [13], Low-Rank Representation (LRR) [20], RAS [7], and HOSC [10]. For each method in our comparisons (outlier-free and our tests with outliers) the implementation of each algorithm is that of the original authors. Most of these codes were found on the respective authors websites, although some codes were obtained from the authors directly when they could not be found online. As a matter of good testing methodology, we ran each method 10 times on each file. This is because we want to avoid capturing any fluke occurrences of any method, but instead seek the "usual case" results (this is important for repeatability of the results). Of the 10 runs for a given file and method, the median error is reported. For deterministic methods, we get the same exact results for each run. GDM involves randomness, but the average standard deviation of the misclassification errors was 0.73%, meaning that it behaved very consistently in the experiment. GDM was run with $n_1 = 10$. The other parameters (ϵ and p) are fixed throughout all experiments and are addressed earlier. SCC was run with $d = 3$ for the linearly embedded data (this was found to give the best results), and $d = 7$ for the nonlinearly embedded data (as recommended in [11]). MAPA was run without any special parameters. SSC was run with no data projection (because of the low ambient dimension to start with), the affine constraint enabled (we tried it both ways and this gave better results), optimization method = "Lasso" (Default for authors code), and parameter lambda = 0.001 (found through trial and error). SLBF was run with $d = 3$ for the linearly embedded data and $d = 6$ for the nonlinearly embedded data and σ was set to $20,000$ for both cases (d and σ were selected by trial and error to give the best results). LRR was run with $\lambda = 100$ for the linear case and $\lambda = 10000$ for the

non-linear case (these seemed to give the best results). RAS proved rather sensitive to its main parameter ("angleTolerance"), and no single value gave good across-the-board results. We ran with all default parameters and many other combinations. The results presented were generated using angleTolerance = 0.22 and boundaryThreshold = 5, as this combination gave the best results from our tests (better than the algorithms defaults). HOSC was run with η automatically selected by the algorithm from the range $[0.0001, 0.1]$. The parameter "knn" was set to 20, and the default "heat" kernel was chosen. The algorithm was tried with d set to 2 and 3. Both of these cases are presented. $d = 2$ gave better results, but the authors of HOSC argue for using $d = 3$ in this setting.

For our comparison on the outlier-free Hopkins 155 database, the algorithms that were selected for the comparison were run once on each of the 155 data files. The mean and median performance for each category is reported. GDM was run with $n_1 = 30$ to improve reliability. All other parameters were left fixed, and (as before) the non-linearly embedded data was used. Each competing 2-view method was run on the non-linearly embedded data with the same parameters that gave the best performance on the RAS database. The competing n-view methods have their parameters given in the results tables.

For our outlier comparison on the corrupted RAS database, we ran GDM with $n_1 = 30$ (same as for the Hopkins 155 database). For the naive approach, we used $\alpha = 0.02$. For "GDM - Known Fraction" we rejected 20% of the dataset. For "GDM - Model Reassign" we used $\kappa = 0.05$. "GDM - Classic" was the same algorithm as in the outlier-free comparisons and so had no extra parameters. RAS was run with angleTolerance = 0.22 and boundaryThreshold = 5 (same as in the outlier-free tests). We ran LRR with $\lambda = 0.1$, and outlierThreshold = 0.138 (these gave the best results of the combinations we tried). HOSC was run with $d = 2$ (which gave the best results in the outlier-free case) and $\alpha = 0.11$.

The code for GDM can be found on our supplemental webpage. For each of the algorithms used in our comparisons, we have made an effort to provide (on the supplemental webpage) the code or a link to where the code can be found.

Chapter 3

Part II: Using Subspace Constraints To Improve General-Purpose Feature Tracking

Based on work done with Gilad Lerman and Arthur Szlam

3.1 Introduction

Feature tracking in video is an important computer vision task, often used as the first step in finding structure from motion or simultaneous location and mapping (SLAM). The celebrated Kanade-Lucas-Tomasi algorithm [26, 27, 28] tracks feature points by searching for matches between templates representing each feature and a frame of video. Despite many other alternatives and improvement, it is still one of the best video feature

tracking algorithms [1].[1] However, there are several realistic scenarios when Lucas-Kanade and many of its alternatives do not perform well: poor lighting conditions, noisy video, and when there are transient occlusions that need to be ignored. In order to deal with such scenarios more robustly it would be useful to allow the feature points to communicate with each other to decide how they should move as a group, so as to respect the underlying three dimensional geometry of the scene.

This underlying geometry constrains the trajectories of the track points to have a low-rank structure; see [8, 2] for the case when tracking a single rigid object under an affine camera model, and [29, 30, 31, 32] for non-rigid motion and the perspective camera. In this work we will combine the low-rank geometry of the cohort of tracked features with the successful non-linear single feature tracking framework of Lucas and Kanade [26] by adding a low-rank regularization penalty in the tracking optimization problem. To accommodate dynamic scenes with non-trivial motion we apply our rank constraint over a sliding window, so that we only consider a small number of frames at a given time (this is a common idea for dealing with non-rigid motions [33, 34, 35]). We demonstrate very strong performance in rigid environments as well as in scenes with multiple and/or non-rigid motion (since the trajectories of all features are still low rank for short time intervals). We describe experiments with several choices of low-rank regularizers (which are local in time), using a unified optimization framework that allows real time regularized tracking on a single CPU core.

3.1.1 Relationship With Previous Work

Geometric structures (and low-rank structures in particular) have been effectively utilized for the problem of optical flow estimation. Irani [36] showed how 3-d constraints in the real world and various camera models imply the existence of a low-rank constraint on the flow problem. Brand extended Irani's work to non-rigid motions, while developing a robust, subspace-estimating, flow-based tracker via an incremental singular value

[1] Feature tracking should be distinguished from object tracking, where there has been significant progress in the development of novel algorithms that significantly improve previous efforts.

Acknowledgements: This work was supported by NSF award DMS-09-56072, the Sloan foundation, the University of Minnesota Doctoral Dissertation Fellowship Program, and the Feinberg Foundation Visiting Faculty Program Fellowship of the Weizmann Institute of Science.

Supp. web page: http://www.math.umn.edu/∼lerman/RCTracking/

decomposition with missing values as well as by learning an object model [37, 38, 39, 29]. Torresani et al. [30] also extended Irani's work to non-rigid motions by applying rank-bounds for recovering 3D non-rigid motions. More recently, Garg et al. [40] introduced hard subspace constraints for long range optical flow estimation in a variational scheme. Garg et al. [35] improved the performance of this former work by making the constraint weak (as an energy regularizer), using a robust energy term and allowing more general basis terms. At last, Ricco and Tomasi [34] proposed a Lagrangian approach for long range motion estimation that allows more reliable detection of occlusion. It estimates a basis for a low-dimensional subspace of the trajectories (as in [35]) and employs a variational method to solve for the best-fit coefficients of the motion trajectories in this basis.

In optical flow estimation the goal is to find displacements of features between consecutive frames, while assuming that the flow field is locally nearly constant. Although the goal in the feature tracking problem is similar, it does not require estimating the flow by enforcing the brightness constancy constraint or a weaker version of it. The subspace constraints above were translated by Irani [36] to an image brightness constraint. However, small errors in the flow field in each frame from this approach lead to the accumulation of errors in the trajectories obtained by integrating the flow. These errors are unacceptable for tracking. Weak versions of this constraint for estimating flow along many frames (as in [29, 34, 35]) require rather dense trajectories, which represent continuous regions in the image frame. Indeed, they are based on either continuous variational methods [34, 35] (which often track all pixels in the image domain) or careful model estimation [29] (which requires sufficiently dense sampling from objects in the videos).

In tracking, one instead uses a formulation that allows for very precise feature registration (like the Lucas-Kanade tracker [26]), and there is no need to linearize the image to solve an approximation to the feature displacement problem. It is desirable to have a sparse set of features and track them only in local neighborhoods to allow real time implementation. There is not a canonical method for introducing an explicit low rank constraint as in [36]. We will argue below that any strict subspace constraint is not ideal in the tracking problem and will promote a soft constraint. This soft constraint is different than the ones advocated for flow estimation in both [34] and [35] since they

carefully learn local basis elements and require dense feature sampling.

Torresani and Bregler [41] suggested the partial application of hard low-rank constraints to improve tracking (applying rank bounds as in [30]). They rely on initial Lucas-Kanade tracking [26] from which "reliable" features are identified and used to estimate a model for the scene. They used their constraint to re-track the "unreliable" features (the trajectories are now confined to a known subspace). Since they search in the space of trajectories, their minimization strategy is completely different than ours. Their tracker is also non-casual since it needs the full sequence to start tracking, so a real-time implementation is not possible. This work was extended in [42] to develop a causal tracker in the same spirit that also does not rely on a set of "reliable" feature tracks. However, both methods require setting the rank of the constraint a-priori and they impose the constraint over very long time spans (up to the entire sequence), making the algorithms less applicable to dynamic scenes.

Buchanan and Fitzgibbon [33] continuously update a non-rigid motion model over a sliding temporal window. This motion model is used as a motion prior in a conventional Bayesian template tracker for a single feature. The local information is combined with weaker global low rank approximation for the set of initial local trajectories (in the spirit of [29, 34, 35], while different than the low rank constraint of this paper). Similarly to [29] this low rank constraint guides the tracking via Bayesian modeling.

Another line of work takes tracked feature points in videos, and then uses the underlying subspace structure of rigid bodies to segment different motions of such bodies [8, 43, 12, 13, 44]. This is related to the large body of work on recovering rigid or non-rigid structure from motion; see [2] or [32] and the references therein. However, these works are highly dependent on good tracking and it would be desirable to simultaneously track and segment motion, or exploit the subspace structure to improve tracking prior to finding structure from motion.

3.2 On Low-Rank Feature Trajectories

Under the affine camera model, the feature trajectories for a set of features from a rigid body should exist in an affine subspace of dimension 3, or a linear subspace of dimension 4 [8, 2]. However, subspaces corresponding to very degenerate motion are

lower-dimensional than those corresponding to general motion [2].

Feature trajectories of non-rigid scenarios exhibit significant variety, but some low-rank models may still be successfully applied to them [29, 30, 31, 32, 35]. Similarly to [33, 35] (though in a different setting) we consider a sliding temporal window, where over short durations the motion is simple and the feature trajectories are of lower rank. The restriction on the length of feature trajectories can also help in satisfying an approximate local affine camera model in scenes which violate the affine camera model. In general, depth disparities give rise to low-dimensional manifolds [2] which are only locally approximated by linear spaces.

At last, even in the case of multiple moving rigid objects, the set of trajectories is still low rank (confined to the union of a few low rank subspaces). In all of these scenarios the low rank is unknown in general.

3.3 Feature Tracking

Notation: A *feature* at a location $\boldsymbol{z}_1 \in \mathbb{R}^2$ in a given $N_1 \times N_2$ frame of an $N_1 \times N_2 \times N_3$ video is characterized by a *template* T, which is an $n \times n$ sub-image of that frame centered at $\boldsymbol{z}_1$ (n is a small integer, generally taken to be odd, so the template has a center pixel). If $\boldsymbol{z}_1$ does not have integer coordinates, T is interpolated from the image. We denote $\Omega = \{1, ..., n\} \times \{1, ..., n\}$ and we parametrize T so that its pixel values are obtained by $\{T(\boldsymbol{u})\}_{\boldsymbol{u} \in \Omega}$.

A classical formulation of the single-feature tracking problem (see e.g., [26]) is to search for the translation $\boldsymbol{x}_1$ that minimizes some distance between a feature's template T at a given frame and the next frame of video translated by $\boldsymbol{x}_1$; we denote this next frame by I. That is, we minimize the *single-feature energy function* $c(\boldsymbol{x}_1)$:

$$c(\boldsymbol{x}_1) = \frac{1}{n^2} \sum_{\boldsymbol{u} \in \Omega} \psi \left(T(\boldsymbol{u}) - I(\boldsymbol{u} + \boldsymbol{x}_1) \right), \tag{3.1}$$

where, for example, $\psi(x) = |x|$ or $\psi(x) = x^2$. To apply continuous optimization we view $\boldsymbol{x}_1$ as a continuous variable and we thus view T and I as functions over continuous domains (implemented with bi-linear interpolation).

3.3.1 Low Rank Regularization Framework

If we want to encourage a low rank structure in the trajectories, we cannot view the tracking of different features as separate problems. For $f \in \{1, 2, ..., F\}$, let $\boldsymbol{x}_f$ denote the position of feature f in the current frame (in image coordinates), and let $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_F) \in \mathbb{R}^{2F}$ denote the joint state of all features in the scene. We define the total energy function as follows:

$$C(\boldsymbol{x}) = \frac{1}{Fn^2} \sum_{f=1}^{F} \sum_{\boldsymbol{u} \in \Omega} \psi \left(T_f(\boldsymbol{u}) - I(\boldsymbol{u} + \boldsymbol{x}_f) \right), \tag{3.2}$$

where $T_f(\boldsymbol{u})$ is the template for feature f. Now, we can impose desired relationships between features in a scene by imposing constraints on the domain of optimization of (3.2).

Instead of enforcing a hard constraint, we add a *penalty term* to (3.2), which increases the cost of states which are inconsistent with low-rank motion. Specifically, we define:

$$\bar{C}(\boldsymbol{x}) = \alpha \sum_{f=1}^{F} \sum_{\boldsymbol{u} \in \Omega} \psi \left(T_f(\boldsymbol{u}) - I(\boldsymbol{u} + \boldsymbol{x}_f) \right) + P(\boldsymbol{x}), \tag{3.3}$$

where $P(\boldsymbol{x})$ is an estimate of, or proxy for, the dimensionality of the set of feature trajectories over the last several frames of video (past feature locations are treated as constants, so this is a function only of the current state, $\boldsymbol{x}$). Notice that we have replaced the scale factor $1/(Fn^2)$ from (3.2) with the constant α, as this coefficient is now also responsible for controlling the relative strength of the penalty term. We will give explicit examples for P in section 3.3.2.

This framework gives rise to two different solutions, characterized by the strength of the penalty term (definition of α). Each has useful, real-world tracking applications. In the first case, we assume that most (but not necessarily all) features in the scene approximately obey a low rank model. This is appropriate if the scene contains non-rigid or multiple moving bodies. We can impose a *weak constraint* by making the penalty term small relative to the other terms. If a feature is strong, it will confidently track the imagery, ignoring the constraint (regardless of whether the motion is consistent with the other features in the scene). If a feature is weak in the sense that we cannot fully

determine its true location by only looking at the imagery, then the penalty term will become significant and encourage the feature to agree with the motion of the other features in the scene.

In the second case, we assume that all features in the scene are supposed to agree with a low rank model (and deviations from that model are indicative of tracking errors). We can impose a strong constraint by making the penalty term large relative to the other terms. No small set of features can overpower the constraint, regardless of how strong the features are. This forces all features to move is a way that is consistent with a simple motion. Thus, a small number of features can even be occluded, and their positions will be predicted by the motion of the other features in the scene. We further explain these two scenarios and demonstrate them with figures in the supplementary material.

3.3.2 Specific Choices of the Low-Rank Regularizer

There is now a large body of work on low rank regularization, e.g., [45, 46, 47]. We will restrict ourselves to showing results using three choices for P described below. Each choice we present defines $P(\boldsymbol{x})$ in terms of a matrix $\boldsymbol{M}$. It is the $2(L+1) \times F$ matrix whose column f contains the feature trajectory for feature f within a sliding window of $L+1$ consecutive frames (current frame and L past frames). Specifically, $\boldsymbol{M} = [m_{i,j}]$, where $(m_{0,f}, m_{1,f})^T$ is the current (variable) position of feature f and $(m_{2l+1,f}, m_{2l+2,f})^T$, $l = 1, ..., L$ contains the x and y pixel coordinates of feature f from l frames in the past (past feature locations are treated as known constants). One may alternatively center the columns of $\boldsymbol{M}$ by subtracting from each column the average of all columns. Most constraints derived for trajectories (assuming, for instance, rigid motion) actually confine trajectories to a low rank *affine* subspace (as opposed to a *linear* subspace). Centering the columns of $\boldsymbol{M}$ transforms an affine constraint into a linear one. Alternatively, one can forgo centering and view an affine constraint as a linear constraint in one dimension higher. We report results for both approaches.

Explicit Factorizations

A simple method for enforcing the structure constraint is to write $\boldsymbol{M} = \boldsymbol{BC}$, where $\boldsymbol{B}$ is a $2(L+1) \times d$ matrix, and $\boldsymbol{C}$ is a $d \times F$ matrix. However, as mentioned in the previous section, because the feature tracks often do not lie exactly on a subspace due

to deviations from the camera model or non-rigidity, an explicit constraint of this form is not suitable.

However, an explicit factorization can be used in a penalty term by measuring the deviation of $\boldsymbol{M}$, in some norm, from its approximate low rank factorization. For example, if we let

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{3.4}$$

denote the SVD of $\boldsymbol{M}$, we can take $P(\boldsymbol{x})$ in (3.3) to be $||\boldsymbol{B}\boldsymbol{C} - \boldsymbol{M}||_*$, where $\boldsymbol{B}$ is the first three or four columns of $\boldsymbol{U}$, and $\boldsymbol{C}$ is the first three or four rows of $\boldsymbol{\Sigma}\boldsymbol{V}^T$. Then this P corresponds to penalizing $\boldsymbol{M}$ via $\sum_{i=d+1}^{F} \sigma_i$, where $\sigma_i = \boldsymbol{\Sigma}_{ii}$ is the i'th singular value of $\boldsymbol{M}$. As above, since the history is fixed, $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}^T$ are functions of $\boldsymbol{x}$.

This approach is the closest analogue of [36] in the tracking setting, next to an explicit rank constraint. It assumes knowledge of the low-rank d. For simplicity, we assume a local rigid model and thus set $d = 3$ when centering M and $d = 4$ when not centering (following [8, 2]).

Nuclear Norm

A popular alternative to explicitly keeping track of the best fit low-dimensional subspace to $\boldsymbol{M}$ is to use the matrix nuclear norm and define

$$P(\boldsymbol{x}) = \|\boldsymbol{M}\|_* = \|\boldsymbol{\sigma}\|_1. \tag{3.5}$$

This is a convex proxy for the rank of $\boldsymbol{M}$ (see e.g., [45, 46]). Here $\boldsymbol{\sigma} = (\sigma_1 \ \sigma_2 \ \ldots \ \sigma_{2(L+1)\wedge F})^T$ is the vector of singular values of $\boldsymbol{M}$, and $||\cdot||_1$ is the l_1 norm. Unlike explicit factorization, where only energy outside the first d principal components of $\boldsymbol{M}$ is punished, the nuclear norm will favor lower-rank $\boldsymbol{M}$ over higher-rank $\boldsymbol{M}$ even when both matrices have rank $\leq d$. Thus, using this kind of penalty will favor simpler track point motions over more complex ones, even when both are technically permissible.

Empirical Dimension

Empirical Dimension [48] refers to a class of dimension estimators depending on a parameter $\epsilon \in (0, 1]$. The empirical dimension of $\boldsymbol{M}$ is defined to be:

$$\hat{d}_\epsilon(\boldsymbol{M}) := \frac{\|\boldsymbol{\sigma}\|_\epsilon}{\|\boldsymbol{\sigma}\|_{\left(\frac{\epsilon}{1-\epsilon}\right)}}. \tag{3.6}$$

Notice that we use norm notation, although $\|\cdot\|_\epsilon$ is only a pseudo-norm. When $\epsilon = 1$, this is sometimes called the "effective rank" of the data matrix [16].

Empirical dimension satisfies a few important properties, which are verified in [48]. First, empirical dimension is invariant under rotation and scaling of a data set. Additionally, in the absence of noise, empirical dimension never exceeds true dimension, but it approaches true dimension as the number of measurements goes to infinity for spherically symmetric distributions. Thus, d_ϵ is a true dimension estimator (whereas the nuclear norm is a proxy for dimension). To use empirical dimension as our regularizer, we define $P(\boldsymbol{x}) = d_\epsilon(\boldsymbol{M})$.

Empirical dimension is governed by its parameter, ϵ. An ϵ near 0 results in a "strict" estimator, which is appropriate for estimating dimension in situations where you have little noise and you expect your data to live in true linear spaces. If ϵ is near 1 then d_ϵ is a lenient estimator. This makes it less sensitive to noise, and more tolerant of data sets that are only approximately linear. In all of the experiments we present, we use $\epsilon = 0.6$, although we found that other tested values also worked well. Refer to §2.3.1 for a more detailed developent of empirical dimension.

3.3.3 Implementation Details

We fix $L = 10$ for the sliding window and let $\psi(x) = |x|$ in (3.3). We use this form for ψ so that all terms in the total energy function behave linearly in a known range of values. If our fit terms behaved quadratically, it would be more challenging to balance them against a penalty term. We also tested a Huber loss function for ψ and have concluded that such a regularization is not needed.

We fix a parameter m for each penalty form (selected empirically - see the supplementary material for our procedure), which determines the strength of the penalty. The

weak and strong regularization parameters are set as follows:

$$\alpha_{weak} = \frac{1}{mn^2} \quad \text{and} \quad \alpha_{strong} = \frac{1}{mFn^2}. \tag{3.7}$$

The weak scaling implies that a perfectly-matched feature will contribute 0 to the total energy, and a poorly-matched feature will contribute an amount on the order of $1/m$ to the total energy. The penalty term will contribute on the order of 1 to the total energy. Since we do not divide the contributions of each feature by the number of features, the penalty terms contribution is comparable in magnitude to that of a single feature. The strong scaling implies that the penalty term is on the same scale as the sum of the contributions of all of the features in the scene.

Minimization Strategy

The total energy function we propose for constrained tracking is non-convex since the contributions from the template fit terms are not convex (even if P is convex); this is also the case with other feature tracking methods, including the Lucas-Kanade tracker. We employ a 1^{st}-order descent approach for driving the energy to a local minimum.

To reduce the computational load of feature tracking, some trackers use 2^{nd}-order methods for optimization (see [1]). This works well when tracking strong features, but in our experience it can be unreliable when dealing with weak or ambiguous features. Since we are explicitly trying to improve tracking accuracy on poor features we opt for a 1^{st}-order descent approach instead.

The simplest 1^{st}-order descent method is (sub)gradient descent. Unfortunately, because there can be a very large difference in magnitude between the contributions of strong and weak features to our total energy, our problem is not well-conditioned. If we pursue standard gradient descent, the strong features dictate the step direction and the weak features have very little effect on it. Ideally, once the strong features are correctly positioned, they will no longer dominate the step direction. If we were able to perfectly measure the gradient of our objective function, this would be the case. In practice, the error in our numerical gradient estimate can be large enough to prevent the strong features from ever relinquishing control over the step direction. The result is that in a scene with both very strong and very weak features, the weak features may not be

tracked.

To remedy this, we compute our step direction by blending the gradient of the energy function with a vector that corresponds to taking equal-sized gradient descent steps separately for each feature. We use a fast line search in each iteration to find the nearest local minimum in the step direction. This compromise approach allows for efficient descent while ensuring that each feature has some control over the step direction (regardless of feature strength).

Because the energy is not convex, it is important to choose a good initial state. We use a combination of two strategies to initialize the tracking: first, we generate our initial guess of $\boldsymbol{x}$ by registering an entire frame of video with the previous (at lower resolution). Secondly, we use multi-resolution, or pyramidal tracking so that approximate motion on a large scale can help us get close to the minimum before we try tracking on finer resolution levels (see [49]).

We now explain the details of the algorithm. Let $\boldsymbol{I}$ denote a full new frame of video and let $\boldsymbol{x}^{\text{prev}}$ be the concatenation of feature positions in the previous frame. We form a pyramid for $\boldsymbol{I}$ where level 0 is the full-resolution image and each higher level m (1 through 3) has half the vertical and half the horizontal resolution of level $m-1$. To initialize the optimization, we take the full frame (at resolution level 3) and register it against the previous frame (also at resolution level 3) using gradient descent and an absolute value loss function. We initialize each features position in the current frame by taking its position in the previous frame and adding the offset between the frames, as found through this registration process). Once we have our initial $\boldsymbol{x}$, we begin optimization on the top pyramid level. When done on the top level, we use the result to initialize optimization on the level below it, and so on until we have found a local minimum on level 0. On any given pyramid level, we perform optimization by iteratively computing a step direction and conducting a fast line search to find a local minimum in the search direction. We impose a minimum and maximum on the number of steps to be performed on each level ($\min_i$ and $\max_i$, respectively). Our termination condition (on a given level) is when the magnitude of the derivative of $\bar{C}$ is not significantly smaller than it was in the previous step. To compute our search direction in each step, we first compute the gradient of $\bar{C}$ (which we will call $D\bar{C}$) and set $\boldsymbol{a} = -D\bar{C}$. We then compute a semi-normalized version of $\boldsymbol{a}$. This is done by breaking it into a collection

of 2-vectors (elements 1 and 2 are together, elements 3 and 4 are together, and so on) and normalizing each of them. We then re-combine the normalized 2-vectors to get $\boldsymbol{b}$. We blend $\boldsymbol{a}$ with $\boldsymbol{c}$ to compute our step direction. Algorithm 2 summarizes the full process. Source code for our implementation of this algorithm will be available on the first authors web page.

Algorithm 2 Optimization of rank-penalized energy

Input: $\boldsymbol{x}^{\text{prev}}$, $\boldsymbol{I}$, $\boldsymbol{T}_f$ $f \in \{1, 2, ..., F\}$, $\boldsymbol{M}$, α, $\min_i$, $\max_i$
Output: $\boldsymbol{x}$

```
Initialize x = x^prev + [Δx, Δx, ..., Δx]^T where Δx is the result of registering I against the previous frame.
for m = 3 : 0 do
    x ← (1/2)^m x
    ‖DC̄‖_old ← ∞
    for i = 1 : max_i do
        Let a ← −DC̄
        for f = 1 : F do
            y_f ← [a(2f − 1), a(2f)]^T
            b_f ← y_f/|y_f|
        b ← [b_1^T, b_2^T, ..., b_F^T]^T
        c ← 0.5a + 0.5b
        x = x + cd where d is output of line search
        if ‖DC̄‖ > 0.99‖DC̄‖_old and i > min_i then
            Exit for loop early
        else
            Assign ‖DC̄‖_old = ‖DC̄‖
    x ← 2^m x
Return x
```

Efficiency and Complexity

We have found that our algorithm typically converges in about 20 iterations or less at each pyramid level (with fewer iterations on lower pyramid levels). In our experiments, we used a resolution of 640-by-480 (we have also done tests at 1000×562), and we found that 4 pyramid levels were sufficient for reliable tracking. Thus, on average, less than 80 iterations are required to track from one frame to the next. A single iteration requires one gradient evaluation and multiple evaluations of $\bar{C}$. The complexity of a gradient

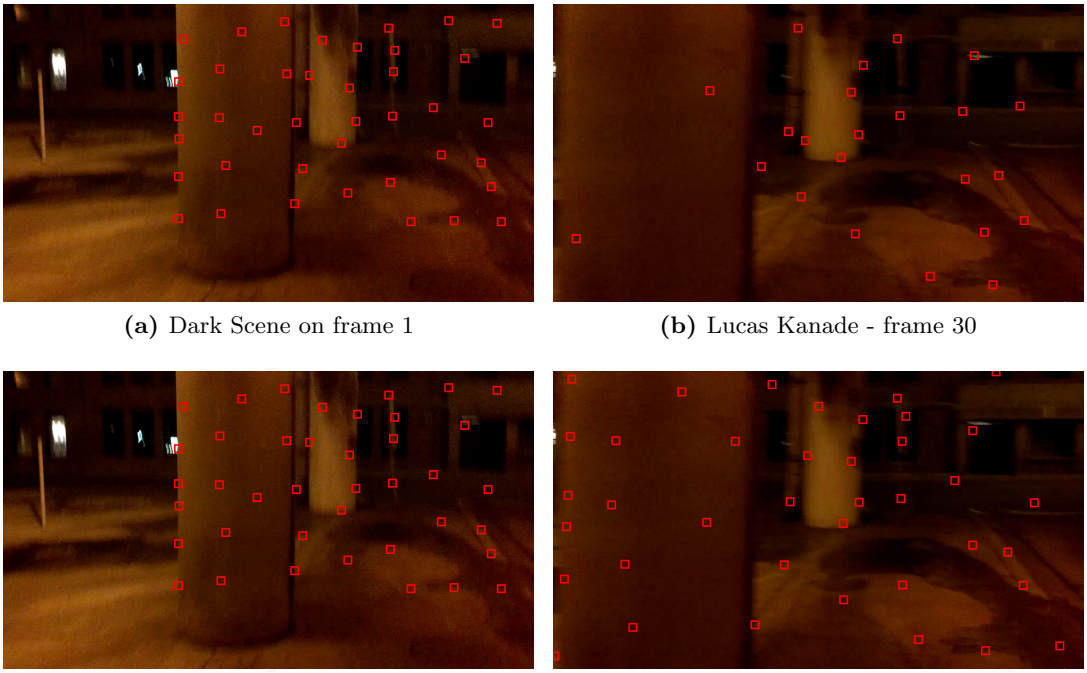
evaluation is $k_1 F n^2 + k_2 L F^2$, and the complexity of an energy evaluation is $k_3 F n^2 + k_4 L^2 F$ (details are given in the supplementary material). Our C++ implementation (which makes use of OpenCV) can run on 35 features of size 7-by-7 with a temporal window of 6 frames ($L = 5$)[2] on a 3rd-generation Intel i5 CPU at approximately 16 frames per second. SIMD instructions are used in places, but no multi-threading was used, so faster processing rates are possible. With a larger window of $L = 10$ our algorithm still runs at 2-5 frames per second.

3.4 Experiments

To evaluate our method, we conducted tests on several real video sequences in circumstances that are difficult for feature tracking. These included shaky footage in low-light environments. The resulting videos contained dark regions with few good features and the unsteady camera motion and poor lighting introduced time-varying motion blur.

In these video sequences it proved very difficult to hand-register features for ground-truth. In order to present a quantitative numerical comparison we also collected higher-quality video sequences and synthetically degraded their quality. We used a standard Lucas-Kanade tracker on the non-degraded videos to generate ground-truth (the output was human-verified and corrected). We therefore present qualitative results on real, low-quality video sequences, as well as quantitative results on a set of synthetically degraded videos.

[2] Accuracy for $L = 5$ is only slightly worse than for $L = 10$ and enables faster processing. See the supp. material for a brief comparison.



(a) Dark Scene on frame 1

(b) Lucas Kanade - frame 30

(c) Dark Scene on frame 1

(d) Our method - frame 30

Figure 3.1: Results of tracking features in real low-light video. Most of the features wander significantly with the Lucas Kanade tracker. Our method provides better results on the low-quality features.

(a) Lucas Kanade after 10 frames

(b) Our method after 10 frames

Figure 3.2: Characteristic results of the OpenCV Lucas-Kanade tracker vs our method in our synthetically degraded video experiment. The correct feature locations (according to the Lucas-Kanade tracker on the non-degraded video) are shown in red. Tracker-computed feature locations are shown in green.

Table 3.1: Mean L1 trajectory error after 30 frames of tracking. Lower is better.

		Video Number								Average
		1	2	3	4	5	6	7	8	
Tracker	KLT	959.6	2484.0	958.3	1242.4	1630.2	1391.4	2105.0	4387.6	1894.8
	1st-Order Descent	**92.5**	137.8	159.5	273.2	87.5	198.4	70.6	685.7	213.2
	LDOF	508.0	408.6	898.5	385.2	104.9	**122.3**	256.1	721.3	425.6
	Multi-Tracker - Emp Dim - Uncentered	104.1	139.3	128.8	241.9	75.2	136.9	58.8	305.5	148.8
	Multi-Tracker - Emp Dim - Centered	102.9	**115.3**	**108.3**	**226.8**	**69.7**	128.6	**54.1**	**292.5**	**137.3**
	Multi-Tracker - Nuc Norm - Uncentered	106.8	134.0	131.7	243.9	73.4	132.6	58.4	293.9	146.8
	Multi-Tracker - Nuc Norm - Centered	103.5	137.7	141.5	243.9	73.4	135.5	60.3	341.0	154.6
	Multi-Tracker - Exp Fact - Uncentered	103.4	169.7	131.3	246.1	74.6	134.3	62.5	307.3	153.7
	Multi-Tracker - Exp Fact - Centered	102.9	167.0	129.1	245.0	73.2	133.4	58.9	302.5	151.5

Table 3.2: Average number of frames between feature re-initializations. Higher is better.

		Video Number								Average
		1	2	3	4	5	6	7	8	
Tracker	KLT	10.6	7.8	13.5	9.6	7.6	9.5	7.8	2.0	8.6
	1st-Order Descent	38.9	30.3	68.7	27.7	34.0	41.1	44.1	3.9	36.1
	LDOF	8.8	12.0	13.7	20.4	25.5	68.4	23.1	6.7	22.3
	Multi-Tracker - Emp Dim - Uncentered	73.4	35.3	111.5	55.6	**70.2**	102.3	63.7	14.0	65.8
	Multi-Tracker - Emp Dim - Centered	74.3	**38.2**	111.5	**58.5**	69.1	104.4	65.6	14.0	67.0
	Multi-Tracker - Nuc Norm - Uncentered	77.2	35.3	108.4	53.8	62.1	105.5	**68.5**	13.6	65.6
	Multi-Tracker - Nuc Norm - Centered	**78.2**	33.8	**114.9**	54.3	66.9	**109.0**	65.6	13.8	**67.1**
	Multi-Tracker - Exp Fact - Uncentered	76.2	34.3	**114.9**	54.3	66.9	98.3	**68.5**	13.6	65.9
	Multi-Tracker - Exp Fact - Centered	74.3	34.8	111.5	53.0	68.0	**109.0**	65.6	**14.3**	66.3

3.4.1 Qualitative Experiments on Real Videos

In our tests on real video sequences containing low-quality features, single-feature tracking does not provide acceptable results. When following a non-distinctive feature, the single-feature energy function often flattens out in one or more directions. A tracker may move in any ambiguous direction without realizing a better or worse match with the features template. This results in the tracked location drifting away from a features true location (i.e. "wandering"). This is not a technical limitation of one particular tracking implementation. Rather, it is a fundamental problem due to the fact that the local imagery in a small neighborhood of a feature does not always contain enough information to deduce the features motion between frames. This claim can be verified by attempting to hand-register low-quality features by only looking at a small neighborhood of the features last known location.

In these situations, our method infers the global motion of the scene from the observable features and uses it to assist in locating the low-quality features. This yields better overall tracking results in hard-to-track videos. Fig. 3.1 shows characteristic results from our tests. Several real videos are included in the supplemental material with results from the OpenCV Lucas Kanade tracker, and from our method.

3.4.2 Experiments on Synthetically Degraded Videos

For this experiment, we collected 8 video sequences of variable length in favorable lighting conditions. We used a Lucas Kanade tracker to track many features and manually verified and corrected the individual trajectories. Features come and go in these sequences (we do not assume all features persist through the entire sequence). These videos include 6 rigid environments as well as one video with multiple rigid bodies (video 7) and one video with a deformable body (video 8). The sequences range in length from 97 frames to 289 frames. On average, they are 210 frames each and contain over 6000 feature-frames each (this is the sum of each tracked features lifespan, measured in frames).

We degraded each video sequence by first darkening and adding noise to each frame, followed by applying a strong Gaussian blur to each frame. After this we added additional Gaussian noise. Adding noise before and after blurring gave the effect of noise

at different scales (harder to deal with than per-pixel noise only). The test videos are included in the supplementary material.

For our comparison, we ran each tracker in two different modes. In the first mode we initialized each feature with its ground-truth location and re-initialized features when they wandered more than 10 pixels from ground truth. We recorded the average number of frames between feature re-initializations. In the second mode, we only tracked the features that were visible in frame 0, and features were never re-initialized. We looked at the mean L_1 difference between the output trajectories and ground truth after 30 frames.

As a reference, we compared against the pyramidal Lucas Kanade tracker in the current OpenCV release (2.4.3). For a more recent comparison, we used LDOF (Large Displacement Optical Flow [50]) to generate dense flow fields for each sequence and we interpolated these flow fields to generate long-run trajectories for features. We also implemented our own single-feature gradient descent tracker (with an absolute value loss function). We present results for our rank-constrained tracker with the three previously introduced penalty functions. For each penalty function we present results with and without centering the history matrix $\boldsymbol{M}$. In this experiment, whenever our algorithm is run with the penalty term, we use a weak constraint. All trackers were run on grayscale video. The results of this experiment are presented in Tables 3.1 and 3.2. An additional set of tests (on shorter video sequences) is included in the supplementary material.

3.4.3 Analysis of Results

From Tables 3.1 and 3.2, we can see that imposing our weak rank constraint significantly improves overall tracking ability, with all three rank regularizers that we tested showing improved tracking performance. Comparing the Lucas Kanade results to the results of our single-feature gradient descent tracker, we see a very large gap in performance. The core differences between these two algorithms are the definitions of ψ (squared error vs. absolute value) and the method of optimization employed. This performance difference supports our previous claim that the 2^{nd}-order optimization technique used to accelerate convergence in the Lucas Kanade algorithm can be unreliable when tracking poor-quality features.

3.5 Conclusion

Rank constraints have been successfully applied to several problems in computer vision, including motion segmentation and optical flow estimation. We have expanded on this previous work by developing a feature tracking framework which allows these constraints to be reliably used to assist in the tracking of features in rigid environments as well as in more general, non-rigid settings. The framework we presented permits these constraints to be imposed forcefully, allowing one to track features on a rigid object even if some features are occluded, or weakly, where the constraints are only used to help locate poor-quality features that cannot be tracked on their own. We showed that the weak constraint can yield significant gains in tracking performance, even in non-rigid scenes (with multiple or deformable objects) The framework we presented is completely causal and does not require explicitly modeling structure or motion in a scene. Furthermore, the algorithm we proposed is not prohibitively computationally expensive (real-time performance has been achieved). Our results provide evidence that when tracking features in low-quality video (especially in a rigid or semi-rigid scene), a 1^{st}-order descent scheme is more robust than 2^{nd} order methods used in standard Lucas-Kanade trackers, and applying rank regularizers to track a cohort of features results in better performance than classical single-feature tracking.

We conclude this section by mentioning that a recent work, [51], builds on our approach in this part and uses the same concepts to create a rank-constrained, multi-body feature tracker that uses the self-expressiveness property of hybrid-linear distributions to allegedly achieve better performance in multi-body scenes, still without requiring explicit scene segmentation.

Chapter 4

Part III: Enhancing Feature Tracking With Gyro Regularization

Based on work done with Gilad Lerman

4.1 Introduction

Feature tracking is the task of determining and maintaining the location of one or more visually interesting points as they move about in motion video. This task is crucial in computer vision, where it is frequently used as a first step in solutions to important problems like simultaneous localization and mapping (SLAM) and structure from motion (SFM). A common solution is to characterize each feature using a *template* image, which is a small image centered on the feature, extracted from a recent frame. This template is updated periodically, and the feature's location in each new frame is determined by searching through the new imagery for the region that is most common to the template. The celebrated Kanade-Lucas-Tomasi (KLT) feature tracker [26, 27, 28, 1] achieves this, for instance, using Gauss-Newton optimization to find the location in a new image that minimizes the mean-squared difference between the image and the template.

For a given camera, the location of a feature in the image plane is a function of the location of the corresponding 3D world point relative to the coordinate frame of the camera. If we imagine a "world frame" fixed to the environment, then the motion of a feature in the image plane (which we will refer to as a feature's *flow*) can be decomposed into two quantities: motion of the corresponding 3D point relative to the world frame, and movement of the camera frame relative to the world frame. This decomposition is useful because motion of world points is often much slower when measured in the coordinate frame of their environment than when measured in the coordinate frame of the camera. If the motion of the camera relative to the environment can be measured independently, then the component of a feature's flow due to camera egomotion can be predicted a-priori, leaving only the smaller component due to motion through the environment to be determined. This is especially helpful in hand-held camera applications, where camera rotation can completely dominate other sources of flow.

Independently measuring camera motion can be quite challenging. Position and orientation can often be determined by measuring external signals like GNSS (Global Navigation Satellite Signals). Such external signals are not always detectable or reliable however, and exploiting them can require additional undesirable hardware (like GNSS antennas). Additionally, for the purpose of predicting a feature's flow we actually only need to measure relative motion (from one frame to the next), instead of knowing our pose relative to an absolute reference frame. In this setting, inertial sensors (which measure accelerations due to specific forces and rotation rates) are a desirable alternative because they can offer excellent relative motion measurements over short time intervals without relying on external signals. Additionally, these sensors have become very low cost in recent years and have made their way into all kinds of consumer electronics (like smart phones and tablet computers). It is now the case that many devices one might use to collect video already happen to have inertial sensors built into them. This makes them a natural choice for integration with feature trackers.

Inertial sensors can be used to estimate relative changes in camera orientation and/or position between two frames and predict the component of the optical flow field due to camera egomotion. The simplest way to exploit this information is to use the predicted flow field to initialize and bound the search for features in the new frame. This has been found to be a successful strategy by several authors (see § 4.1.1). While this strategy

can reduce the number of candidate locations for a feature in a new frame of video, it still relies entirely on the imagery for selecting the best candidate. We propose an additional level of integration where we regularize the tracking energy function to gently penalize deviation from our prior estimate of flow. Thus, in addition to reducing the number of candidate locations for a feature, our method can help differentiate between them when the imagery is not distinctive enough to reveal the location on its own.

The rest of the paper is organized as follows. We review previous work in §4.1.1 and detail the contributions of this work in §4.1.2. Template-based feature tracking is reviewed in §4.2. In §4.3 we summarize how a prior estimate of optical flow can be computed using a gyroscope. In §4.4 we introduce our method of exploiting this prior flow estimate for feature tracking. In §4.5 we present results which demonstrate that our technique offers significant improvements in performance over conventional template-based tracking methods, and is in fact competitive with much more computationally expensive state-of-the-art trackers, but at a fraction of the cost. §4.6 explores possible areas for future research. Finally, §4.7 concludes this work.

4.1.1 Review of Previous Work

Gyroscopes have been used in conjunction with optical systems for multiple purposes in recent years. Perhaps the most ubiquitous application is image stabilization. This broadly refers to two different problems. The first is to reduce blur in individual images during long-exposure acquisitions (e.g., low-light photography). This can be done by detecting camera rotation during the acquisition period and driving actuators connected to the imager to mechanically compensate for the motion [52] (this is called optical image stabilization, or OIS), or it can be done by using the detected camera motion to deblur the image in post-processing (as in [53]). The second problem is to reduce the shakiness of video by measuring rotation during video acquisition and translating frames in the sequence to compensate for short, rapid motions, resulting in video that only reflects the smoother camera motion [54].

Gyro-optical integration has also been used to improve feature tracking for computer vision applications. You et al. [55] used gyroscopes to predict feature motion in the image plane of a camera and restrict the search space for feature tracking. Feature trajectories were then used in turn to yield driftless attitude estimates for use in

augmented reality systems. Hwangbo et al. [56] used gyroscopes to estimate relative changes in orientation to predict feature motion for initializing a KLT feature tracker (and to pre-warp templates), in order to provide more robust feature tracking for vision applications such as 3D reconstruction.

Perhaps the most studied application of integrated optical and inertial sensors has been in aiding inertial navigation systems when conventional sources of navigation information (like GNSS) are unavailable or untrustworthy. Pachter and Mutlu [57] proposed tracking and geo-localizing unknown terrain features and using them as landmarks for self-localization when conventional aides are unavailable. Mourikis et al. [58] and Roumeliotis et al. [59] exploit feature trackers to generate measurements to aid inertial navigation systems for space travel and space vehicle descent. Jones and Soatto [60] use trajectories from a KLT feature tracker with an inertial navigation system in a modern framework for robust SLAM. While these methods employ loose integration between optical and inertial systems, other authors have proposed tighter integration (where the inertial system plays an important role in feature tracking). Hol et al. [61] used measured changes in both position and orientation to predict the motion of features in the image plane and bound the search space for those features. They located features by minimizing the sum-of-absolute-differences between imagery in the search region and feature template images. They used trajectories in turn to provide corrections to the navigation system via a Kalman filter. Veth and Raquet [62, 63] implemented a similar optical-inertial system, while matching SIFT descriptors to locate features in the search region. Gray [64] extended this work to achieve deeper integration between the inertial and optical systems by predicting perturbations in feature descriptors that result from changes in system pose and directly compensating feature descriptors to achieve better matching performance. Predicting feature motion due to sensor translation is considerably more challenging than predicting motion due to rotation, as it requires estimating the range from the camera center to each tracked feature. To deeply integrate feature tracking with a full inertial navigation system one must employ some method of estimating feature range ([61] assumes a 3D scene model is known, [62] and [63] use a binocular camera, and [64] presents results using binocular systems and monocular systems with range estimated online from motion). An alternative is to neglect the effect of camera translation on feature motion (when rotation is dominant). Diel et al. [65] used

gyroscopes to measure relative changes in camera orientation and pre-warped imagery to compensate, thereby making feature tracking simpler. They used tracked features to derive epipolar constraints and applied them to an inertial navigation system through a Kalman filter.

4.1.2 Original Contributions of This Work

Our main contribution is a novel method for deeply integrating inertial sensors with template-based feature tracking. We directly modify the tracking energy function to exploit inertial measurements as a prior estimate of feature position. This modification aids the tracker in localizing features in directions where the imagery is ambiguous (directions where the unmodified energy function is relatively flat). Unlike our proposed method, previous works in integrating inertial sensors with feature trackers generally achieve lower levels of sensor integration. Many methods are very loosely integrated, where feature tracking is nearly independent of the inertial system [58, 59] and realizes no benefit from these other sensors. Other methods exploit the inertial sensors (and sometimes additional sources of navigation information) to initialize and bound the search for features in the image plane [64, 61, 56, 63, 62, 55]. Some techniques also use this information to pre-warp feature template images [56], or to correct feature descriptors to better match new imagery [64]. However, to our knowledge, ours is the first method to directly regularize the tracking energy function using gyro-predicted optical flow.

Our proposed solution competes with state-of-the-art feature trackers in performance, but has only a fraction of the computational cost of some competing methods. In addition, we show that using gyro-predicted optical flow to merely initialize a feature tracker (like KLT) offers no advantage over a careful optical-only initialization method (see (4.17)). This suggests that in some of the advances reported by other authors the gyroscopes were not really needed to achieve the gains from better initialization. A deeper level of integration, like the one proposed here, may be necessary to realize a genuine tracking performance improvement from these inertial sensors.

4.2 A Review of Template-Based Feature Tracking

The goal of feature tracking is to determine the location of a small point or feature as it moves about in motion video. This work focuses on template-based feature tracking, where a feature is characterized by a small (usually square) *template image* that describes the expected appearance of the feature. This fundamentally distinguishes feature tracking from *object tracking*, where the target is potentially larger and can change significantly in appearance between frames, requiring more sophisticated techniques for target characterization. In template-based feature tracking, the goal is usually to locate a feature by minimizing the *single-feature energy function*:

$$c(\boldsymbol{x}) = \frac{1}{n^2} \sum_{\boldsymbol{u} \in \Omega} \psi \left(T(\boldsymbol{u}) - I(\boldsymbol{u} + \boldsymbol{x}) \right), \tag{4.1}$$

where:

T = Template Image characterizing the feature

n = Width and height of template image

I = Frame of video in which we are trying to locate the feature

ψ = Loss function. Typically $\psi(\boldsymbol{y}) = |\boldsymbol{y}|$ or $\psi(\boldsymbol{y}) = |\boldsymbol{y}|^2$

$\boldsymbol{x}$ = The candidate location of the feature in the new frame

Ω = The set of locations where T is defined: $\{(i, j)$ where $i, j \in \{1, 2, ..., n\}\}$

Intuitively, we are overlying our template image T on the video frame I in a location governed by the input $\boldsymbol{x}$. Then, for each pixel in the template, we compute the difference in intensity between T and I and take the square or absolute value of the result. For each pixel in the template, this produces a measure of dissimilarity between the template and the image which is 0 when they are equal and positive otherwise. Our energy is the average of these values over all pixels in the template. Thus, when we minimize (4.1), we are effectively sliding our template around over the video frame, looking for the location where the image looks most similar to the template.

Generally, there is a maximum distance which a feature is expected to move between two consecutive frames. Thus, the energy (4.1) is minimized in a local neighborhood of its previous position. The minimization can be performed through exhaustive search, using 1^{st}-order methods like gradient descent, or using higher-order schemes like Gauss-Newton minimization (as in the KLT tracker). Additionally, most feature trackers are implemented in a coarse-to-fine framework, where tracking is done in stages: first on lower-resolution imagery and then subsequently refined on higher-resolution imagery [66]. This is to increase the likelihood that in any given stage the initial estimate of a feature's position is within the region of convergence of the true minimum of (4.1).

As an alternative to tracking each feature independently, some modern feature trackers have as their state variable the joint positions of a collection of many (or all) features in the scene. Hence, they iteratively refine the position estimates of each feature simultaneously, as opposed to one after another. This allows them to impose constraints on how features can move relative to one other. An example of this is [67], where features are encouraged, through a penalty term, to maintain trajectories over a short temporal window which lie in a low-dimensional subspace. We refer to such tracking methods as *multi-feature trackers* or in short *multi-trackers*.

In the multi-feature tracking framework the state variable encodes the joint positions of a collection of many (or all) features in the scene. The main energy function for multi-feature tracking, which we call the *multi-feature energy function*, is formed by simply combining all of the single-feature energy functions of a collection of F tracked features:

$$\frac{1}{n^2}\sum_{f=1}^{F}\sum_{\boldsymbol{u}\in\Omega}\psi\left(T_f(\boldsymbol{u})-I(\boldsymbol{u}+\mathbf{S}_f\boldsymbol{x})\right),\ \text{where}\ \mathbf{S}_f=\left(\begin{array}{ccccccccc} 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \end{array}\right). \tag{4.2}$$

The state variable $\boldsymbol{x}$ is a $2F$-element vector, where elements $(2f-1)$ and $2f$ form the position of feature f. The matrix $\mathbf{S}_f$ extracts these coordinates from $\boldsymbol{x}$. The inner sum in (4.2) is nothing more than the standard single-feature energy function (4.1) for feature f. The outer sum adds up these energy functions for each feature being tracked. Notice though that each single-feature energy function depends on a different pair of coordinates in the state vector, $\boldsymbol{x}$. Thus, minimizing (4.2) is equivalent to minimizing each single-feature energy function separately. It only becomes different if

we impose constraints or add additional terms that introduce interactions between the locations of different features. As with single-feature trackers, there are many ways this energy can be minimized, and it is often implemented in a multi-resolution, coarse-to-fine framework.

It is important to understand the limitations inherent in both single-feature and multi-feature tracking. In single-feature tracking, unless one pulls in additional sources of information, it is not generally possible to track arbitrary features. For instance, if you are trying to track a feature in a completely textureless, uniform portion of an image, then it can be readily verified that the energy function (4.1) becomes completely flat. When this happens, there is no unique minimizer and the flow of the feature is not observable in the imagery. A less extreme example would be tracking a feature along an edge in an image. In this case, sliding the template along the edge does not change the energy, but sliding the template transverse to the edge results in an energy increase. In this situation it is theoretically possible to partially resolve the flow of the feature (i.e. in the direction orthogonal to the edge). However, template-based trackers are designed to try to fully resolve the location of the feature and they will select what they see as the best minimum, even when there are several, nearly identical candidates. The result is that such features can wander and jump around along the edges to which they belong. Without additional sources of data or additional assumptions about how features interact, the multi-feature tracking framework suffers from the same problem because it essentially minimizes the single-feature energy functions for a set of features.

One straightforward way to address this issue is to simply not attempt to track features which are less than ideal. For instance, [27] defines a criterion for a template that ensures that the corresponding feature is distinctive enough to make tracking possible and suggests discarding features that don't meet the criterion. Many other criteria have been proposed in the literature, but all effectively try to select corner-like features and have the shared goal of screening out features that are likely to cause problems in tracking. This is a reasonable approach to the problem but it is not universally appropriate. For instance, when using trajectories for the purpose of 3D reconstruction, one wants long feature trajectories that survive long enough to allow individual features to be viewed from many different poses. Replacing features when they become

difficult to track has the contrary effect of building a larger set of shorter-lived feature trajectories. Navigation applications also tend to prefer longer feature trajectories. Additionally, there are phenomena that can negatively impact many or all features in a scene together, such as poor lighting conditions. The ability to track through even short-duration events like this can be quite valuable in these applications.

When trying to track a collection of less-than-ideal features the flow of each feature may not be independently observable in the imagery. Thus, some additional assumptions or sources of information must be brought in to make this possible. In [67] a low rank constraint is used as a way of sharing information between features to make low-quality features more trackable. In this work, we bring in an outside estimate of the flow of each feature, in particular, the (often dominant) component due to camera rotation. We rely on this to help locate features in directions where the imagery is not distinctive enough for a features location to be discerned from the imagery alone. Before we discuss how we use this outside estimate, we develop the equations used to produce the estimate using gyroscopes rigidly mounted to our camera.

4.3 Using Gyroscopes to Predict Optical Flow

In our application we estimate the component of optical flow that is due to changes in camera orientation between frames to use as our prior estimate of flow in feature tracking. In many situations this is the dominant source of flow simply because cameras are often free to rotate much faster than they can move through their environments (or relative to other visible objects). For the purpose of illustration, consider a common cellular phone camera with 50° diagonal angle of view. In hand-held applications it is not uncommon for the camera to achieve rotation rates (in part due to unintentional camera "shaking") of 350°/s. If images are being collected at 30 fps, this camera rotation induces motion in the image plane of up to 23% of the image diameter between consecutive frames[1] . For a non-rotating camera viewing an object 50 feet away, in order to generate a comparable flow in the image plane the object would need to be moving at least 199 mph (320 km/h). In many situations objects are not moving nearly

[1] The magnitude of the flow in pixels depends on the sensor resolution. If the sensor has a resolution of 600x800 (1000 pixels across the diagonal), this would correspond to a maximum displacement of 230 pixels.

this fast, and camera rotation is therefore the dominant source of flow.

A typical camera can be modeled using the perspective model [2]. With this model, there is a *calibration matrix* $\mathbf{K}$ associated with the camera, depending only on the internal characteristics of the image sensor and optics, such that the image of a 3D point with coordinates $\boldsymbol{X}$ in the coordinate frame of the camera is $\boldsymbol{x} = \mathbf{K}[\boldsymbol{I}, \mathbf{0}]\boldsymbol{X}$. Both $\boldsymbol{X}$ and $\boldsymbol{x}$ are expressed in homogeneous coordinates. The matrix $\mathbf{K}$ has the following form:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & a_0 \\ 0 & f_y & b_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.3}$$

where f_x, f_y, a_0, and b_0 are constants associated with the imaging sensor and optics. This model is relatively general since most camera non-idealities (e.g., radial distortion) can be compensated for, when necessary, explicitly in pre-processing.

Next, we must identify what information we are able to extract from the gyroscopes. Ideally, they would allow us to measure the rotation of the camera between arbitrary instants in time. Unfortunately, however, gyro measurements are taken in the coordinate frame of the gyro sensor package, which is not generally aligned with the camera frame. We will assume that there is some fixed rotation matrix that takes vectors in the gyro sensors coordinate frame to the cameras internal coordinate frame (this assumption requires that the gyro be physically mounted to the camera so that they experience the same rotations).

We will be considering the camera at two instants in time corresponding to acquisition times of two consecutive frames of video. We will refer to these instants as time 0 and time 1, respectively. We will let $\boldsymbol{X}^0$ represent the position of a feature at time 0 (in homogeneous coordinates), in a coordinate frame which we will refer to as the *gyro frame*. This frame has its origin at the camera center and has its axes parallel to those of the coordinate frame of the gyro sensor package. Similarly, we will let $\boldsymbol{X}^1$ represent the position of the feature at time 1 in the gyro frame. Let $\mathbf{R}_{\text{Gyro}}^{\text{Cam}}$ be the fixed rotation matrix that takes vectors in the gyro frame to the cameras internal coordinate frame (these coordinate frames share the same origin in space so there is no translational component to this transformation). Then, the images of a feature in the focal plane at

times 0 and 1 are given by:

$$\boldsymbol{x}^0 = \mathbf{K}\,[\boldsymbol{I}, \mathbf{0}] \begin{bmatrix} \mathbf{R}_{\text{Gyro}}^{\text{Cam}} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \boldsymbol{X}^0, \quad \text{and} \quad \boldsymbol{x}^1 = \mathbf{K}\,[\boldsymbol{I}, \mathbf{0}] \begin{bmatrix} \mathbf{R}_{\text{Gyro}}^{\text{Cam}} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \boldsymbol{X}^1. \tag{4.4}$$

We can rewrite these equations as:

$$\boldsymbol{x}^0 = \mathbf{K}\mathbf{R}_{\text{Gyro}}^{\text{Cam}}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^0, \quad \text{and} \quad \boldsymbol{x}^1 = \mathbf{K}\mathbf{R}_{\text{Gyro}}^{\text{Cam}}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^1. \tag{4.5}$$

If we define $\tilde{\mathbf{K}} = \mathbf{K}\mathbf{R}_{\text{Gyro}}^{\text{Cam}}$, we get:

$$\boldsymbol{x}^0 = \tilde{\mathbf{K}}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^0, \quad \text{and} \quad \boldsymbol{x}^1 = \tilde{\mathbf{K}}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^1. \tag{4.6}$$

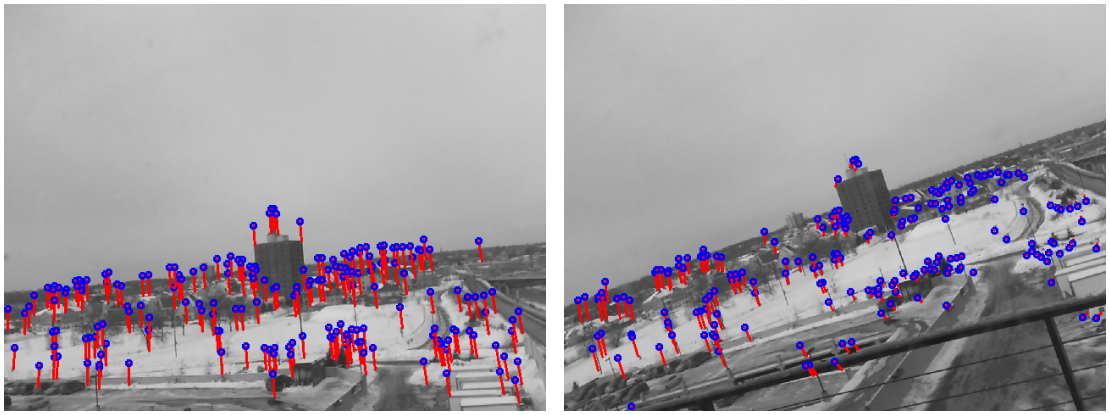
Next, we will collect measurements from our 3-axis gyroscope between time 0 and time 1. What comes out of a gyro are measurements of the rotation rates of the gyro frame about its X, Y, and Z axes relative to inertial space, coordinatized in the gyro frame. We will denote these rates r_x, r_y, and r_z, respectively. From these measurements we want to compute the 3x3 matrix describing the rotation of the gyro frame between times 0 and 1, which we will call $\mathbf{R}$. This is a well-studied problem, and we use a common quaternion representation for rotations to help solve it. It can be shown (e.g., [68]) that the quaternion representing the orientation of the gyro frame obeys the ODE: $\dot{Q} = \left(\frac{1}{2}\right) QP$, where $P = 0 + r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k}$. We initialize our quaternion Q at time 0 as $Q = 1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$, which represents the trivial rotation (by angle 0), and we integrate this ODE numerically from time 0 to time 1 and convert the final quaternion to a rotation matrix, $\mathbf{R}$, as described in [68].

Since $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$ are defined relative to the gyro frame, they are related by:

$$\boldsymbol{X}^1 = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \boldsymbol{X}^0. \tag{4.7}$$

Combining this with (4.6), we can write:

$$\boldsymbol{x}^1 = \tilde{\mathbf{K}}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^1 = \tilde{\mathbf{K}}\,[\boldsymbol{I}, \mathbf{0}] \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \boldsymbol{X}^0 = \tilde{\mathbf{K}}\,[\mathbf{R}, \mathbf{0}]\,\boldsymbol{X}^0 = \tilde{\mathbf{K}}\mathbf{R}\,[\boldsymbol{I}, \mathbf{0}]\,\boldsymbol{X}^0. \tag{4.8}$$



(a) Rotation mostly about cameras horizontal axis

(b) Rotation mostly about cameras optical axis

Figure 4.1: Examples of optical flow prediction using a 3-axis gyroscope. Circles (blue) indicate feature locations in the previous frame; lines (red) connect them to the predicted locations of the same points in the current frame.

Next, observe that $\tilde{\mathbf{K}}$ is invertible because it is the product of a rotation matrix and an invertible camera calibration matrix. We will insert $\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{K}} = \boldsymbol{I}$ in the right spot on the right hand side of (4.8) to get:

$$\boldsymbol{x}^1 = \left(\tilde{\mathbf{K}}\mathbf{R}\tilde{\mathbf{K}}^{-1}\right)\tilde{\mathbf{K}}\left[\boldsymbol{I}, \mathbf{0}\right]\boldsymbol{X}^0 = \tilde{\mathbf{K}}\mathbf{R}\tilde{\mathbf{K}}^{-1}\boldsymbol{x}^0. \tag{4.9}$$

Hence, the images of the feature at times 0 and 1 in the image plane are related by:

$$\boldsymbol{x}^1 = \mathbf{H}\boldsymbol{x}^0, \quad \text{where} \quad \mathbf{H} = \tilde{\mathbf{K}}\mathbf{R}\tilde{\mathbf{K}}^{-1}. \tag{4.10}$$

Thus, if we are able to estimate the rotation of the gyro frame between two consecutive frames, $\mathbf{R}$, and the constant 3x3 matrix $\tilde{\mathbf{K}}$ is known, then we can compute the matrix $\mathbf{H}$ (which is a homography between the two frames) and predict where each point observed in the first frame will appear in the second. Figure 4.1 illustrates the output of this process. It is important to observe that the range to a feature is immaterial in this development. Range only effects the component of flow due to a points translational motion relative to the camera from time 0 to time 1, which is assumed to be negligible in this development.

The constant matrix $\tilde{\mathbf{K}}$ can be determined by collecting a video sequence of a static scene where the camera undergoes rotations about all 3 axes (but only negligible translations). Between each pair of consecutive frames the rotation $\mathbf{R}$ is computed from the gyroscope and $\mathbf{H}$ is estimated via optical image registration. Each consecutive pair of frames yields a set of 9 equations from (4.10), where the unknowns are the elements of $\tilde{\mathbf{K}}$. The system can be solved in a least-squares sense to yield $\tilde{\mathbf{K}}$.

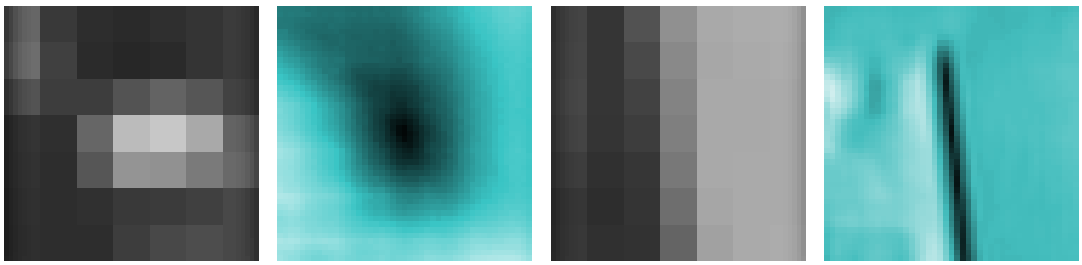
It should also be mentioned that using gyroscopes to predict optical flow requires some amount of calibration and integration between the gyros and the camera. For instance, the data from the gyros must be synchronized with the camera data. Also, gyros have slowly varying biases that must be compensated for. See 4.8.1 for a discussion of these details.

4.4 Exploiting a Prior Estimate of Flow

The idea behind our proposed method is to regularize the registration energy function by adding a term that penalizes feature positions that deviate from our prior estimate of flow (in our case derived from a gyroscope). The term we add will be small enough to not interfere with the well-behaved energy functions of strong features, but it will be significant enough to dominate the energy functions of weak features in ambiguous directions. This allows us to extract all of the information that is present in the imagery[2] , but rely on our prior estimate of position when localizing in ambiguous directions. That is, we want the ability to "ride" our gyro measurements through poor imagery, until a feature can be re-acquired, but our implementation should only effect weak features and only in directions where the feature cannot be localized based purely on the imagery.

When tracking corner-like features, the single-feature energy surface (4.1) is generally well-behaved in a small neighborhood of the global minimum. If we start minimization with an initial guess that is sufficiently close to the minimum we generally have

[2] For instance, we may be able to determine the horizontal position of a feature on a vertical edge very precisely from the imagery, but we may be unable to determine the same features vertical position along the edge. In this case there is still valuable information in the imagery that we do not want to ignore.



(a) Corner-like feature template **(b)** Energy surface for (a) **(c)** Edge-like feature template **(d)** Energy surface for (c)

Figure 4.2: Examples of a corner-like and edge-like feature template images from a real-world video sequence, along with their corresponding energies (see (4.1)). The energy for the edge-like feature does not have a unique minimum. The energy images are colorized to help distinguish them from the template images. Black is low energy, blue is medium energy, and white is high energy.

no problems converging to it. However, when tracking features that are not corner-like, the energy surface is often not so well-behaved. In fact there may not even be a unique global minimum. In this situation, it is not possible to completely identify the correct location of a feature using template registration alone (this has nothing to do with the optimization scheme employed; it is a problem with the energy function itself). In Figure 4.2 we give sample template images and energy functions for corner-like and edge-like features. Notice that there is a line of global minimums for the energy function corresponding to the edge-like feature (we will call this a line of ambiguity). This is because sliding the template image along the edge does not result in a better or worse match between the image and the template. Of course, because the energy function is derived from real-world data there will be a single global minimum somewhere on the line of ambiguity, but the location of the minimizer on this line will be unstable and unpredictable. This is why edge-like features tend to wander (and sometime jump) along their lines of ambiguity during tracking. It is important to note that simply initializing (or even bounding) the search of such a feature using the prior estimate of flow (as in [56]) will have little effect here since the issue is the instability of the minimizer of the energy function, not the inadequacy of a given optimization scheme at finding it.

In this work, we combine information about the flow of a feature from two very different sources. On one hand, we have imagery, which promises to yield an extremely

accurate estimate of flow, provided the imagery is distinctive enough to reliably align with the feature's template. On the other hand, we have a flow estimate based on measurements of camera rotation. This estimate is more crude because it does not reflect flow due to translational motion of the feature relative to the camera, but it is not effected by ambiguity in the imagery. In summary, we have one source of information (the imagery) that can accurately capture all sources of flow (both due to camera rotation and translational motion of the feature), but which is susceptible to ambiguity in the imagery. Then we have another source of information (gyroscopes) that can only observe the often-dominant portion of flow, but which is not susceptible to ambiguity in the imagery. These properties make the two sources complimentary for the purposes of tracking.

When bringing together information from complimentary sources such as these, there is a temptation to produce estimates of the flow from both sources of information and combine them in a filter, exploiting known error characteristics of both estimates. This approach is impractical in this situation, however, because the error of the gyro-derived flow estimate is difficult, if not impossible to accurately characterize because it is blind to translational motion of features. Depending on the subject matter of the video, it may not even be reasonable to assume that the gyro-derived flow estimate is unbiased (imagine video of traffic, where most features are moving in one direction, in no part due to camera motion). What can surely be said is this: When the imagery is distinctive enough to localize a feature in a given direction, that estimate should be preferred over any other estimate. When the imagery is not distinctive enough to localize a feature in some direction, then it makes sense to fall back on the gyro-based estimate of flow. This is precisely what is accomplished by our proposed method. By weakly regularizing the image-based registration energy function using the gyro-derived estimate of flow, we use the imagery as our primary instrument for localizing a feature. However, when the imagery is not distinct enough to localize a feature in some direction, our gyro-based estimate takes over to fill in the missing information.

4.4.1 Single-Feature Tracker Implementation

In §4.3 we covered how we can take a features location in one frame and predict its location in the next using gyro measurements. We will let $\boldsymbol{x}_{\text{gyro}}$ denote this predicted

position for a feature (2D position, in units of pixels). To achieve our goal in a single-feature tracking framework, we will add a term to (4.1) to penalize locations that differ from this prior prediction. The regularization term we add should be small enough to be completely dominated by a strong registration energy function. This must still be the case if the minimum of the registration energy function is relatively far from the prior estimate because there may be valid reasons for a significant discrepancy between true flow and predicted flow. In particular, since our prior is estimated using a gyroscope and only accounts for camera rotation, features on objects that are moving relative to the environment can have a significant component of flow that will not be reflected in the prior estimate. Thus, our regularization term must not grow too quickly as we move away from the prior estimate. It is also desirable, of course, for the regularization function to not contain local extrema. The penalty function we use is:

$$P(\boldsymbol{x}) = \lambda \ \ln(\alpha|\boldsymbol{x} - \boldsymbol{x}_{\text{gyro}}| + 1)/\ln(\alpha x_{\max} + 1), \tag{4.11}$$

where $x_{\max}$ is a constant (in units of pixels) reflecting the greatest expected deviation from the gyro prediction, the constant α controls the "pointiness" of the curve, and the constant λ controls the overall strength of the penalty (see Figure 4.3). In this entire work, we keep α fixed at 0.5 and $x_{\max}$ fixed at 25 pixels. We discuss in §4.5 how the parameter λ can be learned.

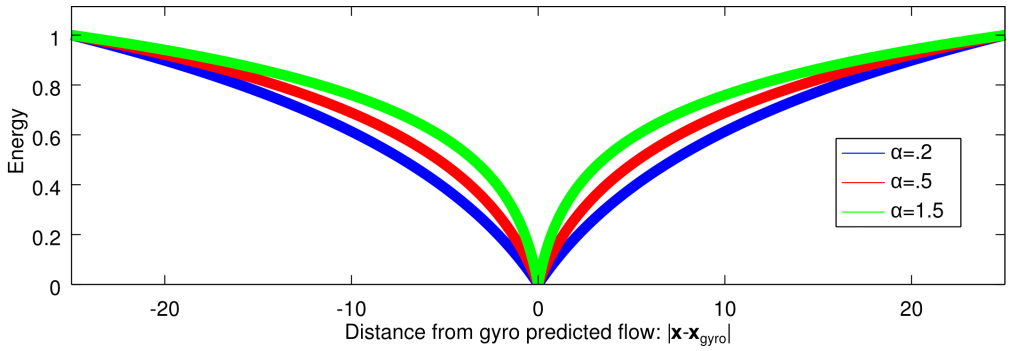


Figure 4.3: The penalty (4.11) with $x_{\max} = 25$ pixels, $\lambda = 1$ and $\alpha = 0.2, 0.5, 1.5$. It levels off not far from the gyro prediction and thus allows a strong registration energy function to dominate, even if the global min is far from the gyro prediction.

After incorporating the above regularization term, we see that to track a feature

when we have a prior estimate of flow we need to minimize the energy function:

$$c_{\text{single}}(\boldsymbol{x}) = \frac{1}{n^2} \sum_{\boldsymbol{u} \in \Omega} \psi \left(T(\boldsymbol{u}) - I(\boldsymbol{u} + \boldsymbol{x}) \right) + \lambda \, \frac{\ln(\alpha |\boldsymbol{x} - \boldsymbol{x}_{\text{gyro}}| + 1)}{\ln(\alpha x_{\max} + 1)}. \tag{4.12}$$

In this work, we use the absolute-value loss function for ψ. Other variables, such as n, Ω, T, and I have the same meanings as in §4.2. In the above energy function, the sum on the left measures the dissimilarity between the template image for the feature and the equally-sized region of the video frame, centered about the point $\boldsymbol{x}$. The term on the right measures the dissimilarity between the point $\boldsymbol{x}$ and the gyro-derived predicted location for the feature. The value of λ will be small, so as to put greater emphasis on the objective of matching the template with the energy. We caution against trying to impose a strict statistical interpretation on the combination of terms in (4.12). The sum on the left and the term on the right are not estimates of the location of the feature. Instead, they are energy functions which are, under suitable assumptions, minimized by the location of the feature. While our term on the right is derived from an estimate of the features location, little can be said about the error characteristics of that estimate. In general, as a result of translational motion of world points relative to the camera, the estimate may not even be unbiased. We suggest viewing (4.12) as a modification of the classical feature tracking energy function, where we add some additional shape to the energy function using a prior estimate of flow. This addition is very slight, and is only consequential when the classical energy function has an ambiguous minimizer (i.e. there are multiple candidate locations in the frame that appear to match the template).

As was the case when tracking without a prior flow estimate, this energy function can be minimized in a number of ways. In our implementation we use gradient descent with fast line search in a coarse-to-fine framework. This is much faster than exhaustive search, while still providing reliable, consistent behavior, even with poorly conditioned energy functions. This is an iterative algorithm where in each step, the gradient of the energy function, ∇E, at the current location is computed and $-\nabla E$ is taken as the search direction. Then, the energy is approximately minimized on a line segment starting at the current location and continuing in the search direction. These steps are repeated for a minimum number of steps and until a maximum number of steps is reached or a stop condition is met. In this work we have two stop conditions: (a) The gradient of

the energy has sufficiently small magnitude (0.00001), or (b) the gradient magnitude is not sufficiently smaller than in the previous iteration (greater than 0.9999 times the magnitude from the previous iteration). This algorithm is detailed in Algorithm 3. The gradient of the energy function is differentiated numerically using the centered derivative approximation (we perturb the position of the feature by 0.25 pixels in each direction). We initialize the tracker on a given feature to the gyro-predicted location for that feature:

$$\boldsymbol{x} = \boldsymbol{x}_{\text{gyro}}. \tag{4.13}$$

The source code for this tracker will be available on our supplemental web page.

Algorithm 3 Gradient Descent With Fast Line Search

Input: $E : \mathbb{R}^D \to \mathbb{R}$: Energy function. $\mathbf{p}_0 \in \mathbb{R}^D$: Initial guess of the minimizer of E ((4.13) in our method, or (4.17) for optical-only methods). minSteps (default = 40 for lowest resolution level, 3 for all other levels), maxSteps (default = 40), stepSize (default = 2.0 pixels), maxRefinements (default = 10).

Output: Minimizer of energy function (at least a local minimizer).

$\mathbf{p} \leftarrow \mathbf{p}_0$.

for $n = 1 :$ maxSteps **do**

$\boldsymbol{v} \leftarrow -\nabla E(\mathbf{p})$ ▷ (Compute search direction)

if $n >$ minSteps and ($|\boldsymbol{v}| \approx 0$ or $|\boldsymbol{v}| > 0.9999|\boldsymbol{v}_{\text{prev}}|$) **then**

Break from loop

$\boldsymbol{v}_{\text{prev}} \leftarrow \boldsymbol{v}$ ▷ (Save $\boldsymbol{v}$ for future convergence tests)

$\boldsymbol{v} \leftarrow \text{stepSize}\,(\boldsymbol{v}/|\boldsymbol{v}|)$ ▷ (Compute step vector)

$\boldsymbol{x}_1 \leftarrow \mathbf{p}, \quad a \leftarrow E(\boldsymbol{x}_1)$

$\boldsymbol{x}_2 \leftarrow \mathbf{p} + \boldsymbol{v}, \quad b \leftarrow E(\boldsymbol{x}_2)$

$\boldsymbol{x}_3 \leftarrow \mathbf{p} + 2\boldsymbol{v}, \quad c \leftarrow E(\boldsymbol{x}_3)$

$R \leftarrow 1$ ▷ (Initialize refinement counter)

while $R <$ maxRefinements **do**

if $a > b > c$ **then** ▷ (Step forward in search direction)

$\boldsymbol{x}_1 \leftarrow \boldsymbol{x}_2, \quad a \leftarrow b$

$\boldsymbol{x}_2 \leftarrow \boldsymbol{x}_3, \quad b \leftarrow c$

$\boldsymbol{x}_3 \leftarrow \boldsymbol{x}_3 + \boldsymbol{v}, \quad c \leftarrow E(\boldsymbol{x}_3)$

else ▷ (Shorten step size)

$\boldsymbol{v} \leftarrow \boldsymbol{v}/2$

$\boldsymbol{x}_3 \leftarrow \boldsymbol{x}_2, \quad c \leftarrow b$

$\boldsymbol{x}_2 \leftarrow \boldsymbol{x}_1 + \boldsymbol{v}, \quad b \leftarrow E(\boldsymbol{x}_2)$

$R \leftarrow R + 1$

Return($\boldsymbol{x}_1$)

4.4.2 Multi-Feature Tracker Implementation

In addition to the single-feature tracker of §4.4.1, our proposed method can also be used in a multi-feature tracking framework (see §4.2). To incorporate our gyro prior, we add a collection of terms to (4.2) which penalize each feature's deviation from its predicted position. Thus, we minimize the following regularized multi-feature energy function:

$$c_{\text{multi}}(\boldsymbol{x}) = \frac{1}{n^2}\sum_{f=1}^{F}\sum_{\boldsymbol{u}\in\Omega}\psi\left(T_f(\boldsymbol{u}) - I(\boldsymbol{u}+\mathbf{S}_f\boldsymbol{x})\right) + \lambda\sum_{f=1}^{F}\frac{\ln(\alpha|\mathbf{S}_f\boldsymbol{x}-\boldsymbol{x}_{f,\text{gyro}}|+1)}{\ln(\alpha x_{\max}+1)}, \tag{4.14}$$

where $\boldsymbol{x}_{f,\text{gyro}}$ denotes the prior estimate of position for feature f (2D position, in units of pixels). As with the single-feature implementation, we use the absolute-value loss function for ψ. This energy function can be minimized by whatever means would be used to minimize (4.2). We minimize this energy in a coarse-to-fine scheme using 4 pyramid levels, and we use a slightly modified version of the gradient descent method described in Algorithm 3. The only difference is that the search direction is modified to prevent strong features from completely controlling the search direction, causing weak features to be lost. The search direction computation is detailed in Algorithm 4, and is very similar to the method used in [67].

Algorithm 4 Search direction for multi-feature tracker with gyro prior

Input: Dc_{multi}: Gradient of energy function, F: number of features
Output: $\boldsymbol{v}$: Final search direction

```
a ← −Dc_multi
a ← a/|a|
for f = 1 : F do
    y_f ← [a(2f − 1), a(2f)]^T
    b_f ← y_f/|y_f|
b ← [b_1^T, b_2^T, ..., b_F^T]^T
v ← 0.5a + 0.5b
```

Like the single-feature tracker, we compute the gradients of the image-template fit terms numerically, using the centered gradient approximation and sampling 0.25 pixels in each direction. However, rather than grouping together regularization terms with

image-template fit terms, we evaluate the gradients for these explicitly. We will denote the sum of all of the gyro-prior terms in (4.14) using P:

$$P = \lambda \sum_{f=1}^{F} \frac{\ln(\alpha|\mathbf{S}_f \boldsymbol{x} - \boldsymbol{x}_{f,\text{gyro}}| + 1)}{\ln(\alpha x_{\max} + 1)}. \tag{4.15}$$

Now, let x_i denote the i'th entry of $\boldsymbol{x}$, and let $x_{i,\text{gyro}}$ denote the gyro-derived prior estimate of x_i. The gradient of P is given by:

$$\frac{dP}{dx_i} = \frac{\lambda\alpha(x_i - x_{i,\text{gyro}})}{\ln(\alpha x_{\max} + 1)(\alpha N_i + 1)N_i}, \tag{4.16}$$

where:

$$N_i = \sqrt{(x_i - x_{i,\text{gyro}})^2 + (x_{i+1} - x_{i+1,\text{gyro}})^2} \quad \text{if } i \text{ is odd, and}$$
$$N_i = \sqrt{(x_{i-1} - x_{i-1,\text{gyro}})^2 + (x_i - x_{i,\text{gyro}})^2} \quad \text{if } i \text{ is even.}$$

We also present results for our multi-feature tracker with gyro prior with an additional rank penalty from [67]. We use the rank penalty based on empirical dimension and the centered trackpoint matrix and we compute the gradient of the rank term using the method described in [67]. When minimizing the energy function with an additional rank penalty term, we use the same algorithm (including the search direction rule) as we use when there is no additional rank term. The source code for the multi-feature tracker will be available on our supplemental web page.

4.5 Experiments

To evaluate our method, we collected multiple video sequences using a custom-built gyro-optical data collection system. This system simultaneously collects imagery from a standard webcam and gyro data from an ST L3GD20 3-axis MEMS gyroscope (see 4.8.1 for more hardware information). A quantitative evaluation of feature trackers requires trustworthy ground-truth trajectories to be known for a large set of features. Manual feature registration or manual correction of feature trajectories in real-world, low-quality video is both challenging and somewhat subjective. Thus, we collected many

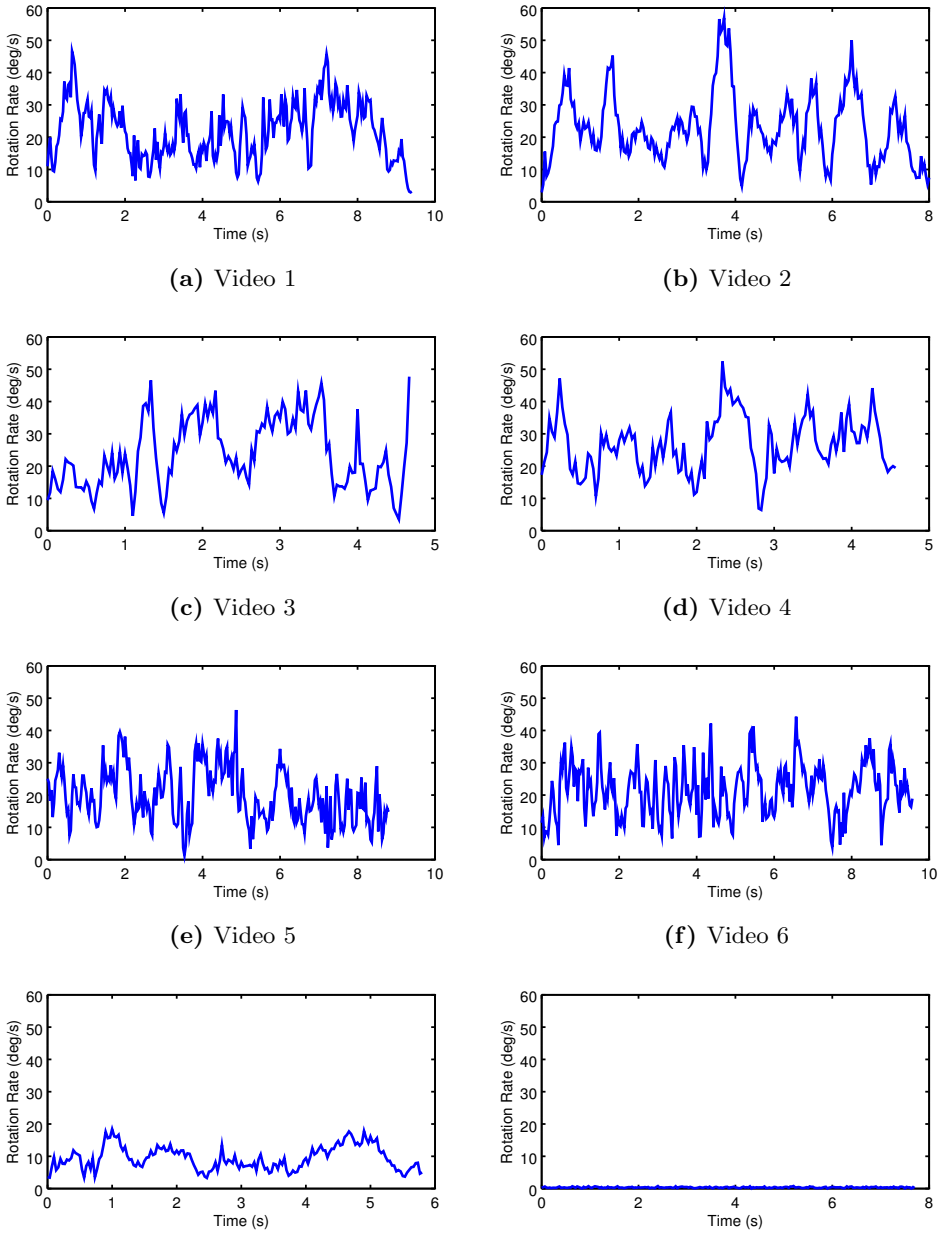
sequences under mostly favorable conditions and synthetically degraded their quality. We used a standard KLT tracker on the non-degraded videos to generate ground-truth (the output was human-verified and corrected). To mimic different levels of video quality we present quantitative results for two different levels of synthetic degradation, which we refer to as "low" and "high" degradation. 4.8.2 contains precise descriptions of how the videos were degraded, along with an example frame with the two different levels of degradation. In addition to these two experiments, we tested our method on several genuine low-quality videos (not synthetically degraded). Ground-truth is not known for these videos, but we overlay output trajectories so the tracking quality can be inspected visually (all trackers are initialized on a common set of features in frame 0). It is easier to evaluate the results by viewing the trajectories overlayed on the videos themselves, so this is included in the supplementary material. We refer to this as our qualitative experiment. Table 4.1 summarizes the maximum and typical rotation rates experienced in each video sequence in both sets of test videos. Figures 4.4 and 4.5 show complete rotation rate profiles (rotation rates as functions of time) for each video in the two test sets. Figure 4.6 shows some characteristic tracker results on videos from our quantitative high-degradation experiment. Sample output frames for several videos from our qualitative experiment are presented in Figures 4.7, 4.8, 4.9, and 4.10.

Our test set for our quantitative experiments consists of 8 video clips, ranging in length from 132 to 289 frames. For generating ground truth new features are detected throughout each sequence, and features are terminated when they leave the field of view or when they could no longer be localized by hand. In total, there are $50,754$ feature-frames in our set of test videos. This is the sum of the lifespan (in frames) of each feature in each video. We have both indoor and outdoor sequences in our collection of test videos. It should be noted that the test videos are biased in the sense that they focus on situations which are difficult for un-aided feature trackers. In particular, all of the test videos are degraded, which makes features less distinctive and corner-like than ideal features in high-quality video. We focus on these situations because this is where there is room for improvement over conventional feature tracking, but one must remember the context of these experiments. There is little or no benefit to using our method when tracking ideal, corner-like features; existing methods like KLT can handle such features perfectly well. In our test videos there is significant flow due to camera

translation through the environment, and in some instances due to non-rigidity of the subject matter (i.e. effects other than camera rotation). This is important to show that the proposed method can resolve conflicts between predicted and observed flow, and that it is actually using both the gyros and the imagery for tracking, as opposed to just predicting feature locations in each frame using the gyros.

Table 4.1: Rotation rate summary for quantitative and qualitative test videos.

	Video	Max Rotation Rate (deg/s)	Mean Rotation Rate (deg/s)	Median Rotation Rate (deg/s)
Quantitative Tests	Video 1	46.554	21.926	21.812
	Video 2	56.925	22.983	21.776
	Video 3	47.702	23.849	21.537
	Video 4	52.408	26.602	26.455
	Video 5	46.347	19.893	18.956
	Video 6	44.285	21.538	21.532
	Video 7	18.397	9.4527	8.9123
	Video 8	0.79148	0.36397	0.35447
Qualitative Tests	Low-Quality 1	107.462	36.169	36.641
	Low-Quality 2	35.282	17.644	17.598
	Low-Quality 3	33.25	15.539	15.864
	Low-Quality 4	36.676	13.49	14.168
	High-Quality 1	30.804	8.2969	9.2238
	High-Quality 2	14.214	7.8396	7.9647
	High-Quality 3	10.486	3.5625	3.8939



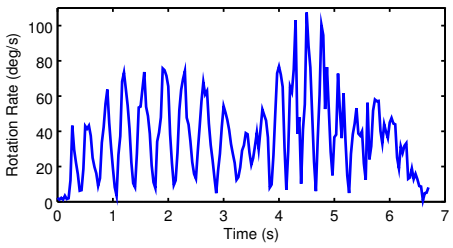
(a) Video 1 **(b)** Video 2

(c) Video 3 **(d)** Video 4

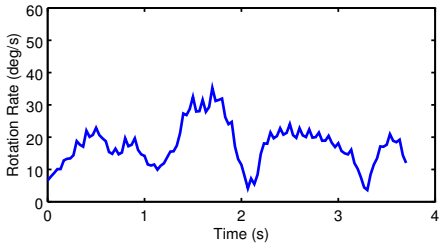
(e) Video 5 **(f)** Video 6

(g) Video 7 **(h)** Video 8

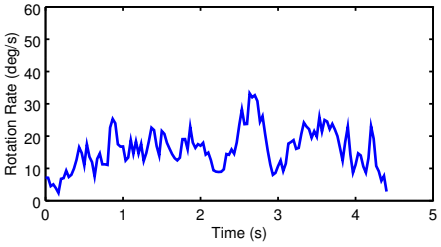
Figure 4.4: Rotation Rates as functions of time in Quantitative Test Videos. Video 7 contained several moving bodies (cars) and Video 8 had a stationary camera watching a non-rigid body. These are meant to demonstrate that the proposed method does not introduce significant failures when the predicted flow deviates substantially from the true flow.



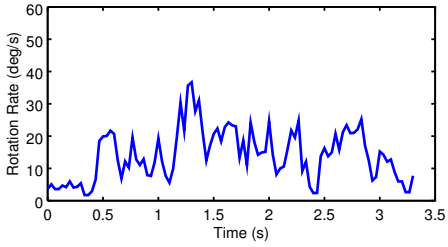
(a) Low Quality 1



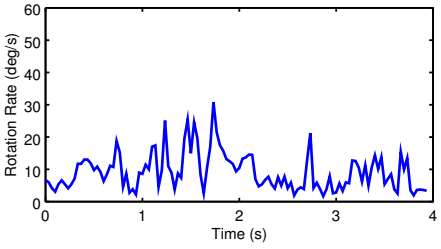
(b) Low Quality 2



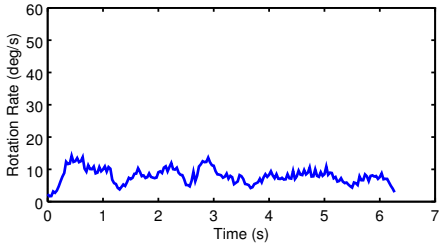
(c) Low Quality 3



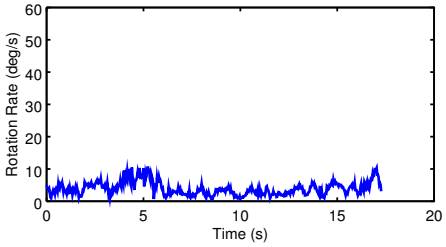
(d) Low Quality 4



(e) High Quality 1

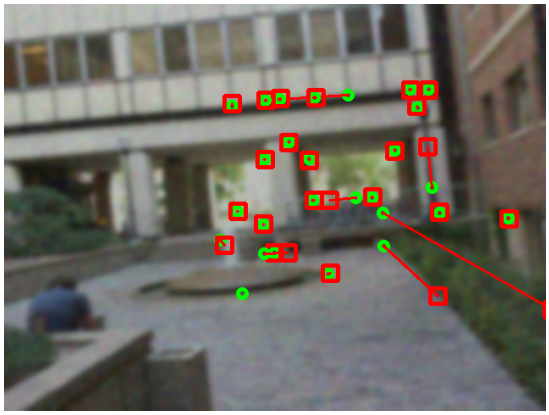


(f) High Quality 2



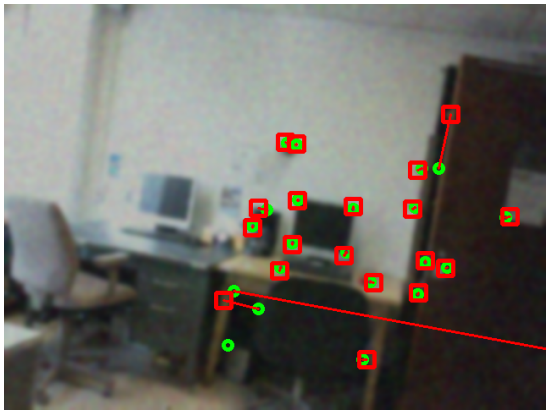
(g) High Quality 3

Figure 4.5: Rotation Rates as functions of time in Qualitative Test Videos. As in the quantitative experiments, we captured a few high-quality videos (some with multiple moving bodies and some where there is little rotation so gyro-aiding should not be needed). These demonstrate that our method does not create significant problems in such cases.



(a) KLT + gyro init.

(b) Our method

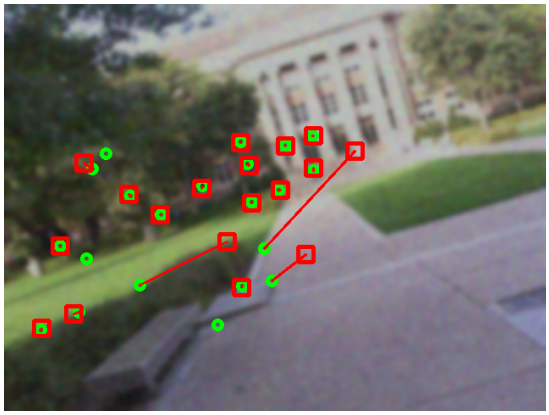


(c) KLT + gyro init.

(d) Our method

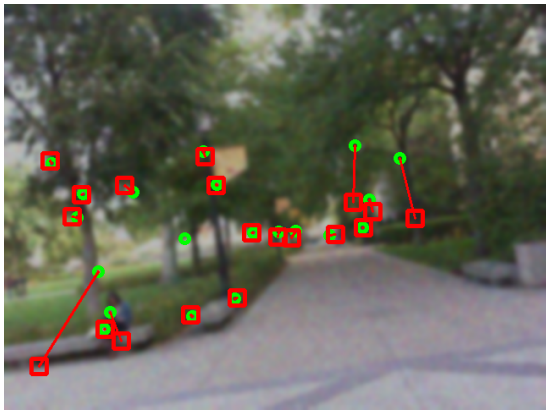
(e) KLT + gyro init.

(f) Our method

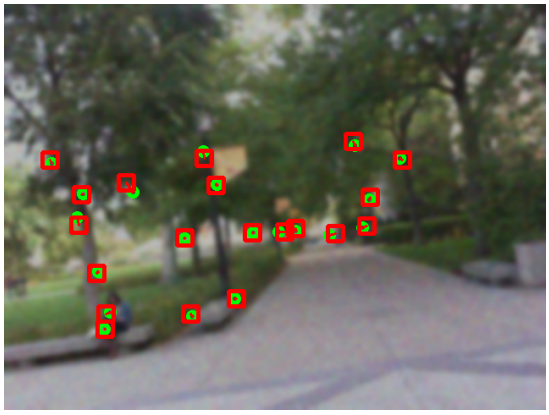


(g) KLT + gyro init.

(h) Our method



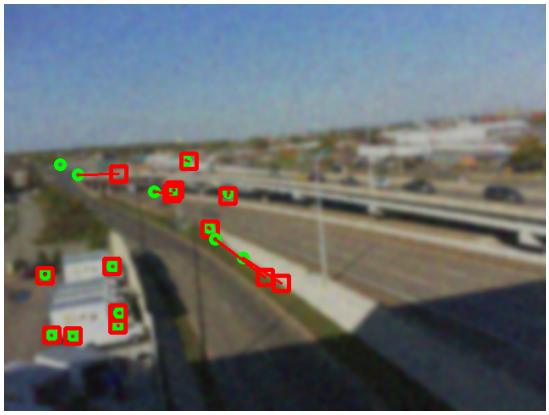
(i) KLT + gyro init.



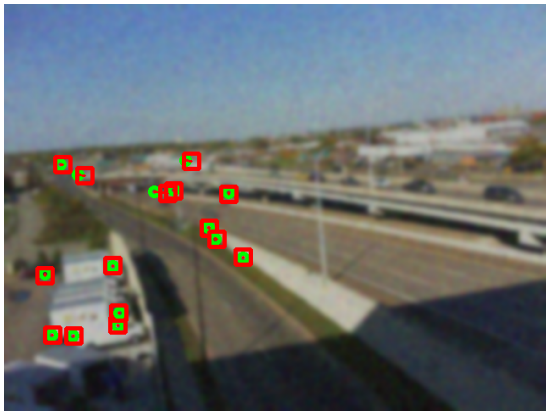
(j) Our method

(k) KLT + gyro init.

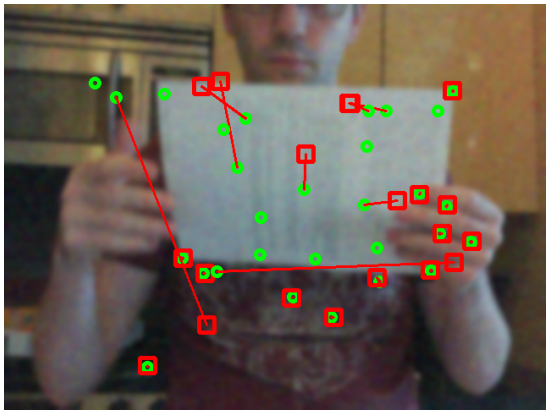
(l) Our method



(m) KLT + gyro init.



(n) Our method



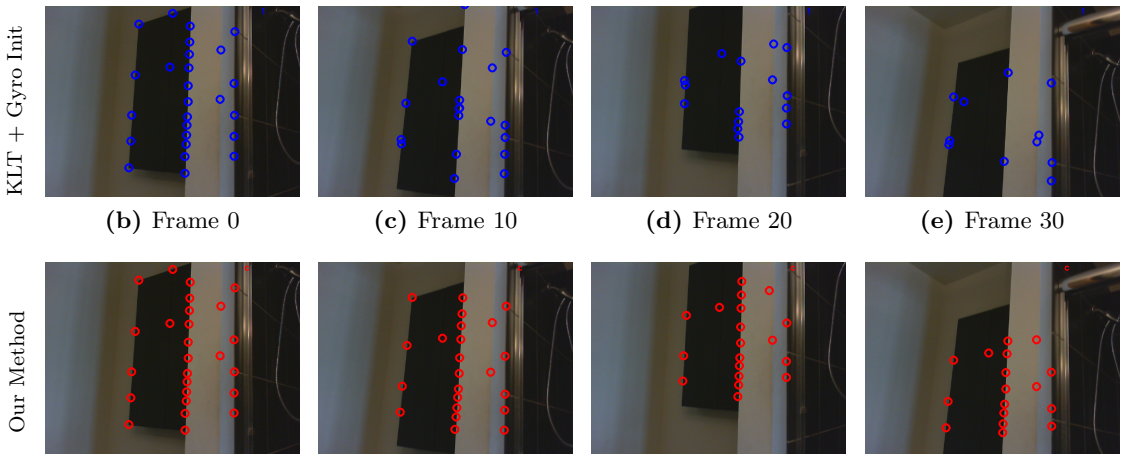
(o) KLT + gyro init.

(p) Our method

Figure 4.6: Characteristic results for KLT with gyro initialization and 1^{st}-order descent tracker with gyro initialization and regularization (denoted "Our method") on high-degradation videos when all trackers are initialized with the same set of features and run without being corrected. Tracker output locations (red squares) are connected by red lines to ground-truth locations (green circles). Green circles without squares attached are lost features (drifted off-screen).

To evaluate a given tracker on a test video in our quantitative experiment, we initialize the set of tracked features using ground-truth feature positions. When the tracker "loses" a feature (which we define as wandering by at least 10 pixels from ground truth), the feature is re-initialized using its current ground truth position. The mean number of frames between re-initializations (a.k.a. mean track length) is used as our

performance measure. This is very similar to the performance measure used in [33, 67] to evaluate rank-constrained feature trackers. An alternative performance measure would be to initialize all trackers on a common set of features, let them run un-aided for some number of frames and then compute mean drift or the average L^1 or L^2 deviation of the output trajectories from ground-truth. Unfortunately, these simple measures are rather unstable because once a feature is lost trackers can behave very unpredictably. Some will drift around slowly while others may quickly wander off screen or snap to the closest corner-like object. While these differences have a large effect on trajectory error, they are not practically very important. What is of interest is how long a feature tracker can hold features and how well it tracks them during that period. Thus, we select mean track length for our performance measure.

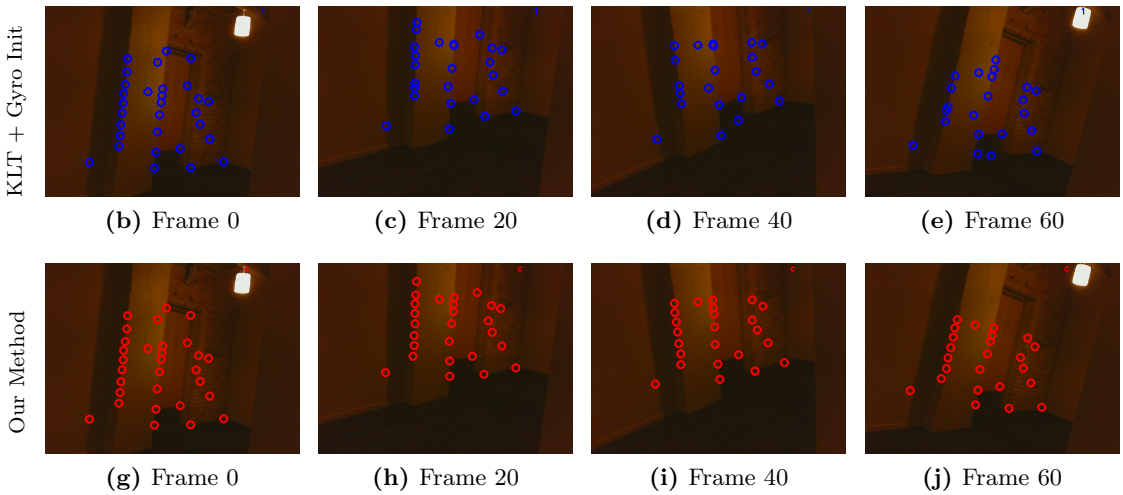


(b) Frame 0 **(c)** Frame 10 **(d)** Frame 20 **(e)** Frame 30

(g) Frame 0 **(h)** Frame 10 **(i)** Frame 20 **(j)** Frame 30

Figure 4.7: Sample frames showing the output of KLT + Gyro Initialization (blue circles) and our method (red circles) on a real-world low-light video ("Low Quality 1") with many ambiguous features. Notice the number of features that are lost by KLT + Gyro Init.

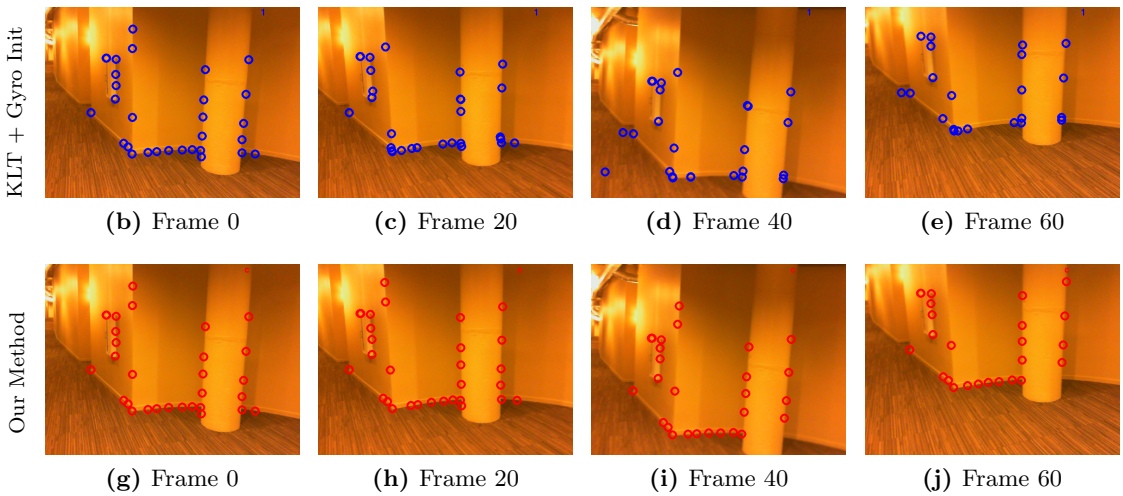
In this experiment we compared a pyramidal KLT tracker (OpenCV 2.4.6 implementation), the same OpenCV KLT tracker but with initial flow estimates computed using the gyroscope, which we refer to as "KLT + Gyro init" (similar to [56]), a standard gradient-descent single-feature tracker ("1$^{\text{st}}$-Order Descent"), a gradient descent tracker with gyro initialization ("1$^{\text{st}}$-Order Descent + Gyro init"), our proposed method with



(b) Frame 0 **(c)** Frame 20 **(d)** Frame 40 **(e)** Frame 60

(g) Frame 0 **(h)** Frame 20 **(i)** Frame 40 **(j)** Frame 60

Figure 4.8: Sample frames showing the output of KLT + Gyro Initialization (blue circles) and our method (red circles) on a real-world low-light video ("Low Quality 2") with many ambiguous features. Notice the bunching of features along the column edges with KLT + Gyro Init.



(b) Frame 0 **(c)** Frame 20 **(d)** Frame 40 **(e)** Frame 60

(g) Frame 0 **(h)** Frame 20 **(i)** Frame 40 **(j)** Frame 60

Figure 4.9: Sample frames showing the output of KLT + Gyro Initialization (blue circles) and our method (red circles) on a real-world low-light video ("Low Quality 3") with many ambiguous features. Notice the ambiguous features wandering and accumulating in corner areas with KLT + Gyro Init.

(b) Frame 0 **(c)** Frame 20 **(d)** Frame 40 **(e)** Frame 60

(g) Frame 0 **(h)** Frame 20 **(i)** Frame 40 **(j)** Frame 60

Figure 4.10: Sample frames showing the output of KLT + Gyro Initialization (blue circles) and our method (red circles) on a real-world low-light video ("Low Quality 4") with many ambiguous features. Notice that several features "jump" with KLT + Gyro Init (lower left of pillar and along the middle section of the image frame).

the single-feature tracking implementation ("1$^{\text{st}}$-Order Descent + Gyro Prior"), a modern rank-penalized multi-feature tracker from [67], which we refer to as "Multi-Tracker + Rank" (which uses "empirical dimension" for rank estimation and a centered trackpoint matrix for generating the rank penalty), our proposed method with the multi-feature implementation ("Multi-Tracker + Gyro Prior"), and finally, our multi-feature implementation integrated with the rank penalty of [67] ("Multi-Tracker + Gyro Prior + Rank"); the rank penalty here is also based on "empirical dimension" and a centered trackpoint matrix. All trackers were implemented pyramidally with 4 resolution levels. For those trackers that do not use the gyro for initialization ("KLT", "1$^{\text{st}}$-Order Descent", and "Multi-Tracker + Rank"), features are initialized using "average flow initialization". With this scheme, a given frame is registered against the previous at 1/4'th resolution to get the displacement, $\boldsymbol{a}$, of the new frame. Then, each feature is initialized in the new frame with position $\boldsymbol{x}$ given by:

$$\boldsymbol{x} = \boldsymbol{x}_{\text{prev}} + \boldsymbol{a}, \tag{4.17}$$

where $\boldsymbol{x}_{\text{prev}}$ is the position of the same feature in the previous frame.

Some of the trackers in this comparison have tuning parameters. In our method λ controls the relative strength of the gyro prior term, and in the rank-penalized multi-feature tracker there is a similar coefficient to control the strength of the rank penalty. The "Multi-Tracker + Gyro Prior" has both of these parameters. For learning these parameters we collected an additional 4 videos using our data collection system and generated ground-truth for them. We made a set of training videos by including these 4 videos as well as 2 degraded copies of each ("low" and "high" degradation). The parameters were learned by exhaustively searching for the values which gave the best average performance across all training videos. The learned parameters for each tracker were used in both the low-degradation and high-degradation experiments. No tracker had access to any of the 8 test videos during training. See 4.8.2 for the learned parameter values for each tracker.

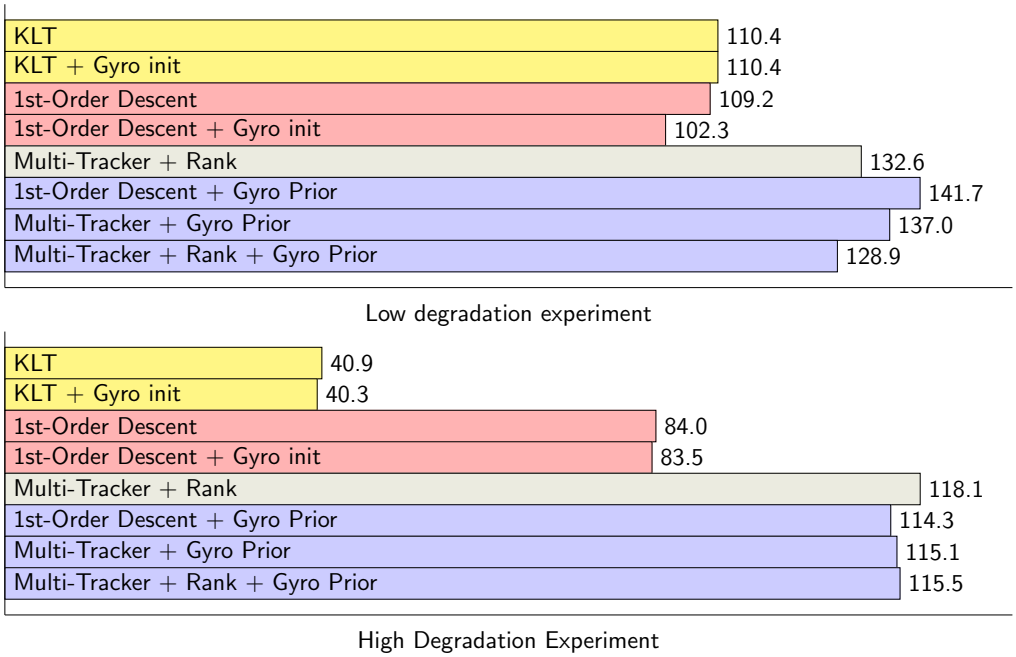


Figure 4.11: Mean track length (frames) for low and high degradation experiments. Higher is better.

4.5.1 Analysis of Results

In Figure 4.11 we present average track lengths for each tracker for both the low degradation and high degradation experiments. These results are averaged across all of our test videos. Tables 4.2 and 4.3 show per-video comparisons for each experiment. Table 4.4 shows the average processing rate (measured in frames per second) of each tracker on our testing computer (3^{rd}-generation Intel Core i5-based laptop). These processing rates only take into account time spent inside the actual tracking routines and do not include common processing tasks like loading frames from the disk into memory. It should be noted that each method was configured (where possible) for the best tracking performance, and no compromises were made to reduce computational cost. The rank-penalized multi-feature tracker was able to run closer to 15 frames per second with different settings, although this resulted in slightly degraded performance.

Table 4.2: Mean track length (frames) - Low degradation. Methods with integrated gyro prior are highlighted in gray. Higher is better. Winning entries are bold.

		Video Number								Average
		1	2	3	4	5	6	7	8	
Tracker	KLT	81	128	19	**113**	**290**	93	150	9	110
	KLT + Gyro init	79	126	19	**113**	**290**	92	156	9	110
	1st-Order Descent	157	103	100	47	124	176	156	12	109
	1st-Order Descent + Gyro init	154	103	75	48	108	169	150	12	102
	Multi-Tracker + Rank	188	103	**134**	64	193	**223**	133	23	133
	1st-Order Descent + Gyro Prior	**214**	**165**	82	73	130	166	**282**	20	**142**
	Multi-Tracker + Gyro Prior	179	123	130	65	198	208	167	**26**	137
	Multi-Tracker + Rank + Gyro Prior	175	104	**134**	58	184	**223**	129	24	129

Table 4.3: Mean track length (frames) - High degradation. Methods with integrated gyro prior are highlighted in gray. Higher is better. Winning entries are bold.

		Video Number								Average
		1	2	3	4	5	6	7	8	
Tracker	KLT	52	68	17	29	52	60	45	4	41
	KLT + Gyro init	52	61	17	29	54	62	44	4	40
	1st-Order Descent	135	81	75	40	81	125	125	9	84
	1st-Order Descent + Gyro init	135	79	66	39	79	125	137	9	84
	Multi-Tracker + Rank	183	101	104	**56**	**160**	**208**	110	22	**118**
	1st-Order Descent + Gyro Prior	**208**	**130**	74	53	88	120	**226**	15	114
	Multi-Tracker + Gyro Prior	171	103	110	**56**	137	176	146	**23**	115
	Multi-Tracker + Rank + Gyro Prior	171	109	**119**	52	145	194	113	21	116

The first thing to notice is that we seldom see any advantage from only initializing KLT or the gradient descent tracker using the gyroscope. This is seen in both the

Table 4.4: Average processing frame rate (frames per second). Methods with integrated gyro prior are highlighted in gray. Higher is better.

Tracker	FPS
KLT	101.2
KLT + Gyro init	100.9
1st-Order Descent	44.9
1st-Order Descent + Gyro init	49.8
Multi-Tracker + Rank	5.1
1st-Order Descent + Gyro Prior	41.8
Multi-Tracker + Gyro Prior	37.3
Multi-Tracker + Rank + Gyro Prior	4.9

low and high-degradation experiments, where the performance of both KLT and 1st-order descent appears independent of the initialization scheme which was used. The only problems that can be fixed by better tracker initialization are convergence problems and it appears that the 4-level pyramidal tracking scheme, combined with careful optical-only initialization, appears to be good enough to ensure minimization of the respective energy functions in our experiments.

This is contrary to the findings of some other authors, for instance, [56], where significant improvements in tracking performance were attributed to gyro-based initialization. However, it is suggested in [56] that for their un-aided tracker they simply initialize each feature using its location from the previous frame. This is certainly inferior to estimating "average flow" between frames, as we did for the non-gyro-initialized trackers in our experiments. Estimating average flow can at least compensate for the common components of flow due to large camera rotations about axes orthogonal to the optical axis. Our combined results would suggest that gyro-based initialization is indeed superior to naive initialization, but with a little effort (and no additional hardware) one can achieve the same results with careful, optical-only initialization. The consequence of this is that to gain true performance advantages from gyroscopes one must employ some other mechanism for exploiting them, beyond initialization (In [56] they also pre-warp template images, which is another exploitation strategy which we do not explore in this work).

Another observation is that both regularization with a gyro-derived prior estimate of flow and low-rank regularization offer measurable performance advantages over un-regularized trackers. In the low-degradation experiment the improvement is somewhat modest, while it is much larger in the high-degradation experiment. This makes sense

because when the video is higher quality, features are frequently distinctive enough to enable tracking without needing additional information. In all videos in both sets of experiments, both forms of regularization result in approximately the same or better performance than 1st-order descent (with or without gyro initialization). There are two videos (#4 and #5), however, in the low-degradation experiment where KLT outperforms all other methods (again, regardless of initialization). The regularized trackers still offer better performance than un-regularized 1st-order descent, however, which suggests that KLT's advantage in these videos is due to the Gauss-Newton optimization scheme, rather than its lack of regularization. When the energy function is nice enough, Gauss-Newton optimization can cover large distances and converge in a very small number of iterations compared to first-order methods. This may be the source of KLT's advantage in these instances. In the high-degradation experiment both forms of regularization offer clear advantages to both KLT and 1st-order descent. It is also clear from these experiments that even without regularization 1st-order descent offers greater reliability than the Gauss-Newton scheme used by KLT. This is in line with intuition, since taking relatively large steps based on local information can be risky when the local information is poor-quality. The fast line search used by our first-order methods is a safer (albeit slower) approach. It should also be noted that while both regularization with a gyro-derived prior estimate of flow and low-rank regularization tend to offer better performance than un-regularized trackers, the improvements offered from these two techniques are not complimentary. That is, combining these two techniques does not offer a significant advantage over using just one of them. Thus, it would not be advisable for one to use both techniques together (as in the "Multi-Tracker + Rank + Gyro Prior").

Finally, the performance of the rank-penalized multi-feature tracker and the trackers which use gyro-based regularization are very similar in both experiments. The advantage that the gyro-regularized tracker has is speed. As can be seen from Table 4.4, the gyro prior term adds very little to the computational expense of a tracker. On the other hand, rank-penalized multi-feature tracking is quite expensive. Even when configured for speed (where the tracker can run at approximately 15 fps in exchange for slightly degraded performance), the rank-penalized multi-feature tracker is several times slower than the single-feature 1^{st}-order descent tracker with gyro prior.

4.6 Future Work

We see three major avenues for continuing this work. The first is to use the regularization technique that we propose with more modern, faster optimization algorithms. The 1st-order descent with line search used in this work is reliable and we chose to use it so that we could evaluate the fundamental ideas of this work without confusing core issues with the challenges of massaging more complicated nonlinear optimization algorithms. However, there is a gap in speed between the method we present and KLT, and the gap will only be closed by using a more sophisticated optimization algorithm.

The second area for farther work is to characterize the types of features that can be reliably tracked when exploiting a prior estimate of flow. It is well-known how to identify whether a given feature is distinctive enough for KLT to track it (see [27], for instance). We have shown in this work that when you have a prior estimate of flow and you exploit it by regularizing the tracking energy function, it is possible to track features that are not trackable on their own. It should therefore be possible to relax the requirements that are used in real-world applications to decide when to drop and replace bad features. This is an important aspect of any practical application of this work.

The third avenue for future work is to develop a framework for estimating the sensor calibration on-line. In order to exploit gyroscopes to predict flow, we need to have estimates of certain quantities, including gyro biases, relative sensor latencies, and $\tilde{\mathbf{K}}$. In this work we estimate these quantities off-line (see §4.3 and 4.8.1). This is not fundamentally problematic, except that some of the quantities that are needed can change with time. For instance, sensor latency can change whenever camera settings are changed, and gyro biases can drift slowly with time. It would make the techniques of this paper more accessible if these items could be estimated on-line by, for instance, adjusting the calibration constants to reduce the distances between flow predictions and measured optical flow in a handful of "nice" features. This would make it possible to exploit gyro-derived optical flow estimates without needing to worry about many of the practical details of the sensors involved.

4.7 Conclusion

We presented a deeply integrated method of exploiting low-cost gyroscopes to improve general purpose feature tracking. Beyond initializing the search for features using a gyro-derived optical flow prediction, we built on previous work in the area by also using the sensors to regularize the tracking energy function to directly assist in the tracking of ambiguous and poor-quality features. We demonstrated that our technique offers significant improvements in tracking performance over conventional template-based tracking methods, and is in fact competitive with more complex and computationally expensive state-of-the-art trackers, but at a fraction of the computational cost. Additionally, we showed that the practice of initializing a template-based feature tracker using gyro-predicted optical flow does not outperform careful optical-only initialization. This suggests that a more tightly integrated solution, like the one proposed here, is needed to achieve genuine gains in tracking performance from these inertial sensors.

Acknowledgements

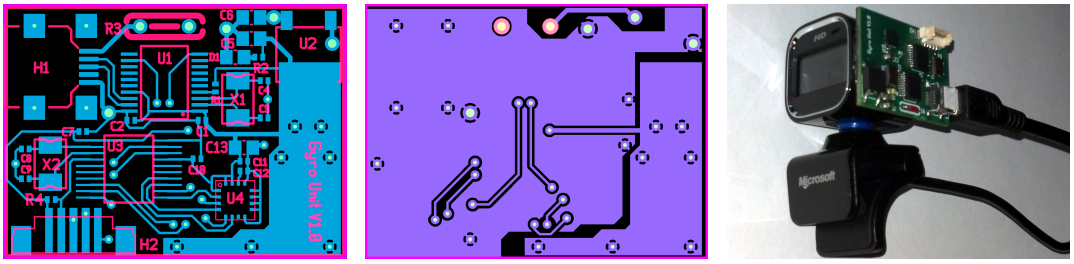
This work was supported by NSF awards DMS-09-56072 and DMS-14-18386, the University of Minnesota Doctoral Dissertation Fellowship Program, and the Feinberg Foundation Visiting Faculty Program Fellowship of the Weizmann Institute of Science.

4.8 Appendix For Part III

4.8.1 Data Collection Hardware

For our experiments we collected video and gyro data from a custom-built data collection system. This consists of a standard webcam (Microsoft LifeCam HD-6000) and a small custom circuit board with a 3-axis MEMS (Microelectromechanical system) gyroscope. This circuit board was physically attached to the webcam casing with glue to ensure that the camera and the gyro experienced the same rotations. The gyro is an ST L3GD20. It is controlled by a small 8-bit Microchip microcontroller (Pic 18F13K50), which uses a USB-serial adapter chip (Microchip MCP2200) to provide a USB interface. We wrote software for data collection that simultaneously collects imagery from the webcam and

gyro data from the L3GD20 and saves the data to a computer running Linux. Images of the gyro circuit and the full data collection system are shown in Figure 4.12. All files necessary to re-produce the gyro circuit, along with our data collection software will be available on our supplemental web page.



(a) PCB Top **(b)** PCB Bottom **(c)** Assembled System

Figure 4.12: Pictures of the data collection hardware. A small printed circuit board (PCB) with a gyroscope and USB interface was attached to a standard webcam.

Gyros suffer from various error sources. Because our proposed method only uses the gyros to propagate attitude for the short periods of time between consecutive frames, we do not need to worry about some of the smaller error sources. However, one source that must be accounted for is sensor bias. This is a constant (or very slowly changing) offset that gets added to each measurement. The biases are generally different on each sensor axis and they can change with temperature, humidity, and sensor age. Fortunately it is easy to measure and compensate for biases. You simply record and average stationary data for a few seconds and then subtract this value from all subsequent measurements. We call this de-biasing. We de-biased our gyros prior to recording each data set. It is also possible to estimate biases "on-line" by measuring typical deviation from gyro-based prior flow estimates and final values of flow. This would effectively eliminate the need for collecting stationary data before recording, but we did not attempt that in this work.

The imagery from the webcam and the gyro data from the custom circuit board are synchronized in software on the data collection computer. In order to use the gyro to predict optical flow it is important that the relative latency between the camera and gyro systems be known with high accuracy. An error as small as 0.01 seconds in

relative latency will result in a measurable drop in optical flow estimation accuracy. We have a program for estimating this latency (which will be available on our supplemental web page). One simply rotates the camera back and forth several times and the cross-correlation between the mean optical flow and the gyro rotation rate (as functions of time) is computed. The maximum of the cross-correlation corresponds to the relative latency of the two systems. This process works well and is rather simple. However, we found that the latency of our camera system depends on the exposure setting (which needs to be adjusted for different lighting conditions), and it can also change with CPU load. Additionally, we found the latency of the camera system to be less stable in low-light (high exposure time) settings. Care must be taken to ensure that this value is estimated correctly when recording test data or the gyro-predicted optical flow will be unusable. It is important to note that this is not a deficiency of our proposed method of integrating gyro-derived optical flow with feature tracking, but is instead a limitation of our data collection system. If one used a scientific camera (preferably with a global shutter) with strobe or trigger capability and synchronized the data in hardware, then relative latency would not even need to be estimated.

4.8.2 Degradation Process for Experiments

The synthetic degradation process for our experiments was a multi-step process. Each frame was degraded separately by first multiplying each pixel by a constant m, and then adding per-pixel, Gaussian, i.i.d noise with mean μ_1 and standard deviation σ_1. Frames were then blurred using a Gaussian mask with standard deviations σ_x and σ_y in the x and y directions, respectively. Finally, we added additional per-pixel, Gaussian, i.i.d noise with mean μ_2 and standard deviation σ_2.

The degradation adds noise before and after blurring to ensure the noise has strong high and low frequency components. The parameters for the low-degradation experiments were: $m = 0.9$, $\mu_1 = 0.0$, $\sigma_1 = 15.0$, $\sigma_x = 1.5$, $\sigma_y = 1.5$, $\mu_2 = 0.0$, $\sigma_2 = 1.5$. The parameters for the high-degradation experiments were: $m = 0.8$, $\mu_1 = 0.0$, $\sigma_1 = 30.0$, $\sigma_x = 3.0$, $\sigma_y = 3.0$, $\mu_2 = 0.0$, $\sigma_2 = 3.0$. A sample frame is shown in Figure 4.13 next to degraded copies of the image using both the high and low degradation profiles. Of course, the source code for degrading the videos will be available on our supplemental web page. We note that we originally used lower multipliers for both degradation

(a) Original Frame **(b)** Low Degradation **(c)** High Degradation

Figure 4.13: Sample frame and synthetically degraded copies.

profiles to farther darken the videos, as done in [19]. However, we found that the performance of the KLT implementation we were using (OpenCV) began to fall off sharply when using multipliers below 0.8. Since this was synthetic degradation we felt it was unfair to select values that seemed to cause unreasonable harm to one of the trackers.

Learned Parameters

For each tracker with parameters, the parameters were learned via the process described in §5 of the paper. The learned parameters for each tracker are given in table 4.5.

Table 4.5: Learned parameters for each feature tracker (trackers not listed in this table did not have tuning parameters)

Tracker	Learned Parameters
Multi-tracker + Rank	rank coeff. $= 0.45$
1^{st}-order descent + Gyro prior	$\lambda = 0.0125$
Multi-tracker + Gyro prior	$\lambda = 0.005$
Multi-tracker + Rank + Gyro prior	rank coeff. $= 0.4$, $\lambda = 0.00003$

Chapter 5

Conclusion and Discussion

In part I we presented a solution to the two-view motion segmentation problem. We exploited the fact that feature trajectories belonging to a rigid object exist in a low-dimensional subspace using a linear or non-linear (kronecker) embedding. This allowed us to formulate (as other authors have done) motion segmentation as a subspace clustering problem. We developed a novel solution to this problem (GDM) and showed that it performed well at solving this old and well-studied problem.

In part II we used the same low-rank observations about feature trajectories in a different way. Instead of using them to segment features in a scene, we used them to develop a feature tracker that can share information between features to enable tracking of poor-quality feature points. When starting on this project, the original intent was to forcefully impose restrictions on the resulting optimization problem to force the tracker to choose trajectories that are consistent with rigid motion. The usefulness of this solution is limited, however, due to the fact that one must know a-priori that a cohort of tracked features do, in fact, belong to a single rigid body. This is something that is not typically known until after feature tracking has been performed. We found great utility, however, in a modification to this idea. Instead of forcing trajectories to respect the low-rank constraint, we only weekly encouraged the tracker to select trajectories consistent with this constraint. The "soft" constraint allowed us to get out of the way and not interfere with the tracking of strong, crisp features (features which a conventional tracker doesn't need help with anyways). However, when a scene contains some poor-quality features, the penalty becomes significant and makes it possible to

resolve the position of features that might not be trackable on their own. This soft constraint approach was able to improve feature tracking even in scenes with multiple objects and non-rigid objects, making it more broadly applicable than our original idea.

In part III we took the soft constraint portion of part II and used the same idea in a different way. Instead of relying on low-rank constraints to share information between features, we brought in outside help in the form of independent feature flow estimates from gyroscopes rigidly mounted to the camera. We found that we were able to achieve substantial improvements in feature tracking using this approach. The major drawback of this solution was the addition of external sensors, which adds complexity and may be prohibitive in some settings. However, when the data is available for this approach, it can achieve very good tracking performance and run very quickly compared to the solution from part II.

A major theme of the projects we have presented is the use of low-rank constraints for feature tracking and motion segmentation. Part III is somewhat of a detour from this, but this is at the heart of parts I and II. We have shown that low-rank structure in feature trajectories enables motion segmentation through subspace clustering. We have also shown that this low-rank structure can improve our ability to track features in the first place. A natural question to ask is whether we can devise a way of combining the tracking and segmentation problems and exploit this structure to get an even greater benefit in both tracking and segmentation.

The first thing we will look at in exploring this question is the main argument against it, from my viewpoint. Real-world computer vision problems are often highly complex and solutions require bringing together many different tools and ideas. One thing that makes it possible to develop, test, and debug solutions to these complex problems is modularity. As an example, we will briefly consider the complex problem of 3D scene reconstruction from a set of images. All solutions that I am aware of rely on a processing pipeline that breaks this large and difficult problem into a sequence of smaller, simpler problems:

- At the front of the pipeline, one typically analyzes each image separately and looks for feature points which we expect/hope will be distinctive (a decision generally made without regard to any of the other images in our collection).

- These features are extracted and characterized using some form of descriptor. At this stage we throw away the raw imagery[1] and move forward with our set of extracted features.
- Next, we start comparing features between different pairs of images and try to identify pairs that share a large number of features.
- We match features in each pair to develop sets of correspondences.
- These correspondences are used to estimate an essential or fundamental matrix between each pair of overlapping images.
- Each matrix is then decomposed to estimate the relative positions and orientations of the cameras associated with each image pair.
- At this point we start the pose recovery stage, where we take our large set of relative positions and orientations and try to find a position and orientation for each camera that respects as many pairwise measurements as possible. This stage makes no use of raw imagery, or feature descriptors, or pairwise essential or fundamental matrices directly. Instead it only uses the pairwise relative position and orientation measurements that were extracted from all of this data.
- after we have found positions and orientations for each camera we triangulate positions for each feature.
- We (optionally) go back and adjust feature positions and camera positions/orientations to make sure that features project onto the correct locations in each image (a process called bundle adjustment).

Each of these steps uses outputs from some other step or steps and tries to ignore as much of the larger problem as it can, while it focuses on one specific task. This type of processing pipeline is not unusual in computer vision applications, and it demonstrates the importance of modularity. Imagine trying to understand why an algorithm isn't working correctly without being able to break up the problem into smaller parts

[1] In the sense that we don't use it in later stages

and inspect and validate each component separately. The point of this example is to demonstrate that combining different problems in computer vision comes at a cost.

The above argument is not meant to imply that problems should *never* be combined. However, we must achieve a significant and tangible benefit from combining the problems in order to offset the reduction in modularity. If the combined solution doesn't work significantly better than solving each problem separately, then they should probably be kept separate. What this means in our situation is that we must realize an improvement from a joint tracking and segmentation framework over performing feature tracking and segmentation as two separate problem, even if we successfully exploit low-rank constraints in our solutions to each of these problems separately.

So the question is, what is there to be gained by combining these problems and developing a joint tracking and segmentation framework? If we assume that we are already getting everything we can out of the low-rank structure of feature trajectories in the tracking and segmentation problems, then there is nothing to be gained. This is not the case though. In theory, we are giving something up in our solution to feature tracking by imposing our constraint weakly. By not forcing the tracker to respect the constraint, we make it possible for the tracker to be confused in some situations and incorrectly resolve some features. Also, our approach essentially allowed strong features to be tracked normally, which made the constraint naturally adapt to the scene. For instance, if there were two distinct motions in a scene and each had at least a few good features, then the empirical dimension of the set of trajectories will be higher than it would be if there were only one rigid motion. This makes the constraint less strict when there are many different objects than if there is only one, since the subspace we are trying to push ambiguous trajectories towards is effectively larger. This works rather well, but we could do better still if we knew the precise dimension that a collection of trajectories was supposed to have a-priori, and strictly required the trajectories to reside in a subspace of the correct dimension. One situation where the potential advantage can be readily seen is in the case of partial occlusion. If we have features on an object that is partially occluded, then the soft constraint approach we presented in part II will generally let the occluded features stick to the occluding object. However, if we know the correct dimension of a set of trajectories in advance and we force a hard constraint when tracking, then it is possible for the non-occluded features to over-power the influence

of the occluded features and allow us to infer their correct positions even though the features are not visible in the imagery. In summary, our soft constraint approach in part II was necessary to make the tracker useful without knowing the structure of the scene or the number of objects in it. However, if we had a joint tracking and segmentation framework, this information would theoretically be available and we could gain some additional advantages by imposing hard constraints when tracking. This is where the potential advantage of a joint tracking and segmentation framework comes from, over performing each of these tasks separately.

A first approach to a combined tracking and segmentation framework, based on the work from the previously presented projects might be to minimize an energy function that combines image-template fit terms and a term for the global dimension of the set of trajectories. The state for such a tracker includes feature positions as well as scene segmentation, and it alternates between solving for feature positions and updating the segmentation hypothesis. In this arrangement we would compute the global dimension of the soft assignment as is done in GDM. Unfortunately, this approach has some problems. First, making the global dimension term very strong is not the same as imposing strict agreement with a hybrid-linear model. Indeed, giving the global dimension term a large coefficient would drive features towards degenerate configurations (where the GD approaches 0). On the other hand, if we add the term weakly, then in effect we have something very similar to a straightforward combination of parts I and II of this thesis. As we already stressed earlier, if we are not going to be able to farther exploit our subspace structure (i.e. through strong constraints) with a combined approach, than there is really no point to bringing tracking and segmentation into a common framework.

An alternative to combining feature tracking and global dimension energies would be to literally alternate between GDM (to maintain a segmentation hypothesis) and feature tracking where we track the features associated with each object using a strict subspace constraint. The problem with this approach might be referred to as the "self-fulfilling prophecy". If our segmentation algorithm assigns a feature to an object cluster, then from that point on future tracking stages force the feature to move with that object. This, in turn, makes the feature look even more like it belongs to that object because its motion is consistent with the other features on the object. In other words, we can't rely on the trajectories to reveal object affiliation when we are forcing trajectories to

respect the motion constraints of their assigned objects. Analyzing these straightforward approaches shows us that combining the tracking and segmentation problems is more complicated and subtle that it might initially seem.

Developing a complete, working joint tracking and segmentation framework is beyond the scope of my work here, and it is unknown whether such a framework could yield measurable performance improvements over performing each task separately, and if so whether those improvements are worth the reduction in modularity that must be incurred. However, there are some things we can learn about what such a framework might look like from the projects presented here. The last section is devoted to this discussion.

5.1 On Developing A Joint Tracking and Segmentation Framework

In the problem of feature-based motion segmentation, one either needs to know or estimate the number of objects in a scene. In the interest of focusing on one specific problem at a time and for comparing different techniques specifically at the task of segmentation, this is often taken out of the equation and treated as a known constant. This is the correct approach. If one solution works better or worse than another, it is important to identify what aspect of the solution is responsible for that success or failure. This is accomplished by comparing different methods on the smallest problems possible, keeping all outside variables (like the number of objects) constant. This gets more challenging, however, when working on a framework for joint tracking and segmentation. In this setting, if we get the number of objects in the scene wrong or we segment them incorrectly, it may affect our ability to track the very features we are trying to segment. This seems to require either:

- Including object counting in the tracking/segmentation framework, or
- Tracking each feature in an unconstrained manner, in addition to using whatever constraints we impose, to allow for a separate algorithm to determine the number of objects from the unconstrained trajectories.

If the framework is able to recover from an incorrect estimate of the number of objects in the scene, then the question of determining the number of objects can be reduced to two simpler questions:

- Do I have too many objects?
- Do I have too few objects?

You can then start under the assumption that the entire scene belongs to one object and make adjustments to this hypothesis as tracking progresses and it becomes apparent that our estimate is in error. After an initial "spin up", the system would zero in on the correct number of objects. It is important though that these questions be answered in a consistent manner, or it is possible to get oscillations in your estimate of the number of objects. Also, when we do decide that we have too few objects, we need to have an intelligent way of introducing a new one and updating our segmentation hypothesis accordingly. Similarly, if we decide that we have too many objects, we need to be able to collapse multiple objects into one. These are both non-trivial problems in their own right. For instance, we can just naively re-segment the scene from scratch whenever we change the number of objects, but this would ignore what we have previously learned about the scenes segmentation. Also, it would not exploit our knowledge of how objects tend to come and go in a scene. For instance, if we have one too many objects, it is likely because one object we were following left the field of view - in this case we should try to eliminate one of the existing objects and leave the segmentation of the others intact. Segmenting the scene from scratch would not necessarily have this effect.

When we think about the issues surrounding object counting it becomes apparent that whether or not we build in a mechanism for estimating the number of objects, tracking features without constraints is really a necessity. Imagine that we are correctly tracking and segmenting features in a scene when we decide to initialize a new feature. How do we determine which object the feature belongs to? How do we know it belongs to an existing object at all? Perhaps it is the first feature on a new object. Making this decision purely based on geometry or appearance would be to ignore one of the most useful clues we have available... how the feature moves. However, to see how the feature moves before it is assigned to an object we must be tracking it without constraints. We might think of a system that tracks only new features without constraints. Then, once

a feature has been assigned to an object we resign to tracking with the associated constraint. However, it is clear that such a system throws away information. For instance, it is always possible that a mistake is made in the initial assignment of a feature. If we stop tracking the feature without constraint, how do we ever revisit the question of the features assignment? The motion of the feature will respect the motion of its assigned object because we *force* it to. The only clue that we made a mistake is the comparatively poor image-template fit residual for the feature, which is a difficult value to rely on since it is effected by many other factors, like the brightness of the feature.

The above argument suggests that multi-hypothesis tracking may be the best approach. Each feature is tracked independently, without constraint, and then also tracked with a constraint associated with some object in the scene. This can be taken farther still. We could actually track each feature with an array of different hypotheses. We could have one hypothesis per object in the scene, where we track the feature as if it belongs to the object at hand. The final hypothesis would always be that the feature doesn't belong to any existing object and it is tracked without constraint.

A full multi-hypothesis tracking approach introduces some additional questions that need to be answered:

1. In a single-hypothesis tracker where each feature is assigned to a single object, an object can be characterized by the set of features that belong to it. In a multi-hypothesis tracker, where each feature hypothetically belongs to each object, how do we characterize an object? In other words, what sets the different objects apart?

2. Do we update feature templates for a feature separately for each hypothesis, or do we maintain one template for each feature?

To address the first question, we need to clarify what state we are maintaining in a multi-hypothesis tracker. For each feature in the scene, we maintain trajectories for each hypothetical assignment of the feature to the different objects in the scene. We also maintain one or more template images to characterize the feature. In addition, we must keep track of which object we think the feature belongs to, i.e. which hypothesis is preferred. This may be done with either a hard assignment to a single object (which we

re-evaluate periodically), or with a soft assignment vector that we update each frame. So even though each feature is hypothetically assigned to each object, each feature also has a preferred object to which we think it most likely belongs. Thus, to answer the first question, we can characterize an object by the set of features which we think most likely belong to it. This collection of features must be used to determine the model for the objects motion. This is what sets the different objects in the scene apart from one another.

The second question is difficult to address. From one viewpoint, each hypothesis should be treated separately, since they correspond to completely different assumptions about which object a feature belongs to. This would mean that if the positions of a feature under different hypotheses start to diverge, then when it comes time to update the template image for the feature, we may have two different (potentially very different) candidate images with which to update it. It would seem that we should maintain separate templates images for each hypothesis, and simply update them separately. This has the unnerving consequence that the templates characterizing a single feature can diverge from one another. Since it is but a single feature, we know that if two candidate templates are significantly different from one another, then at least one of them is wrong and doesn't accurately reflect the appearance of the feature anymore. So the counter-argument is that if we allow template images to diverge, then the hypotheses with inaccurate templates have no chance of correctly tracking the feature anymore. This viewpoint would suggest that we use a single template image to characterize each feature. One possible approach may be to create candidate template updates from each hypothesis, and select the one that most closely matches the original or most recent template for a feature.

To conclude, we can make a few observations from this discussion. First, any combined feature tracking and segmentation framework should use strong constraints on feature trajectories during tracking in order to outperform a good combination of independent feature tracking and trajectory segmentation algorithms. Next, it is important that features also be tracked without any constraints. It is not clear whether a full multi-hypothesis framework is needed or if just two hypotheses will do[2] , but if features are only tracked with constraints, then it is very difficult to use feature motion as an

[2] The two hypotheses being assignment to a single object, and unconstrained/no object affiliation

input for scene segmentation since the feature trajectories are already biased by a presupposed segmentation hypothesis. Unconstrained trajectories can also support object counting, which is likely something that needs to be considered part of the joint tracking and segmentation problem. We have also discussed how one might derive motion models in a multi-hypothesis tracking framework, and we have explored the problem of template updates in such a framework. Whether a combined feature tracking and segmentation framework can outperform a conventional track-then-segment pipeline, and whether a less modular solution would ever be embraced in real-world scenarios is yet to be seen. However, we have shown through the projects presented in this thesis that low-rank trajectory structure can be used as a powerful tool in both the tracking and segmentation problems and we have (hopefully convincingly) argued that there is still room to farther exploit this structure through a combined framework. We have developed some tools which may aid in this undertaking, and we hope that they may steer anyone pursuing this goal in the right direction.

References

[1] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(1):221 – 255, March 2004.

[2] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2000.

[3] Y. Ma. *An invitation to 3-D vision: from images to geometric models*. Interdisciplinary applied mathematics: Imaging, vision, and graphics. Springer, 2004.

[4] X. Feng and P. Perona. Scene segmentation from 3d motion. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 225 –231, jun 1998.

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

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

[7] R. Shankar, A. Yang, S. Sastry, and Y. Ma. Robust algebraic segmentation of mixed rigid-body and planar motions from two views. *Int. J. Comput. Vision*, 88(3):425–446, 2010.

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

[9] R. Vidal. Subspace clustering. *Signal Processing Magazine, IEEE*, 28(2):52 –68, march 2011.

[10] E. Arias-Castro, G. Chen, and G. Lerman. Spectral clustering based on local linear approximations. *Electron. J. Statist.*, 5:1537–1587, 2011.

[11] G. Chen, S. Atev, and G. Lerman. Kernel spectral curvature clustering (KSCC). In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on Computer Vision*, pages 765–772, Kyoto, Japan, 2009.

[12] G. Chen and G. Lerman. Spectral curvature clustering (SCC). *IJCV*, 81(3):317–330, 2009.

[13] T. Zhang, A. Szlam, Y. Wang, and G. Lerman. Hybrid linear modeling via local best-fit flats. *International Journal of Computer Vision*, 100:217–240, 2012.

[14] G. Chen and M. Maggioni. Multiscale geometric and spectral analysis of plane arrangements. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[15] O. Roy and M. Vetterli. The Effective Rank: A Measure of Effective Dimensionality. In *European Signal Processing Conference (EUSIPCO)*, pages 606–610, 2007.

[16] R. Vershynin. Introduction to the non-asymptotic analysis of random matrices. In Y. C. Eldar and G. Kutyniok, editors, *Compressed Sensing: Theory and Applications*. Cambridge Univ Press, to appear.

[17] D. Bertsekas. *Nonlinear programming*. Optimization and neural computation series. Athena Scientific, 1995.

[18] Y. Ma, H. Derksen, W. Hong, and J. Wright. Segmentation of multivariate mixed data via lossy coding and compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1546–1562, September 2007.

[19] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 09)*, pages 2790 – 2797, 2009.

[20] G. Liu, Z. Lin, and Y. Yu. Robust subspace segmentation by low-rank representation. In *Proc. of the 2010 International Conference on Machine Learning*, 2010.

[21] R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –8, june 2007.

[22] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma. Robust recovery of subspace structures by low-rank representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):171 –184, jan. 2013.

[23] L. Grafakos. *Classical and modern Fourier analysis*. Pearson/Prentice Hall, 2004.

[24] R. Bhatia. *Matrix*. Graduate Texts in Mathematics Series. Springer Verlag, 1997.

[25] T. Papadopoulo and M. Lourakis. Estimating the jacobian of the singular value decomposition: Theory and applications. In *In Proc. European Conf. on Computer Vision, ECCV 00*, pages 554–570. Springer, 2000.

[26] B. D Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679, 1981.

[27] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, 1991.

[28] J. Shi and C. Tomasi. Good features to track. In *CVPR*, pages 593–600. IEEE, 1994.

[29] M. Brand and R. Bhotika. Flexible flow for 3d nonrigid tracking and shape recovery. In *CVPR*, 2001.

[30] L. Torresani, D.B. Yang, E.J. Alexander, and C. Bregler. Tracking and modeling non-rigid objects with rank constraints. In *CVPR*, volume 1, pages I–493–I–500 vol.1, 2001.

[31] R. Hartley and R. Vidal. Perspective nonrigid shape and motion recovery. In *ECCV (1)*, pages 276–289, 2008.

[32] Y. Dai, H. Li, and M. He. A simple prior-free method for non-rigid structure-from-motion factorization. In *CVPR*, pages 2018–2025, 2012.

[33] A. Buchanan and A. Fitzgibbon. Combining local and global motion models for feature point tracking. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[34] S. Ricco and C. Tomasi. Dense lagrangian motion estimation with occlusions. *CVPR 2012*, 0:1800–1807, 2012.

[35] R. Garg, A. Roussos, and L. Agapito. A variational approach to video registration with subspace constraints. *IJCV*, 104(3):286–314, 2013.

[36] M. Irani. Multi-frame correspondence estimation using subspace constraints. *IJCV*, 48(3):173–194, 2002.

[37] M. Brand. Subspace mappings for image sequences. In *Proc. Workshop Statistical Methods in Video Processing*, 2002.

[38] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *ECCV*, volume 2350, pages 707–720. 2002.

[39] M. Brand. Morphable 3d models from video. In *CVPR*, volume 2, pages II–456–II–463, 2001.

[40] R. Garg, L. Pizarro, D. Rueckert, and L. de Agapito. Dense multi-frame optic flow for non-rigid objects using subspace constraints. In *ACCV*, pages 460–473. 2010.

[41] L. Torresani and C. Bregler. Space-time tracking. In *ECCV (1)*, pages 801–812, 2002.

[42] Lorenzo Torresani, Aaron Hertzmann, and Christoph Bregler. Robust model-free tracking of non-rigid shape. Technical report, Technical Report TR2003-840, New York University, 2003.

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

[44] E. Elhamifar and R. Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Trans. PAMI*, PP(99):1–1, 2013.

[45] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9:717–772, 2009.

[46] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *J. ACM*, 58(3):11, 2011.

[47] S. Negahban, P. D. Ravikumar, M. J. Wainwright, and B. Yu. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Stat. Science*, 27(4):538–557, 2012.

[48] Bryan Poling and Gilad Lerman. A new approach to two-view motion segmentation using global dimension minimization. *International Journal of Computer Vision*, 108(3):165–185, 2014.

[49] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV*, volume 588, pages 237–252. 1992.

[50] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE TPAMI*, 33(3):500–513, 2011.

[51] Pan Ji, Hongdong Li, Mathieu Salzmann, and Yiran Zhong. Robust multi-body feature tracker: A segmentation-free approach. *CoRR*, abs/1603.00110, 2016.

[52] D. Sachs, S. Nasiri, and D. Goehl. Image stabilization technology overview. `http://www.invensense.com/jp/mems/gyro/documents/whitepapers/ImageStabilizationWhitepaper_051606.pdf`.

[53] Neel Joshi, Sing Bing Kang, C Lawrence Zitnick, and Richard Szeliski. Image deblurring using inertial measurement sensors. *ACM Transactions on Graphics (TOG)*, 29(4):30, 2010.

[54] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. Technical report, CSTR 2011-03, Stanford University Computer Science, 2011.

[55] S. You, U. Neumann, and R. Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 260–267, 1999.

[56] M. Hwangbo, J.-S. Kim, and T. Kanade. Gyro-aided feature tracking for a moving camera: fusion, auto-calibration and gpu implementation. *The International Journal of Robotics Research*, 30(14):1755–1774, 2011.

[57] M. Pachter and G. Mutlu. The navigation potential of ground feature tracking for aircraft navigation. In *American Control Conference (ACC), 2010*, pages 3975–3979, 2010.

[58] A.I. Mourikis, N. Trawny, S.I. Roumeliotis, A.E. Johnson, A. Ansar, and L. Matthies. Vision-aided inertial navigation for spacecraft entry, descent, and landing. *Robotics, IEEE Transactions on*, 25(2):264–280, 2009.

[59] S.I. Roumeliotis, A.E. Johnson, and J.F. Montgomery. Augmenting inertial navigation with image-based motion estimation. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 4326–4333, 2002.

[60] Eagle S Jones and Stefano Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*, 30(4):407–430, 2011.

[61] J.D. Hol, T.B. Schön, H. Luinge, P.J. Slycke, and F. Gustafsson. Robust real-time tracking by fusing measurements from inertial and vision sensors. *Journal of Real-Time Image Processing*, 2(2-3):149–160, 2007.

[62] M.J. Veth. *Fusion of Imaging and Inertial Sensors for Navigation*. Air Force Institute of Technology, 2006. `http://books.google.co.il/books?id=Ak4zOAAACAAJ`.

[63] M. Veth and J. Raquet. Fusion of low-cost imaging and inertial sensors for navigation. In *ION GPS GNSS -CD ROM EDITION-; 19th International technical meeting, Institute of Navigation*, pages 1093–1103, 2006.

[64] J.R. Gray. *Deeply-integrated Feature Tracking for Embedded Navigation*. Air Force Institute of Technology., 2009. `http://books.google.co.il/books?id=zITBXwAACAAJ`.

[65] D.D. Diel, P. DeBitetto, and S. Teller. Epipolar constraints for vision-aided inertial navigation. In *Application of Computer Vision, 2005. WACV/MOTIONS'05 Volume 1. Seventh IEEE Workshops on*, volume 2, pages 221–228, 2005.

[66] J.-Y. Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm, 2001. `http://robots.stanford.edu/cs223b04/algo_affine_tracking.pdf`.

[67] B. Poling, G. Lerman, and A. Szlam. Better feature tracking through subspace constraints. In *Computer Vision and Pattern Recognition, 2014. CVPR 2014. IEEE Conference on*, 2014.

[68] V. Chi. Quaternions and rotations in 3-space, 1998. `http://gamma.cs.unc.edu/courses/planning-f07/PAPERS/quaternions.pdf`.