A Brute-Force Algorithm for Reconstructing a Scene from Two Projections

Olof Enqvist Fangyuan Jiang Fredrik Kahl
Centre for Mathematical Sciences, Lund University, Sweden
{olofe,fangyuan,fredrik}@maths.lth.se

Abstract

Is the real problem in finding the relative orientation of two viewpoints the correspondence problem? We argue that this is only one difficulty. Even with known correspondences, popular methods like the eight point algorithm and minimal solvers may break down due to planar scenes or small relative motions. In this paper, we derive a simple, brute-force algorithm which is both robust to outliers and has no such algorithmic degeneracies. Several cost functions are explored including maximizing the consensus set and robust norms like truncated least-squares.

Our method is based on parameter search in a four-dimensional space using a new epipolar parametrization. In principle, we do an exhaustive search of parameter space, but the computations are very simple and easily parallelizable, resulting in an efficient method. Further speed-ups can be obtained by restricting the domain of possible motions to, for example, planar motions or small rotations. Experimental results are given for a variety of scenarios including scenes with a large portion of outliers. Further, we apply our algorithm to 3D motion segmentation where we outperform state-of-the-art on the well-known Hopkins-155 benchmark database.[1]

1. Introduction

Already in 1981, Longuet-Higgins suggested a simple and yet elegant solution to the problem of finding the relative orientation of two viewpoints [10]. The algorithm, known as the eight point algorithm, still plays a major role in computer vision [8]. However, the algorithm suffers from many degeneracies, e.g., if the scene is planar then the algorithm fails. Perhaps even more serious is that the algorithm assumes that the correspondence problem is already solved. Therefore, more robust approaches have been developed to cope with outliers. Here RANSAC methods using minimal solvers are considered to be state-of-the-art [12, 3]. Still, the problem of algorithmic degeneracies remains for minimal solvers.

[1]This work was supported by the European Research Council (grant 209480) and the Swedish Foundation for Strategic Research.

Another problem that has been recognized by several researchers is the importance of optimizing a suitable cost function, where costs based on reprojection errors are preferable to algebraic errors [8]. Bundle adjustment does optimize the statistically correct criterion (given that measurement errors are independent and normally distributed), but the method is sensitive to initialization. Therefore global optimization algorithms have been developed [9, 7, 5, 2] which are not susceptible to local minima.

Looking back at 30 years of algorithmic development since the eight point algorithm [10], a set of criteria has emerged that we believe a good relative orientation algorithm should possess:

(i) Robustness to outliers,
(ii) No algorithmic degeneracies,
(iii) Cost function based on reprojection errors,
(iv) Not dependent on a good initialization,
(v) Practical.

For example, RANSAC is designed to fulfill *(i)*, requires no initialization *(iv)* and has been successfully applied in many real systems *(v)*, but the method does not meet objectives *(ii)* and *(iii)*. Similarly, recent global relative orientation methods [7, 5] do meet criteria *(ii)*-*(iv)*, but cannot be considered to be practical *(v)* since the running times are in some cases extreme (order of minutes). In practice, a heuristic combination of different algorithms is often used to overcome the difficulties in fulfilling these objectives. For example, homographies are often used to detect if the scene is planar or if the motion is a pure rotation. Another example of this phenomena is that particular motions have been examined separately [15].

In this paper, an algorithm using exhaustive search is developed and it fulfills *(i)*-*(iv)* by design. For example, provided that the discretization of the parameter space is fine enough, the method is guaranteed to find the globally optimal solution. The key idea in order to make it practical is that the expensive computations are done in lower dimensions, and only very simple calculations are required in the high-dimensional search. The ultimate proof is of course by showing that it works on real experiments - this is done in the experimental section. In particular, when applied to

3D motion segmentation, our approach significantly outperforms state-of-the-art methods on 104 video sequences in the Hopkins 155 database [14]. Note that the database contains a variety of relative motions and scenes that are considered to be degenerate for several of the above standard algorithms.

2. Preliminaries

Consider two views of a scene. Let x denote an image point - represented by a unit vector - in the first view and x' the corresponding image point in the second view. The assumption is that these are both the projection of some 3D point X. If we choose a global coordinate system such that the first camera lies at the origin and the second camera at $t = (0, 0, 1)^T$, we get

$$\lambda Rx = X, \quad \lambda' R'x' = X - t, \tag{1}$$

where R and R' are 3×3 rotation matrices and λ and λ' positive reals. Note that λ and λ' are distances rather than depths. To simplify the derivations, we will assume that there are no points at infinity, although the method works just as well with points at infinity.

Theorem 1. *Let R and R' be rotation matrices with row vectors r_1, r_2, r_3 and r'_1, r'_2, r'_3, respectively and x and x' corresponding points. Then,*

$$(r_1 x, r_2 x) = k(r'_1 x', r'_2 x') \text{ with } k > 0, \tag{2}$$

$$r_3 x > r'_3 x', \tag{3}$$

if and only if there exists a 3D point X satisfying (1)*.*

Proof. Clearly

$$k = \frac{||(r_1 x, r_2 x)||}{||(r'_1 x', r'_2 x')||} = \sqrt{\frac{1 - (r_3 x)^2}{1 - (r'_3 x')^2}} < 1. \tag{4}$$

Let λ be the solution to

$$\lambda r_3 x - 1 = \lambda k r'_3 x', \tag{5}$$

and put $\lambda' = \lambda k$. From $k < 1$ and $r_3 x > r'_3 x'$ it is straightforward to show that $\lambda > 0$ and hence $\lambda' > 0$. Now let $X = \lambda Rx$. To see that (1) is satisfied, consider

$$X - t = \lambda Rx - t = \begin{pmatrix} \lambda r_1 x \\ \lambda r_2 x \\ \lambda r_3 x - 1 \end{pmatrix} \tag{6}$$

but by (2) and (5) this is equal to

$$\begin{pmatrix} \lambda k r'_1 x' & \lambda k r'_2 x' & \lambda k r'_3 x' \end{pmatrix}^T = \lambda' R' x'. \tag{7}$$

This proves the *if* part and *only if* follows easily from (1). □

The description gets even simpler if we switch to spherical coordinates,

$$Rx = \begin{pmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{pmatrix}, \; R'x' = \begin{pmatrix} \sin\theta'\cos\varphi' \\ \sin\theta'\sin\varphi' \\ \cos\theta' \end{pmatrix}. \tag{8}$$

Now the necessary and sufficient constraints are

$$\varphi = \varphi' \text{ and } \theta < \theta'. \tag{9}$$

Remark 1. *In this article angles are considered equal if they are equal modulo 2π but to simplify the presentation this is not always written explicitly. For example $\xi \in [\alpha, \beta]$ if $\xi + 2\pi k$ does for some $k \in \mathbb{Z}$.*

The next step is to allow measurement errors. We say that corresponding points x and x' are consistent with a relative orientation if the reprojection errors are less than some prescribed threshold, ϵ. In this article we will constrain the angular reprojection errors, i.e.

$$\angle(Rx, X) < \epsilon \quad \text{and} \quad \angle(R'x', X - t) < \epsilon. \tag{10}$$

Theorem 2. *Consider rotation matrices R and R' and spherical coordinates as defined in* (8)*. Further define w in the following way. For $\theta < \theta'$,*

$$w = \arcsin(\sin\epsilon / \sin\theta) + \arcsin(\sin\epsilon / \sin\theta'), \tag{11}$$

if this is defined and otherwise $w = \pi$. For $\theta' < \theta < \theta' + 2\epsilon$

$$w = \arccos\left(\frac{\cos 2\epsilon - \cos\theta\cos\theta'}{\sin\theta\sin\theta'}\right), \tag{12}$$

if this is defined and otherwise π. Then,

$$\begin{aligned} &\theta < \theta' + 2\epsilon \\ &\varphi \in [\varphi' - w, \varphi' + w], \end{aligned} \tag{13}$$

if and only if the angular reprojection errors are less than ϵ.

Proof. See Appendix A. □

3. A Search Algorithm

The proposed method for estimating relative orientation is to search for rotation matrices R and R' satisfying Theorem 2 for as many point correspondences as possible.

Naturally, using two rotation matrices to represent a relative orientation is an overparameterization. In fact if S is a rotation about the z-axis, then (R, R') and (SR, SR') describe the same relative orientation. We will soon see how to avoid this ambiguity, but first we need a method to expand a unit vector to a rotation matrix. More precisely, let

Γ be a function that maps a given unit vector r to a rotation matrix, Γ_r, having r as its third row. Then any relative orientation can be written as

$$R = S_\alpha \Gamma_r \text{ and } R' = \Gamma_{r'} \tag{14}$$

where S_α is a rotation by α about the z-axis. Hence the set of parameters consists of two unit vectors and an angle α. Now consider the spherical coordinates in (8). Only φ depends on α. Let $\varphi(r)$ denote the value if $\alpha = 0$. This changes the last constraint of Theorem 2 to

$$\varphi(r) + \alpha \in [\varphi'(r') - w, \varphi'(r') + w], \tag{15}$$

which is easily translated to a constraint $\alpha \in [\alpha_{lo}, \alpha_{up}]$, where

$$\begin{aligned} \alpha_{lo} &= \varphi'(r') - \varphi(r) - w \\ \alpha_{up} &= \varphi'(r') - \varphi(r) + w. \end{aligned} \tag{16}$$

Each pair of corresponding points yields such an interval and by sorting these lower and upper bounds for all correspondences, one can find the interval having the maximal number of inliers; see Algorithm 2.

Algorithm 1. ***Brute-Force Search***

For a given level of discretization and error threshold ϵ, a relative orientation having the maximal number of inliers $n_{\max}$ is computed.

Compute a discretization, $\mathcal{D}$ of S^2.
For each $r \in \mathcal{D}$
  For each x
    Compute $\varphi(r)$, $\theta(r)$ and $v(r)$.
  For each x'
    Compute $\varphi'(r)$, $\theta'(r)$ and $v'(r)$.
Put $n_{max} = 0$
For each pair $(r, r') \in \mathcal{D} \times \mathcal{D}$
  For each correspondence (x, x')
    If $\theta'(r') + \epsilon > \theta(r) - \epsilon$
      Compute $w = v(r) + v'(r')$.
      A lower bound $\alpha_{lo} = \varphi'(r') - \varphi(r) - w$.
      An upper bound $\alpha_{up} = \varphi'(r') - \varphi(r) + w$.
  Find the max intersection n, using Algorithm 2.
  If $n > n_{\max}$
    Store the current parameters.
    Set $n_{\max} = n$.

Now we are ready to look at the complete algorithm; see Algorithm 1. The idea is to perform a search for vectors r and r'. Since both vectors have unit length they lie in S^2, being the unit sphere in $\mathbb{R}^3$, and the search space is a discretized version of $S^2 \times S^2$. For each pair (r, r') we compute the lower and upper bounds on α which are given in (16). We sort these bounds and compute the maximal number of inliers. Hence the complexity of the algorithm is $O(k^2 m \log(m))$ where k is the number of points in the discretization of S^2 and m is the number of correspondences.

Remark 2. *As it matters only rarely and complicates the description, we ignore case 2 in the computation of w. Thus w can always be divided into*

$$w = v + v' \tag{17}$$

where v does not depend on x' and v' does not depend on x.

Algorithm 2. ***Maximal Intersection***

Given lower bounds $\mathcal{L}$ and upper bounds $\mathcal{U}$, a point is found that lies in as many intervals as possible. Outputs the number of intersecting intervals, n and the point.

Sort $\mathcal{L}$ and $\mathcal{U}$.
Initialize $j = 1$ and $n = 0$.
For $i \in \{1, \ldots, |\mathcal{L}|\}$
  While $\mathcal{U}_j < \mathcal{L}_i$
    Increase $j = j + 1$.
  If $i - j > n$
    Store $\mathcal{L}_i$.
    Set $n = i - j$.

4. Other Cost Functions

So far we have simply counted the number of inliers to assess the quality of a relative orientation. Inliers are correspondences with the reprojection errors less than some prescribed threshold, ϵ. This approach is simple and generally yields good results, but it does have its limitations; see [8]. One problem is that the method might be sensitive to the choice of ϵ, but also that the distribution of the inlier errors are not considered. In [1] a more refined cost function is proposed. The assumption is that correct matchings have a clock-shaped error distribution similar to the Gaussian distribution, whereas incorrect matchings have approximately uniformly distributed errors. These assumptions lead to the cost function

$$C(d) = -\log\left(c + \exp\left(-d^2\right)\right) \tag{18}$$

where d is the reprojection error; see Figure 1. In the same book it is noted that a good approximation of this cost function can be obtained by truncating the ordinary squared error. A cost function of this kind cannot be handled directly by the proposed method, but one can approximate the function to arbitrary precision. An example of such an approximation is shown Figure 1. As the reprojection error increases it changes value three times. This means that when computing w in Algorithm 1, we should do so for

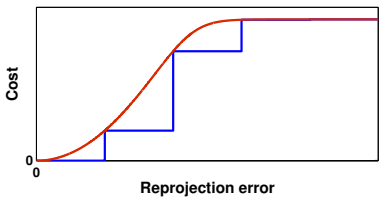

Figure 1. The robust cost function (red) suggested in [1], and a piecewise constant approximation of it (blue) which can be optimized using the proposed framework.

three thresholds $\epsilon_1, \epsilon_2, \epsilon_3$. Consequently each correspondence will yield three intervals $I_1 \subset I_2 \subset I_3$ - one for each time the value of the cost function changes. The different types of intervals also get a weight indicating how much the cost function changes when entering this interval.

In Algorithm 2 we get three lists of lower bounds $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ and similarly for the upper bounds. The different lists are sorted separately and then gone through like before. Passing a lower bound from $\mathcal{L}_i$, weight w_i is subtracted from the current cost, and passing an upper bound from $\mathcal{U}_i$ the same weight is added. The computational cost will be approximately linear in the number of steps of the cost function.

5. Restricted Motions

One advantage of the suggested approach to relative orientation estimation, is that restricted motions can be handled easily. In this section we present a few standard restrictions and discuss how they can be enforced.

Planar motion. If the rotation axis is known and perpendicular to the translation, this can be used in the following way. Let f be the rotation axis. We get the following constraints,

$$r_3 f = 0, \quad r_3' f = 0 \quad \text{and} \quad \alpha = 0 \tag{19}$$

The first two constraints can easily be enforced in the discretization step. Only epipoles in these planes are generated. The third constraint reduces the set of angles that has to be considered in Algorithm 2.

Small motion. In tracking applications, the motion between consecutive frames is generally small. This can easily be enforced by adding constraints

$$\angle(r_3, r_3') < \gamma_{max} \quad \text{and} \quad \alpha < \alpha_{max}. \tag{20}$$

These constraints reduce the number of pairs that have to be considered in Algorithm 1.

6. Motion Segmentation

To examine how well the brute-force algorithm works in practice, it was tried in a simple system for motion segmentation. Given a sequence of images of multiple moving objects, the aim of motion segmentation is to estimate all these motions as well as the motion of the camera. Moreover, each detected feature point should be classified as belonging to one motion.

Algorithm 1 Multiple Motions

Given two views A and B with multiple moving objects and point tracks $\mathcal{T}$, N hypothetical motions are estimated. An extra view C is used to validate motions.

Repeat N times
  Set $\mathcal{H} = \mathcal{T}$.
  For the view pairs (A, B), (A, C), (B, C)
    Estimate relative orientation using $\mathcal{H}$ and threshold ϵ_1.
    Remove tracks with error larger than ϵ_2 from $\mathcal{H}$.
  Reestimate a relative orientation between A and B using $\mathcal{H}$ and threshold ϵ_1. Store this solution.
  Set $\mathcal{T} = \mathcal{T} \setminus \mathcal{H}$.

Like much of the work in this field, we assume that the number of motions is known. For the discussion let us assume that this number is three. A seemingly straightforward approach to segmentation would be to keep track of the top three motions in our brute-force search, but this turns out be difficult in practice. The peaks in the relative orientation space are rather flat and it is hard to distinguish different motions.

Therefore, we proceed in a sequential manner using Algorithm 3. The first step is to estimate N hypothetical motions. This is done in a sequential manner, using Algorithm 1. Typically, N is chosen significantly larger than the true number of motions not to miss any motion. The next step is to choose three of these N motions to perform the motion segmentation. We do this by going through all possible choices of three motions and choosing choosing the ones that yield the lowest number of outliers. Just as in Algorithm 1 outliers are tracks having an error larger than ϵ_2. Having decided on three motions we match each point track to that motion which yields the smallest errors.

The classification obtained in this manner can be refined by standard bundle adjustment. Details are given in the experimental section.

6.1. Adding a Spatial Prior

To further improve the motion segmentation results, we tried using a spatial prior assuming that close points probably belong to the same motion. We formulate the spatial prior in an energy minimization framework with a data term

and a smoothness term,

$$C(\mathbf{x}) = \sum_{p \in \mathcal{V}} C_p(x_p) + \lambda \sum_{(p,q) \in E} C_{pq}(x_p, x_q). \qquad (21)$$

Here $G = (V, E)$ is an undirected graph. The set of nodes $\mathcal{V}$ corresponds to the point tracks and x_p denotes the label of node p. The edges E describes the neighborhood relationship. We use the reprojection error of point p as data term $C_p(x_p)$ and define the smoothness term as,

$$C_{pq}(x_p, x_q) = \begin{cases} 0 & \text{if } x_p = x_q \\ \frac{d_{max} - d(p,q)}{d_{max}} & \text{if } x_p \neq x_q \end{cases} \qquad (22)$$

where $d(p, q)$ denotes the Euclidean distance of point p and q and d_{max} is a threshold to define the size of the neighborhood. If $d(p, q) < d_{max}$, then $(p, q) \in E$. This smoothness term will penalize the case when two points lie close to each other but belong to different motions. The constant λ determines the balance between the data and smoothness term. Energy minimization was performed using α-expansions; see [**?**].

7. Parallel Implementation

Normally, the weakness of a brute-force algorithm is its computational performance. However, studying Algorithm 1 we note that the computations for different pairs (r, r') are independent, so we can easily parallelize the algorithm using a MapReduce model. In the Map step, the lower and upper bounds are sorted simultaneously and then intersections are computed simultaneously for all pairs (r, r'). In the Reduce step, the pair (r, r') that yields most inliers is picked by reduction operations.

Nvidia's parallel computing architecture, CUDA, was used for the parallel implementation. Algorithm 2, was implemented in a 2-dimensional grid with k by k blocks, where k is again the number of points in the discretization. Each block executes the computation for one pair of epipoles, (r, r'). Inside each block, a parallel bitonic sorting algorithm with complexity $O(n \log(n)^2)$ is implemented since it is well-suited for sorting within a block using shared memory. To find the maximum intersection, each thread goes through the upper bound list to find the maximal intersection for the current lower bound. This is done using binary search.

In the end, the parallel implementation is up to 30 times faster than the serial implementation, making the performance of our algorithm quite practical. To make sure global memory access coalescing, we pad the lower and upper bounds with dummy values. Constant memory is used to store the epipoles during the computation of spherical coordinates. This works to reduce global memory latency.

8. Experiments

For the testing we primarily used the GPU implementation. Timings are for 3GHz Core2 Duo with 8GB Memory with an NVidia Tesla 2050 with 3GB global memory.

To get some data on the execution times, synthetic data was generated. First 100 random 3D point were generated in a cube centered at the origin, having side 300. The cameras were placed randomly at distance of approximately 1000. Gaussian noise with standard deviation 0.0002 was added to the image points. Figure 2 shows angular errors in rotation and translation when compared to the ground truth. The threshold $\epsilon = 0.005$ was used with different degrees of discretization.

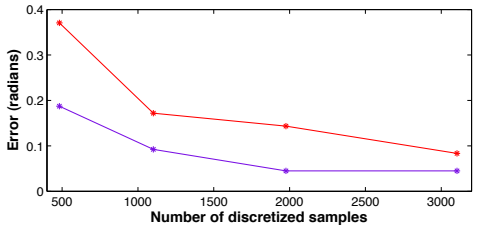

Figure 2. The plot shows errors for different discretizations. The error in rotation is shown in red and the error in translation is shown in blue.

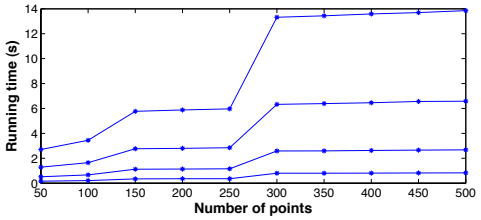

Figure 3. Execution times for different discretizations. Starting from below the curves were generated using 700, 1258, 1976 and 2862 points in the discretization of the unit sphere, S^2.

8.1. Other Cost Functions

To verify the possibility of using other cost functions we tried it on some random data generated as described above. Using the appoximated truncated L_2 norm in the way described in Section 4 the rotational error decreased from the average 0.17 radians to an average of 0.11 radians. This was using 1100 points in the discretization. The threshold for the standard method was $\epsilon = 0.005$ and the thresholds for the approximate truncated L_2 was set to $\epsilon/2$, ϵ, $3\epsilon/2$ and 2ϵ.

8.2. Outliers

To test the proposed algorithm on data with a lot of outliers, synthetic data was generated in the following way.

First 50 random 3D point were generated in a cube centered at the origin, having side 100. The cameras were placed randomly at distance of approximately 1000. Then 450 outliers were added to each image. They were generated in the same way as the inliers but separately for the two images. Gaussian noise with standard deviation 0.0002 was added to the image points. Figure 4 shows how many of the 50 inliers were found by the proposed algorithm. The threshold $\epsilon = 0.0005$ was used in the algorithm and the average computation time for the parallel implementation was 6s.

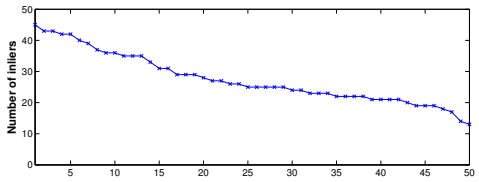

Figure 4. The number of inliers for the 50 outlier experiments. The list was sorted for better visualization. In each example there were 50 inliers and 450 outliers. The error threshold was $\epsilon = 0.0005$ and 1976 points were used in the discretization of the sphere.

The outlier rate in these experiments was 90%. This means that using standard RANSAC and a five-point solver, the expected number of iterations before picking just one single set with 5 inliers is 100 000 and using reprojection errors that also means performing 50 million triangulations.

8.3. Planar Motion

The performance on planar scenes was tested on 64 image pairs from eniro.se. These are street-view images taken from a car so the motion is approximately planar. Since the images are given with direction information we could compute the deviation between the estimated rotation matrix and the ground truth. This deviation in radians is given in Figure 5. The results were produced using 100 points to discretize the unit circle and a threshold of 0.0005. The average execution time was 0.47 s for a sequential java implementation.

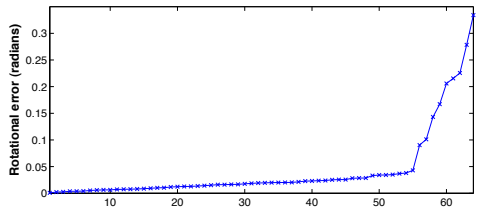

Figure 5. Angular error when comparing with the ground truth rotation.

8.4. Motion Segmentation

We will now look at the performance of this 3D motion segmentation algorithm for rigid scenes from the Hopkins 155 database [14]. Current state-of-the-art results are reported in [4] and all the top performers are included in the comparison below. In each sequence, there are typically 20-30 frames and a few hundred 2D feature tracks given. The number of motions in each sequence is also specified.

Some of the sequences contain articulated motions to which the presented framework does not apply. Therefore we focus on the subset of checkerboard sequences, 26 sequences with 3 motions and 78 sequences with 2 motions, hence 104 out of the 155 sequences are considered. Based on [4], one can conclude that the checkerboard sequences are the most difficult ones as the classification errors are significantly lower for the remaining ones.

All of the top performing algorithms are based on the affine camera model. Hence they are not dependent on the internal calibration of the cameras, whereas we assume calibrated cameras. To resolve this, the principal point is set to the middle of the image and the focal length to 700 pixels for images of size 480×640. This is the size for all sequences, but the last one, which has frame size 240×320 and consequently we halve the focal length for this case. Note that the true focal length is unknown, so the chosen is only empirically motivated[2].

The thresholds $\epsilon_1 = 0.0003$ and $\epsilon_2 = 0.0015$ are the same for all sequences. Parameters for spatial regularization: $\lambda = 1.66 \times 10^{-4}$ and $d_{max} = 0.04$. These have been found empirically and fixed for all sequences.

We compare with the following algorithms: Generalized Principal Component Analysis (GPCA) [16], Local Subspace Affinity (LSA) [17], RANSAC [6], Multi-Stage Learning (MSL) [13], Agglomerative Lossy Compression (ALC) [11] and two variants of Sparse Subspace Clustering (SSC) [4]. There are two versions of our brute-force algorithm. The first one (BF) is implemented according to the description in Section 6 and the second one is with the addition of a spatial prior (BF-S) as described in Section 6.1.

In Tables 1 and 2, the misclassification rates are presented. Our brute-force algorithm achieves very low error rates, both in terms of mean and median error rates. Note that even though we are only using three frames (the first, the middle and the last) in each sequence, we are able to obtain state-of-the-art results. Since we are actually recovering the 3D motion, it is very simple to add spatial regularization to the results. Still, even without such regularization, our approach outperforms the competitors, and with regularization, the error rates are significantly lower.

9. Discussion

Using a brute-force algorithm for computing the relative orientation of two projections may seem like a step back

[2]In the dataset, a 3×3 calibration matrix is provided, but this calibration is clearly incorrect since it has an aspect ratio of 0.75.

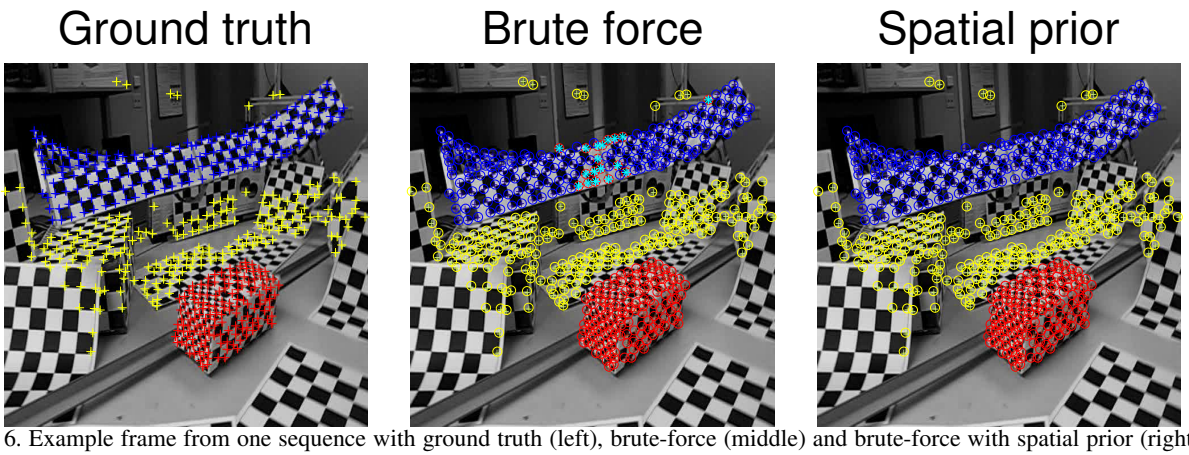

Figure 6. Example frame from one sequence with ground truth (left), brute-force (middle) and brute-force with spatial prior (right). The colors of the feature points indicate which motion class (blue, yellow, red). Feature points that are misclassified have been colored cyan (see middle figure). Note that the spatial prior is able to correct for all the errors.

Method	GPCA	LSA	RANSAC	MSL	ALC	SSC-B	SSC-N	BF	BF-S
Mean	31.95	5.80	25.78	10.38	5.20	4.49	2.97	2.11	**0.99**
Median	32.93	1.77	26.00	4.61	0.67	0.54	0.27	0.81	0.00

Table 1. Classification errors (%) for the 26 checkerboard sequences with 3 motions.

considering the many sophisticated algorithms that have been developed over the years. But why is it that none of the best performing algorithms for 3D motion segmentation does not use a pinhole camera model? This paper shows that a pinhole model is the correct choice and the lack of perspective methods that perform well on the Hopkins 155 benchmark is likely due to algorithmic failure modes, for example, the incapability of handling planar scenes.

The reported running times of the algorithm are well within the limits of being a suitable choice for many vision applications. Of course, the full search space cannot be used for a real-time system, but restricting the parameter space to small motions, the brute force approach becomes a viable and robust alternative for real-time visual odometry. Such an investigation is left as an avenue of further research.

A. Proof of Theorem 2

Since it is always possible to change coordinates, we can assume that $R = R' = I$. Furthermore, we note that if we can find points $\bar{x}$ and $\bar{x}'$ that satisfy the constraints in Theorem 1 as well as

$$\angle(\bar{x}, x) < \epsilon \text{ and } \angle(\bar{x}', x') < \epsilon, \tag{23}$$

then (by Theorem 1) we have also found our point X. This will prove useful. Let $\bar{\theta}$, $\bar{\theta}'$, etc denote the spherical coordinates of these points as defined in (8). We assume that $\bar{x}'$ is fixed and examine what constraints we get on $\bar{x}$. Recall the constraints from Theorem 1,

$$\bar{\theta} < \bar{\theta}' \tag{24}$$

$$\bar{\varphi} = \bar{\varphi}'. \tag{25}$$

From (23) we have that $\bar{x}'$ must lie in a small circle around x'. Consequently, (25) means that $\bar{x}$ must lie in the spherical wedge shown on the left in Figure 7 and (24) constrains it to the upper part of that wedge, as shown on the right in Figure 7.

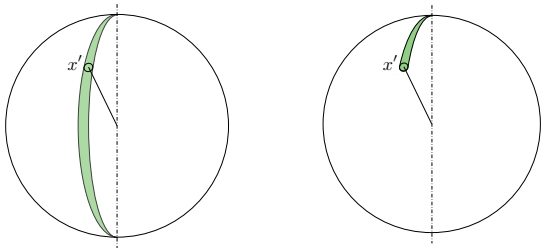

Figure 7. The constraints imposed on $\bar{x}$ being the reprojection of the 3D point in first camera. Equation (25) constrains $\bar{x}$ to the spherical wedge (left) and (24) to the upper part of that wedge (right).

But we also want $\angle(\bar{x}, x) < \epsilon$, which constrains $\bar{x}$ to a small circle around x. This means we must require the wedge from above to intersect this small circle. To complete the proof we need to translate this constraint to a constraint in the spherical coordinates. We get three cases.

Case 1: $\theta < \theta'$ Figure 8 shows the critical case. If the difference between φ and φ' is larger than this, then the two

Method	GPCA	LSA	RANSAC	MSL	ALC	SSC-B	SSC-N	BF	BF-S
Mean	6.09	2.57	6.52	4.46	1.55	0.83	1.12	0.85	**0.43**
Median	1.03	0.27	1.75	0.00	0.29	0.00	0.00	0.00	0.00

Table 2. Classification errors (%) for the 78 checkerboard sequences with 2 motions.

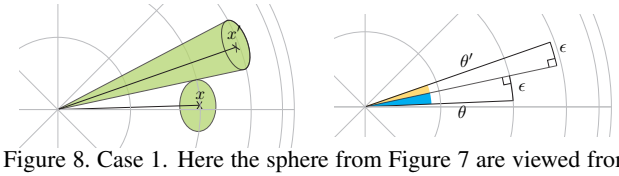

Figure 8. Case 1. Here the sphere from Figure 7 are viewed from above, i.e. the z-axis is pointing out of the paper. The green areas show the constraints on $\bar{x}$. For the two constraints to intersect they must not be further apart than this. The left image shows the setup for computing this limit angle.

sets have empty intersection. The limit can be computed by considering two right-angled triangles, see Figure 8. Let v denote the blue angle in that figure and v' the yellow one. The spherical law of sines yield,

$$\sin v = \frac{\sin \epsilon}{\sin \theta} \text{ and } \sin v' = \frac{\sin \epsilon}{\sin \theta'}. \tag{26}$$

and if we define $w = v + v'$, we can write the constraint $|\varphi - \varphi'| < w$. Note that, if either $\sin\theta$ or $\sin\theta'$ is smaller than $\sin\epsilon$ then w is not defined. In these cases one of the triangles is degenerated and the intersection is non-empty regardless of the φ's. One way to describe this is to set $w = \pi$.

Case 2, $\theta' < \theta < \theta' + 2\epsilon$: Figure 9 illustrates the critical position. Using the spherical law of cosines, we can compute w,

$$\cos\theta\cos\theta' + \sin\theta\sin\theta'\cos w = \cos 2\epsilon. \tag{27}$$

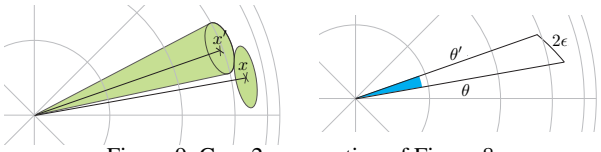

Figure 9. Case 2, see caption of Figure 8.

Case 3, $\theta \geq \theta' + 2\epsilon$: In this case the intersection is empty, regardless of φ and φ'.

References

[1] A. Blake and A. Zisserman. *Visual Reconstruction*. MIT Press, Cambridge, USA, 1987.

[2] A. Chiuso, R. Brockett, and S. Soatto. Optimal structure from motion: local ambiguities and global estimates. 39(3):195–228, 2000.

[3] O. Chum and J. Matas. Optimal randomized ransac. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(8):1472–1482, 2000.

[4] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *Conf. Computer Vision and Pattern Recognition*, Miami, Florida, 2009.

[5] O. Enqvist and F. Kahl. Two view geometry estimation with outliers. In *British Machine Vision Conf.*, London, UK, 2009.

[6] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Commun. Assoc. Comp. Mach.*, 24:381–395, 1981.

[7] R. Hartley and F. Kahl. Global optimization through rotation space search. *Int. Journal Computer Vision*, 82(1):64–79, 2009.

[8] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. Second Edition.

[9] F. Kahl and R. Hartley. Multiple view geometry under the L_∞-norm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(9):1603–1617, 2008.

[10] H. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

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

[12] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

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

[14] R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *Conf. Computer Vision and Pattern Recognition*, Minneapolis, USA, 2007.

[15] A. Vedaldi, G. Guidi, and S. Soatto. Moving forward in structure from motion. In *Conf. Computer Vision and Pattern Recognition*, June 2007.

[16] R. Vidal and R. Hartley. Motion segmentation with missing data using powerfactorization and gpca. In *Conf. Computer Vision and Pattern Recognition*, volume II, pages 310–316, Washington DC, USA, 2004.

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