Fast segmentation of sparse 3D point trajectories using group theoretical invariants

Vasileios Zografos, Reiner Lenz, Erik Ringaby, Michael Felsberg, Klas Nordberg
{vasileios.zografos, reiner.lenz, erik.ringaby, michael.felsberg, klas.nordberg}@liu.se

Computer Vision Lab, Linköping University, Linköping, Sweden

Abstract. We present a novel approach for segmenting different motions from 3D trajectories. Our approach uses the theory of transformation groups to derive a set of invariants of 3D points located on the same rigid object. These invariants are inexpensive to calculate, involving primarily QR factorizations of small matrices. The invariants are easily converted into a set of robust motion affinities and with the use of a local sampling scheme and spectral clustering, they can be incorporated into a highly efficient motion segmentation algorithm. We have also captured a new multi-object 3D motion dataset, on which we have evaluated our approach, and compared against state-of-the-art competing methods from literature. Our results show that our approach outperforms all methods while being robust to perspective distortions and degenerate configurations.

1 Introduction

Motion is a powerful low-level cue, which when correctly disambiguated can significantly aid many computer vision problems, such as object segmentation, video post-processing, visual surveillance, robotic and autonomous vehicle navigation and activity recognition. Thus in the last few years a number of approaches have tried to solve the sparse motion segmentation problem. Namely, grouping point trajectories where each group is associated with a distinct 3D motion. 3D Motion segmentation can be performed directly in 2D. Assume the measured 2D coordinates of a point as:

$$\tilde{x} = \frac{sX}{Z} + o_1 + \varepsilon_1, \qquad \tilde{y} = \frac{sY}{Z} + o_2 + \varepsilon_2, \tag{1}$$

where $\mathbf{x}=[X, Y, Z, 1]^T$ are its 3D homogeneous coordinates relative to the camera coordinate system, Z is the depth, s, o_1, o_2 the intrinsic camera parameters and $[\varepsilon_1, \varepsilon_2]$ the tracking noise. We can rewrite (1) for point $p = 1...P$ and frame $f = 1...F$ as:

$$\mathbf{w}_p^f = \begin{pmatrix} \tilde{x}_p^f \\ \tilde{y}_p^f \end{pmatrix} = \frac{1}{Z_p^f} \left(s \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} + \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} \mathbf{c}_3 \right) \mathbf{x}_p, \tag{2}$$

with $\mathbf{c}_k=[r_{k1}\, r_{k2}\, r_{k3}\, t_k]$ being the elements of the rigid transformation of the 3D points $\mathbf{x}_p$, now expressed relative to the world coordinate system. The vectors $\mathbf{w}_p^f$ are aggregated into a $2F \times P$ data matrix $\mathbf{W}=[\mathbf{w}_p^f]$ for all points and all image frames.

The traditional approach to 3D motion segmentation from 2D uses an affine camera model assumption, in which Z_p^f is constant and as a consequence, $\mathbf{W}$ factorizes into

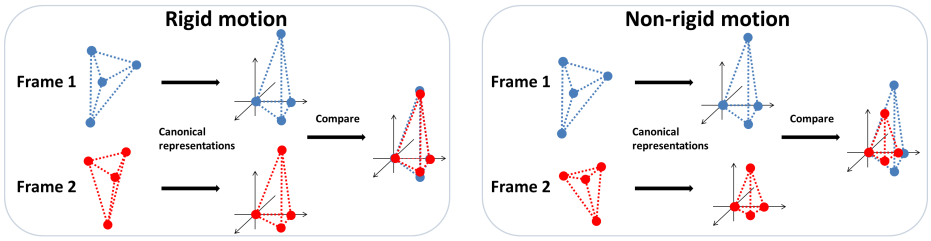

Fig. 1. Overview of our method. Given a set of rigid points (e.g. 4) in 2 frames, we can find a canonical representation, which allows us to compare the points between frames. For rigid motions (left example), the canonical representations will be near-identical. For non-rigid motions (right example), the canonical representations will differ considerably.

a motion component and a 3D shape component [1]. This result implies that each trajectory lies in a 4D subspace and as such motion segmentation may be solved as the equivalent task of subspace clustering [2]. Notable solutions are the SSC method [3], which describes each point by a sparse set of points from the same subspace; the LRR method [4], which tries to recover a low-rank representation of the data points; LSR by [5], which exploits the data sample correlation and groups points that have high correlation together; the SC approach [6], which looks at the cosine angle between pairs of points as the clustering criterion; or the more recent DiSC method [7], which uses an ensemble of quadratic classifiers each trained from unlabelled data. The main advantage of working with 2D data is that 2D point trajectories may be easily obtained from a single camera. However, using only 2D trajectories to segment 3D motions can be problematic due to ambiguities and potential degeneracies in the motions from the lack of associated depth information. Also, because of the affine camera model employed, 2D methods are prone to failure when they are faced with strong perspective distortions.

With the arrival of new and inexpensive RGB-D sensors (structured light and ToF cameras) it is possible to obtain depth, and thus 3D trajectories, from a single device. Three-dimensional data is not so much affected by degeneracies or perspective distortions, and as a result motion segmentation should be more accurate if carried out directly in 3D. If depth is available it may be incorporated into the final solution in different ways. The first way is the *depth scaling* approach: Given a depth measurement $\tilde{Z}{=}Z{+}\delta$ for every point in the scene, where δ is the depth noise, we may instead use the fully projective camera model. Multiplying the depth onto the image coordinates in (1), in homogeneous form, gives us:

$$\mathbf{w}_p^f = \begin{pmatrix} \tilde{Z}_p^f \cdot \tilde{x}_p^f \\ \tilde{Z}_p^f \cdot \tilde{y}_p^f \\ \tilde{Z}_p^f \end{pmatrix} = \begin{pmatrix} s \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} + \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} \mathbf{c}_3 \\ \mathbf{c}_3 \end{pmatrix} \mathbf{x}_p. \tag{3}$$

As with (2), the depth scaled version (3) allows us to construct a $3F{\times}P$ data matrix $\mathbf{W}_{\mathcal{P}}$ that has columns which lie in a 4D subspace. This means that both (2) and (3), can be solved in an identical manner via subspace clustering. Note that where $\mathbf{W}$ and $\mathbf{W}_{\mathcal{P}}$ differ is in the way that they are affected by noise. $\mathbf{W}$ is only perturbed by the

tracking noise $[\varepsilon_1, \varepsilon_2]$, whereas $\mathbf{W}_{\mathcal{P}}$ is perturbed by the multiplicative effects of both the tracking noise and the depth noise δ. As a result, if there is considerable noise in the depth measurements, then using $\mathbf{W}_{\mathcal{P}}$ may produce inferior segmentation results.

The second way is by extracting 3D trajectories and using motion segmentation methods that can work with three-dimensional data. Recent techniques for extracting very accurate 3D trajectories from depth sensors include the work by by [8] that uses a particle filter and that by [9] that tracks surface patches inside a Lucas-Kanade framework. The simplest and fastest, albeit less precise, way of extracting the 3D trajectories is directly from the combination of 2D and depth measurements. Similarly to (3), given a tracked point from (1) and its depth measurement $\tilde{Z}$, the 3D position of the point can be estimated as

$$\begin{aligned} \tilde{X} &= \frac{\tilde{Z}(\tilde{x}-o_1)}{s} = X + \frac{\delta X}{Z} + \frac{\varepsilon_1 Z}{s} + \frac{\varepsilon_1 \delta}{s}, \\ \tilde{Y} &= \frac{\tilde{Z}(\tilde{y}-o_2)}{s} = Y + \frac{\delta Y}{Z} + \frac{\varepsilon_2 Z}{s} + \frac{\varepsilon_2 \delta}{s}. \end{aligned} \tag{4}$$

The estimates $[\tilde{X}, \tilde{Y}, \tilde{Z}]$ are calculated in the camera coordinate system instead of the world coordinate system, which would require estimation of the camera pose. However, since we are only interested here in the relative positions between points, and those do not change from camera to world coordinates, it suffices to use the representation in (4).

The literature on motion segmentation from 3D is rather limited mainly because obtaining 3D trajectories has not always been an easy task. An early example is the work by [10], where they use the variance of the Euclidean distance between pairs of points as the motion similarity (affinity) criterion. Once the authors have constructed a pairwise affinity matrix for all the 3D points in the scene, they obtain the motion segmentation solution by using spectral clustering [11]. More recently, in [12] the authors also use the variance of the Euclidean distance, but instead they recover the final motion clusters using a maximal cliques algorithm. In [13], similarly to [12], the authors build a graph from the interest points in the 2D image. The edges of the graph are pruned using a 3D velocity similarity criterion and additional refinement techniques. In their work on both monocular and stereoscopic vision for driving assistance systems [14], the authors segment motions either by checking a number of geometric constraints (for the monocular case) or by calculating the velocity of 3D points (for the stereoscopic case). Both cases are limited in that they require estimation of the camera's ego-motion beforehand. In [15] the authors obtain dense 3D trajectories from a Time-of-Flight (ToF) camera and segment motions using 3D velocity and distance. Finally in [16] the authors present an algorithm for segmenting motions in 3D SLAM applications, which employs various motion grouping criteria based on pairwise distances. However, their method is only presented for 2 frames at a time and extension to longer sequences is not so straight forward due to the problem of label switching between two-frame windows.

As of late, new methods have appeared for object segmentation from motion and depth, which use additional information beyond 3D geometry, such as image intensity. For example the EM-style method by [17] that switches between segmentation and motion estimation, or the semi-supervised object segmentation from RGB-D by [18] or that by [19]. However related they might appear, these methods solve different, higher-level problems (dense object segmentation vs sparse motion segmentation). In addition, they are not generic solutions but are tied to specific sensors. For these reasons they are not explored further in this paper.

Our method (see Fig. 1) is a novel approach, which works directly with 3D trajectories and uses the theory of transformation groups to define a set of invariants of points located on the same rigid object at different frames. These invariants can be readily converted into a set of robust motion affinities between the points. Because calculation of the invariants is based primarily on QR factorizations of relatively small matrices and can be applied to all points simultaneously, the method is also very fast. Coupled with a localised sampling step and spectral clustering for recovering the final clusters, our method provides a very accurate solution to the problem of 3D motion segmentation from sparse 3D point trajectories. We have tested our approach on real and synthetic sequences of 3D motions and compared against state-of-the-art methods. The results show that our method outperforms all existing methods, especially in sequences that exhibit strong perspective effects and degenerate configurations. Our key contributions are:

- A fast and highly accurate algorithm for segmenting 3D trajectories,
- A framework for simultaneous calculation of group-theoretical motion invariants on sets of 3D points,
- A multi-object dataset for evaluating sparse 3D motion segmentation algorithms,
- Comprehensive evaluation of state-of-the-art motion segmentation approaches on real and synthetic data.

2 Background theory and method

Consider two 3D points located on different rigid objects, with the objects moving independently in two frames (see Ex.1 in Fig. 2). The Euclidean distance between the points is not invariant but changes over time. In other words, non-zero variance in the pairwise distance suggests that the points lie on different moving objects. However, the opposite is not always true. For example, the objects might have a relative motion that does not change the distance between the points (see Ex.2 in Fig. 2). As such, non-varying Euclidean distance is a necessary but not a sufficient condition for determining relative motion between two points. Despite this inherent weakness, the Euclidean distance invariant can be used for motion segmentation [10, 12], and given enough pairwise point samples and image frames, the impact of these ambiguous configurations may be effectively reduced. However, since the number of frames is finite, there is a limitation as to how much we can moderate the effects of ambiguities by using more images.

Our hypothesis is that we can use more invariants, say of N points, to make motion segmentation even more robust to ambiguous and degenerate configurations. Take for example N=4 points that define a tetrahedron in 3D space. There are 6 geometric invariants that uniquely determine a rigid tetrahedron. One such choice is the 6 pairwise lengths of the tetrahedron. Thus if we sample 4 points, we should have 6 different values that we could test for invariance. Even though still not sufficient, this plurality of invariants reduces considerably the likelihood of obtaining a tetrahedron that maintains all its geometric properties unchanged under non-rigid motion. The problem with calculating the geometric invariants from the example above at every frame, is that it can quickly become a very expensive task, especially given a large number of points and

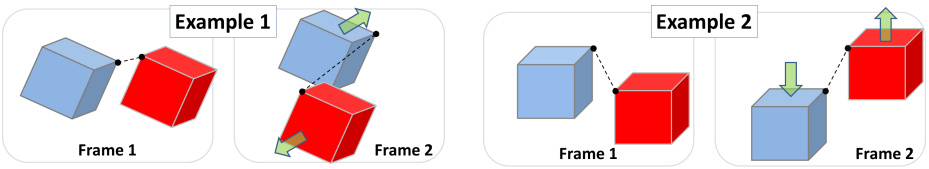

Fig. 2. Example 1 showing that the 3D distance between two points changes if the points lie on different moving objects. Example 2 showing that under certain configurations the distance might stay the same yet the two points lie on different moving objects.

long frame sequences. In addition, if we want to use more than 4 points at a time, the number of invariants as well as their associated computational costs increase considerably. For example, the pairwise lengths between N points form an over-complete set of *dependent* invariants that grows quadratically with increasing N. For this reason, we have used the elements of group theory to define a framework for motion segmentation using invariants, that is *general*: extending to any number $N \geq 2$ of points; *simple*: defining geometric invariants in a straightforward way; and *fast*: where the invariants can be calculated by factorizations of small matrices.

2.1 Group theoretical invariants

The use of group invariants is a well studied subject in computer vision (cf. the overview article [20]), with particular focus on shape matching and object recognition. There are many ways to construct such invariants, for example the approach described by [21] using Lie theory and PDEs, or the method by [22]. While we follow along the same lines, in this paper we apply the theory of group invariants to the mostly unexplored problem of 3D motion segmentation. We use the fact that transformation groups split the space on which they operate into equivalence classes and for each such equivalence class we select one unique representative. The parameters of this representative element are by definition invariant under the group action. This approach for extracting the invariants is simpler than [21, 23] and only involves QR factorizations.

The main idea that we will exploit here, is that in general we may recover a canonical representation of a collection of N 3D points by some unique alignment with the coordinate axes, so that the effects of the rigid transformation (i.e. from the motion of the points) can be removed. In that canonical representation, the N points can be compared along the frames without the temporal effects of their motion. If the points come from the same rigid object then their canonical coordinates should be near-identical (invariant) over all frames F (see Fig. 1 left). If the points do not come from the same rigid object, then there would be other influences outside the rigid transformation group that will not be removed by the alignment procedure and will show up as variation in their canonical coordinates and thus changes in the invariants (see Fig. 1 right).

G-invariant: We define the 3D coordinates of a point as the vector $\mathbf{x}$, a collection of N such vectors as the $3 \times N$ matrix $\mathbf{X}$ and the set of all matrices as $\mathcal{X}$. Next we introduce the special Euclidean transformation group SE(3) as the *set* of all element

pairs $g{=}(\mathbf{R},\mathbf{t})$, where $\mathbf{R}$ is a 3D rotation matrix and $\mathbf{t}$ is a 3D translation vector. We can operate on matrices $\mathbf{X}$ by transforming all points simultaneously as $g\cdot\mathbf{X}$, and define the *orbit* of an element $\mathbf{X}$ under the group SE(3) as the set

$$\mathcal{O}_{\mathbf{X}}^{\mathrm{SE(3)}} := \{g\cdot\mathbf{X} \mid g\in \mathrm{SE(3)}\}\,. \tag{5}$$

A *G-invariant* may be defined as the function $\mathcal{F}:\mathcal{X}\rightarrow\mathbb{R}$, which has *constant* values $\alpha_{\mathbf{X}}$ on the orbit $\mathcal{O}_{\mathbf{X}}^{\mathrm{SE(3)}}$. This is written as

$$\mathcal{F}(g\cdot\mathbf{X}) := \alpha_{\mathbf{X}} \text{ for all } \; g\in \mathrm{SE(3)}. \tag{6}$$

We may generate a complete set of independent G-invariants by starting from an arbitrary but fixed element $\mathbf{X}$ on an orbit, and then constructing a *unique* element $g_{\mathbf{X}}\in$ SE(3) such that $g_{\mathbf{X}}\cdot\mathbf{X}{=}\mathbf{X}_{\mathrm{o}}$. We call $\mathbf{X}_{\mathrm{o}}$ the *canonical* element on the orbit with $g_{\mathbf{X}_{\mathrm{o}}}{=}[\mathbf{I},\mathbf{0}]$ being the identity element in SE(3), $\mathbf{0}$ the zero vector and $\mathbf{I}$ the identity matrix.

The construction[1] of the unique element is carried out by the following steps:

1. Given $\mathbf{X}$*, select* $t{=}{-}\mathbf{x}_1$*, where* $\mathbf{x}_1$ *is the first column in* $\mathbf{X}$*, and subtract it from all other columns in the matrix:*

$$(\mathbf{I},-\mathbf{x}_1)\mathbf{X} = [\mathbf{0},\hat{\mathbf{X}}], \tag{7}$$

where $\hat{\mathbf{X}}$ *are the translated last three columns of* $\mathbf{X}$*.*
2. Using the QR factorization, $\hat{\mathbf{X}}$ *is decomposed as* $\hat{\mathbf{X}}{=}\mathbf{R}\mathbf{V}$*, where* $\mathbf{R}$ *is a rotation matrix (by enforcing a determinant of 1) and*

$$\mathbf{V} = (\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3) = \begin{pmatrix} a & b & d \\ 0 & c & e \\ 0 & 0 & f \end{pmatrix} \tag{8}$$

is a 3×3 upper triangular matrix.
3. Finally define the second part of the transformation by $(\mathbf{R}^T,0)$ *and obtain:*

$$\begin{aligned} g\cdot\mathbf{X} &= (\mathbf{R}^T,\mathbf{0})\,(\mathbf{I},-\mathbf{x}_1)\,\mathbf{X} \\ &= (\mathbf{R}^T,\mathbf{0})[\mathbf{0},\hat{\mathbf{X}}] = [\mathbf{0},\mathbf{V}]. \end{aligned} \tag{9}$$

Due to the uniqueness of the translation and the QR factorization it follows that $g_{[\mathbf{0},\mathbf{V}]}{=}[\mathbf{I},\mathbf{0}]$. Each column in $[\mathbf{0},\mathbf{V}]$ now contains the canonical coordinates of the transformed points from $\mathbf{X}$. This process is shown in Fig. 3. A geometrical interpretation of the above is that the first three columns of $\mathbf{X}$ represent the corners of a base triangle. The ordering of the columns is arbitrary and we may eliminate the effects of permutations by a fixed ordering of the columns in $\mathbf{X}$ according to falling norms and some basic rule to break ties. Geometrically, this means we choose the point nearest to the origin of the original coordinate system as the starting corner of the base triangle. Then we sort out the remaining three points such that the lengths of the corresponding sides of the triangle are increasing.

Following the above construction, we have many choices for defining G-invariant functions $\mathcal{F}$. In particular, we wish to detect large variations in $\mathbf{V}$, over the F frames in the sequence. Large variations in the elements of $\mathbf{V}$ will most likely indicate that

[1] For ease of exposition, we will consider here that N=4, but the following construct is similar for any $N{\geq}2$.

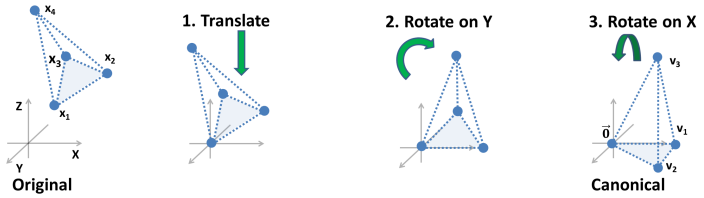

Fig. 3. Calculation of the canonical representation of a set of points. First the points are translated to the origin using the first vertex of a chosen base triangle (shaded), and then a combined rotation is recovered that aligns the base triangle with the axes.

the original 4 points in $\mathbf{X}$ do not come from the same rigid object. We could of course look at the variance of the 6 nonzero elements in $\mathbf{V}$ from (8) and calculate 6 invariants. However since the 3 columns $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ of $\mathbf{V}$ represent the canonical coordinates of the last three 3D points in $\mathbf{X}$ (the canonical coordinates of the first point are always 0 by construction in (9)), a geometrically inspired and more robust set of G-invariants are the median centered ℓ^2-norms of the columns of $\mathbf{V}$:

$$\mathcal{F}_n(g \cdot \mathbf{X}) = \frac{1}{F} \sum_f \left\| \mathbf{v}_n^f - \boldsymbol{\mu}_n \right\|_2 , \tag{10}$$

with $n{=}1, ..., 3$, $f{=}1, ..., F$, $\mathbf{v}_n^f$ the n^{th} column of $\mathbf{V}$ at frame f, and $\boldsymbol{\mu}_n$ being the marginal median of $\mathbf{v}_n$ over the frames. In the absence of noise, (10) will be zero for all the points that lie on the same object. Furthermore due to the geometric nature of the ℓ^2-norm, the 3 resulting scalar G-invariants from (10), will behave in a more predictable way in the presence of noise than the individual elements of $\mathbf{V}$. This is the reason why it is preferable to consider *functions* of the 6 elements and obtain only 3 robust G-invariants, rather than use the 6 elements directly and obtain 6 non-robust G-invariants. It should be noted that by constructing the robust G-invariants via (10), we no longer obtain a complete set of invariants, but rather the robust G-invariants now span a 3-dimensional subspace in $\mathbb{R}^6$. From the robust invariants in (10) we may construct a motion affinity measure (explained in the next section) and form the basis of our motion segmentation algorithm.

3 The motion segmentation algorithm

Our motion segmentation algorithm follows the typical pipeline which involves constructing a pairwise motion affinity matrix for all the points in the scene, and then using spectral clustering [11] for recovering the motion clusters. The construction of the affinity matrix is preceded by an efficient local sampling step, designed to exploit the property of "common fate" in proximal data, and thereby improve the segmentation results. Once again, for ease of understanding we present the algorithm for $N{=}4$ points, but the algorithm has a similar extension to any $N{>}2$.

3.1 The motion affinity

In the previous section we have presented the construction of the G-invariants for $N{=}4$ number of points (i.e. a 4-tuple). In order to utilize these invariants for motion segmentation, we need to apply them to all the 3D points in the scene and extract a motion

relationship between them. In other words, given a $3F\times P$ data matrix $\mathbf{W}$ of P 3D points in F frames, what we want is to use the G-invariants for obtaining a $P\times P$ *pairwise* motion affinity matrix between all the 3D points. The pairwise affinity matrix can be used as input to a clustering algorithm (e.g. [11]) and obtain the final clusters.

Since (10) represents a direct relationship between a 4-tuple we may only say something about 4 points at a time and not about only 2 points (pairwise). Consider the set S_1 of all possible 4-tuples of P points. Sampling all the elements of S_1 and deriving a relationship between pairs of points would be practically infeasible. Instead, we can use the technique by [24], which allows us to sample a small number C from the set S_2 of 3-tuples, with $C\ll|S_2|<|S_1|$, and create the $P\times C$ *4-way* affinity matrix $\mathbf{E}$. The pairwise affinity matrix may be then approximated as $\mathbf{A}\approx\mathbf{E}\mathbf{E}^T$. The 4-way affinity matrix $\mathbf{E}$ may be calculated, one column $\mathbf{e}_c$ at a time, by first sampling a 3-tuple and then calculating the 4-way affinity between that and every other scene point in turn. The 4-way affinity is defined as:

$$\mathbf{e}_c(p)=\exp\left(-d_c(p)/\sigma_c\right),\quad c=1,\ldots,C, \tag{11}$$

where from (10) we have

$$d_c(p)=\mathcal{F}_n(g\cdot\mathbf{X}_{p,c}),\quad p=1,\ldots,P, \tag{12}$$

with $\mathbf{X}_{p,c}=[\mathbf{W}(f,S_2(c)),\mathbf{W}(f,p)]$ being the 4 points. σ_c is a kernel parameter defined as the ρ^{th} percentile of each column $\mathbf{e}_c$. This way of choosing σ_c allows for both local scaling of the kernel in each column and a kernel parameter which adapts to the range of the data.

Completing one column of $\mathbf{E}$ requires P-3 evaluations of (10) (and thus QR factorizations) for the calculation of the G-invariants. However, we may reduce the computations considerably by exploiting the incremental construction of the G-invariants and noting that for each of the P-3 evaluations, the initially sampled 3-tuple $\mathbf{W}(f,S_2(c))$ remains fixed. We may therefore pre-calculate the first two columns $\mathbf{v}_1,\mathbf{v}_2$ of $\mathbf{V}$ in (9) from the fixed 3-tuple, using the exact same steps described in Section 2.1. The last column $\mathbf{v}_3$ of $\mathbf{V}$ may be obtained simultaneously for all the 3D scene points $\mathbf{W}(f,:)$ by a matrix multiplication with the rotation matrix $\mathbf{R}^T$ estimated from the QR factorization of the 3-tuple. This incremental construction allows us to go from $C\times(P-3)$ QR factorizations per frame required to construct $\mathbf{E}$, to C QR factorizations and C matrix multiplications per frame, which is considerably faster.

3.2 Local sampling

Since we are only sampling a small number C of 3-tuples for approximating the pairwise affinity matrix $\mathbf{A}$, it is important to increase the probability that each 3-tuple comes from the same rigid object. Otherwise, there will be a large variance in the canonical coordinates of the points due to non-rigid motion and the 4-way affinity in (11) will always be low, irrespective of the 4^{th} point used. In [24] the authors proposed randomly sampling the 3-tuples. However, this requires a large number C of samples to ensure that enough columns of $\mathbf{E}$ contain points from the same object. An iterative technique was suggested by [25], which involves random sampling, obtaining an initial motion

segmentation solution and then repeating the process by sampling from the identified motion clusters. We propose a better sampling scheme, which exploits the locality property in the data. That is, points on the same object generally lie in close proximity to other points from the same object. Local sampling has a much higher likelihood of obtaining 3-tuples that are rigid and as a result can produce good results with far fewer number of samples. To acquire C local 3-tuples points we start by randomly sampling C points. For each point, we then sample the 2-nearest neighbours in 3D Euclidean distance. The advantage of the local sampling technique can be seen in Fig. 4 and the complete motion segmentation method is formulated in Algorithm 1.

Algorithm 1: Motion Segmentation algorithm

Input: data matrix $\mathbf{W}$, # of samples C, # of motions m ; **Output:** motion labels $\mathbf{Y}$

Local sampling of C number of 3-tuples from S_2

for *sample* $c = 1 : C$ **do**

  for *frame* $f = 1 : F$ **do**

    Let $\mathbf{X} = \mathbf{W}(f, S_2(c))$ and $\mathbf{t} = \mathbf{X}(:, 1)$

    Translate as $\hat{\mathbf{X}} = \mathbf{X} - \mathbf{t}$ and $\mathbf{W}_1 = \mathbf{W}(f, :) - \mathbf{t}$

    QR factorize $\hat{\mathbf{X}} = \mathbf{RV}$

    Transform ALL points $\mathbf{W}_2(f, :) = \mathbf{R}^T \mathbf{W}_1$

  Calculate $d_c(p) = \frac{1}{F} \sum_f \left\| \mathbf{W}_2(f, p) - \boldsymbol{\mu}_p \right\|_2$ from (12)

  Calculate $\mathbf{E}$ column as $e_c(p) = \exp\left(-d_c(p)/\sigma_c\right)$ from (11)

$\mathbf{Y}$ = spectral clustering on $\mathbf{A} = \mathbf{EE}^T$ with m clusters

3.3 Generalization to any $N \geq 2$

The construction of the G-invariants and Algorithm 1 have been described for N=4 points, but have similar constructions for any number $N \geq 2$ points. For N=3 points the construction is identical, leading to a unique QR factorization, 2 unique robust G-invariants from (10), and the formation of a 3-way affinity matrix $\mathbf{E}$ from (11). For N>4, we first calculate the G-invariants for 4 arbitrary but fixed points as in Sec. 2.1 and then compute the canonical coordinates of the remaining points based on the estimated QR factorization. We obtain $3*(N\text{-}2)$ nonzero elements in $\mathbf{V}$ from (8), resulting in $N-1$ robust G-invariants that grow linearly with N. For the special case of N=2, the QR factorization is no longer unique but the resulting G-invariants still are. Furthermore, $\mathbf{V}$ is now a column vector and (10) leads to a single G-invariant. This G-invariant is a robust form of the Euclidean distance variance between the two points. This is very closely related to the motion affinity used by [10, 12] and in fact both published methods can be considered as a special case of our framework for N=2. Finally when N=2, Algorithm 1 simplifies to a direct calculation of the pairwise affinity matrix $\mathbf{A}$ without the need for local sampling.

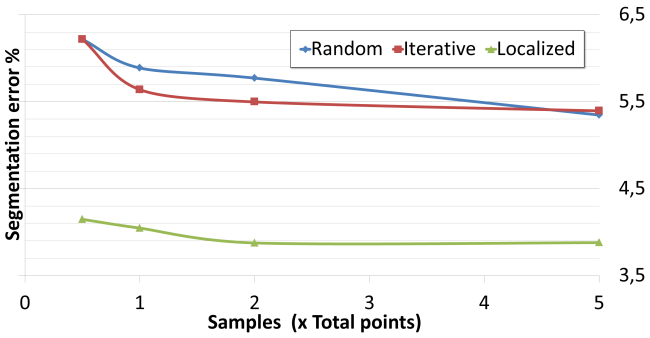

Fig. 4. Effect of different sampling schemes on the segmentation error of the real dataset from Sec. 4, as a function of the number C of N-tuples sampled.

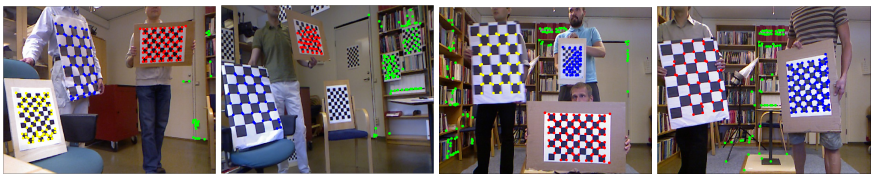

Fig. 5. Samples from our captured real dataset with superimposed tracked feature points on multiple moving objects.

4 Experiments

We present the results from our experiments on real and synthetic data. We have compared our method against compatible state-of-the-art approaches from literature that use sparse point trajectories only, and where computer code was available. Specifically, the 2D motion segmentation methods SC [6], SSC [3], DiSC [7], LSR [5] and LRR [4] and the 3D method by [10, 12] denoted here as Euc3D. Furthermore, all 2D methods were extended by scaling with the depth measurements (depth scaling approach $\mathbf{W}_{\mathcal{P}}$ from (3)), and are denoted here as SC$_{\mathcal{P}}$, SSC$_{\mathcal{P}}$, DiSC$_{\mathcal{P}}$, LSR$_{\mathcal{P}}$ and LRR$_{\mathcal{P}}$ respectively. All competing methods were tuned either as suggested by their corresponding authors or to the best of our ability. Our method's parameters were set to: N=10, C=2P where P is the number of points in the scene, and ρ=15 as the percentile of σ_c from (11). All algorithms were given the number of motions m, and their parameters were kept fixed across all experiments.

Unfortunately there is no publicly available 3D trajectory dataset with multiple motions. To our knowledge there are two dynamic RGB-D datasets in literature, both requiring extraction of 3D trajectories, but none of which were suitable for our tests. The first [26], contains very few sequences where only two articulated objects are moving and over a very limited range. The second [27], consists of multiple objects moving in front of a static camera, but due to the heavy motion blur and limited presence of the objects in the scene, it has not been possible to extract any useful trajectories. We have therefore captured our own 3D dataset and have used it to evaluate the different

methods. This new dataset contains 460 sequences of 10 frames each, with 2-5 moving objects and with on average 390 tracked points per sequence. The sequences have been captured with a Kinect sensor, and 3D trajectories were extracted using the simple approach from (4). We have pre-processed the data in the following ways: first we have used the cross-checking scheme by [28] to only retain complete trajectories and reject those with gross tracking errors. Second, we have discarded any points with missing or ambiguous depth measurements. Such points are usually located on depth discontinuity boundaries, and are therefore easily identified by means of an edge detector in the depth image. Note that despite this preprocessing, the dataset still contains many challenging 3D trajectories due to the considerable amount of noise, particularly in the depth measurements. Labelled examples from the dataset are illustrated in Fig. 5.

The results from running all methods on the 3D dataset are summarised in terms of the segmentation error in Table 1, for the different types of motions. As expected all methods start with a low error for 2 motions (with the exception of LRR and Euc3D) and deteriorate at different rates as the number of motions (and hence the 3D trajectories) increase. We see that only our method has remained consistently accurate with a low error, and obtains the overall best result for the complete set of 460 sequences. Second best is DiSC but at a much higher computational cost. Unsurprisingly, the other competing 3D method (Euc3D) shows quite poor results, which is evidence that a single invariant between two points is not robust enough, and one has to consider multiple invariants. Regarding the depth scaled versions of the 2D methods, they are systematically much worse than when using the 2D trajectories alone. This is expected and can be attributed to two factors: the considerable amount of noise in the depth measurements, and the multiplicative effect of the tracking and depth noise already explained in Sec. 1. The performance of each algorithm is further illustrated in the histograms in Fig. 6. It is evident that our method together with DiSC and SSC have very similar performance in terms of the cumulative segmentation errors that they obtain, with most sequences from the dataset in the range of 0-10%, and with very few sequences in the range 20-50%. This is not the case for methods such as LRR, LSR and Euc3D, since they segment considerably more sequences with a high error in the range 20-50%.

In terms of speed (last column of Table 1), LSR is the fastest method, albeit rather inaccurate, with Euc3D following closely with around 20 msec more computational time on average. Our method is the third fastest at 0.34 sec but achieves more than double the accuracy of Euc3D and LSR. Notice that the next two most accurate methods (DiSC and SSC) are 40-400 times slower by comparison. In summary, only our approach is both fast and accurate enough to provide a viable solution to the motion segmentation problem, while being largely unaffected by the additional depth error in the 3D trajectories.

We have also performed an extra set of experiments on synthetic 3D trajectories in order to further evaluate the performance of each algorithm against: different *types* of motions, increasing *number of objects*, increasing *noise* and decreasing *trajectory length*. Each test was executed 100 times with randomized placement and motion of the objects. Fig. 7 (upper left) shows the segmentation results for general 3D motions, motions with strong perspective distortions and degenerate motions (i.e. all points located on 3D planes undergoing rotations and translations parallel to the image plane).

We see that for general 3D motions, all methods perform equally well. However, for perspective and degenerate motions, the majority of algorithms fail, particularly the 2D methods that assume an affine camera model. Only our method and DiSC are able to deal with the challenging projective and degenerate motions. In Fig. 7 (upper right), we see the segmentation error vs an increasing number of objects. Our method is largely unaffected by the number of objects, something which is also reflected in Table 1. DiSC is the only other method that has similar accuracy in this experiment. In Fig. 7 (lower left), we show segmentation error vs trajectory length, that is, motions becoming increasingly smaller with each experiment. Smaller motions should be more difficult to disambiguate. We observe that our method is once again the most accurate, and not really degrading much by the decreasing trajectories. Fig. 7 (lower right) shows segmentation error vs noise, scaled independently in both 2D and depth. Notice how LRR breaks down due to its sensitive numerical nature. Our method is not the most robust but shows good very good performance in relation to the majority of competitors. Although geometric in nature, our method has the random element in choosing the N point samples and their ordering in the F frames. We believe that we may further improve the robustness to noise by incorporating an explicit error model, which can describe the joint perturbations of 3D points, in order to select more stable base triangles for the canonical representation.

Lastly, we examine the performance of our method against different parameter settings. All the experiments have been run on the real dataset, where each parameter was changed in turn while the rest of the parameters were kept fixed. There are three parameters that are used in our algorithm: the N-tuple **sample size** parameter C, the **size** N of the N-tuple, and the **percentile** parameter ρ for the kernel size σ_c. We see that the performance of the method is very stable and with low error (i.e. wide valleys) across a large region of the parameter space. This illustrates that our method can perform well without the need for extensive tuning.

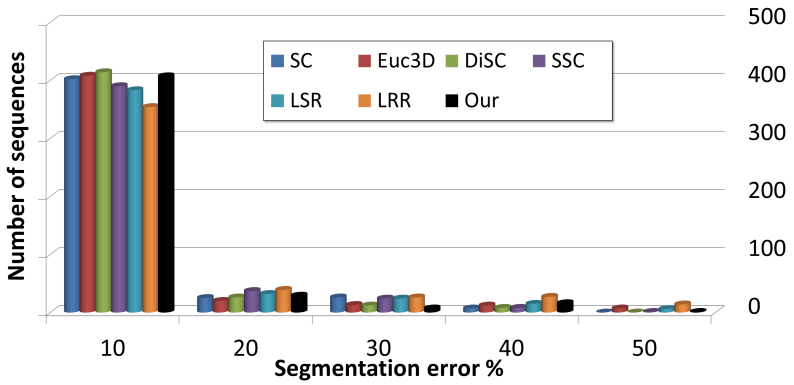

Fig. 6. Segmentation error histograms showing the detailed performance of each algorithm on the real dataset.

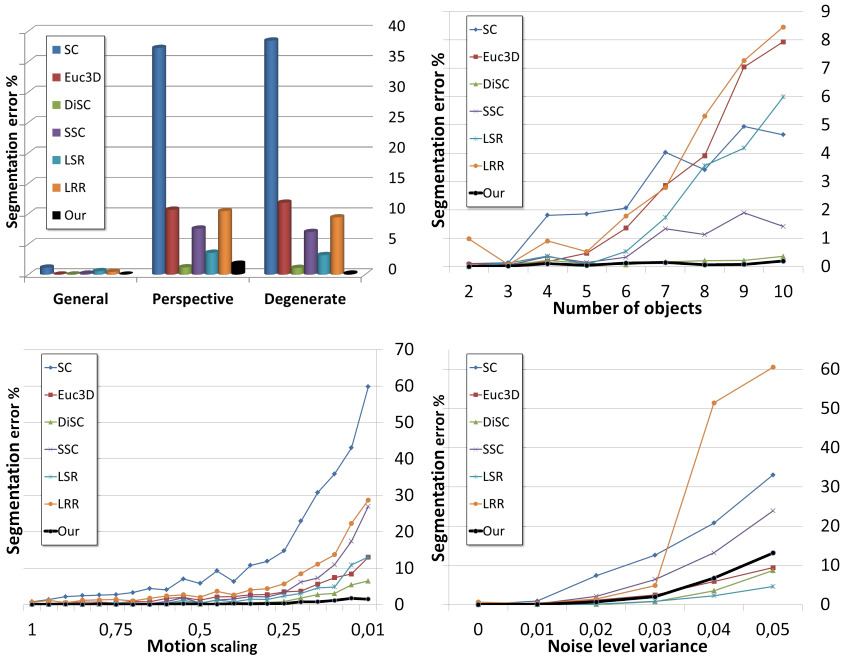

Fig. 7. Results from the synthetic dataset of segmentation error vs different types of motions (upper left), increasing number of objects (upper right), decreasing trajectory length (lower left) and increasing noise (lower right). Our method is displayed in bold.

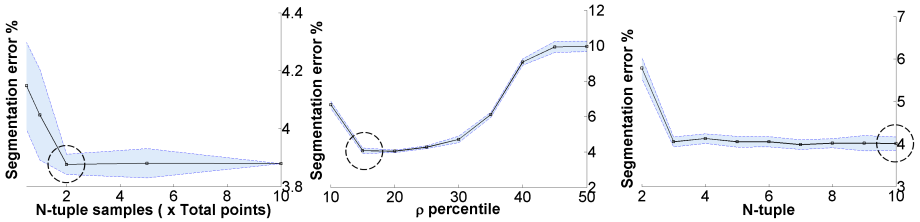

Fig. 8. Average segmentation error % versus different parameter values (solid line). Each test was executed 10 different times and the standard deviation is shown in the shaded regions. The chosen parameters for the main tests in Table 1 are indicated by the circles.

	2 motions 15 seq.	**3 motions** 215 seq.	**4 motions** 220 seq.	**5 motions** 10 seq.	**Total** 460 seq.	**Time**
Method	Average segmentation error					
SC [6]	0.04%	6.21%	8.42%	8.52%	7.12%	0.64 sec
$SC_{\mathcal{P}}$	0.89%	15.62%	16.73%	22.11%	15.81%	— ” —
SSC [3]	0.27%	**5.38%**	5.01%	27.51%	5.52%	113.10 sec
$SSC_{\mathcal{P}}$	0.61%	7.99%	8.19%	36.33%	8.47%	— ” —
DiSC [7]	0.13%	6.66%	2.55%	8.08%	4.51%	9.53 sec
$DiSC_{\mathcal{P}}$	1.03%	24.81%	28.36%	17.57%	25.57%	— ” —
LSR [5]	0.25%	7.21%	8.1%	18.5%	7.66%	0.09 sec
$LSR_{\mathcal{P}}$	14.63%	32.30%	36.22%	28.08%	33.51%	— ” —
LRR [4]	2.92%	7.57%	10.17%	18.77%	8.91%	0.58 sec
$LRR_{\mathcal{P}}$	21.96%	49.56%	61.18%	61.55%	54.48%	— ” —
Euc3D [10, 12]	1.20%	7.29%	9.51%	16.04%	8.34%	0.28 sec
Our	**0.01%**	5.59%	**2.49%**	**4.98%**	**3.92%**	0.34 sec

Table 1. Results from the real dataset. Each column shows the mean segmentation error of each method over the sequences, while the last column shows the execution speed of each algorithm (all tests ran on the same computer). The stochastic methods (DiSC, LSR, LRR, Our) have been executed 100 times and their averaged results are displayed here.

5 Conclusion

We have presented a novel method for segmenting different motions from sparse 3D trajectories. Our approach uses the theory of transformation groups to derive a set of invariants of points located on the same rigid object. Because we exploit the particular structure of the problem, these invariants can be quickly calculated using a QR factorization and readily converted to a set of robust motion affinities for clustering motion trajectories together. We have evaluated our motion segmentation method using synthetic data and a new, real dataset that we have captured specifically for this work. In our comparisons against state-of-the-art motion segmentation methods, we have found that our method is more accurate than the competitors, while also being very fast. We expect that our method will perform even better if a more advanced 3D trajectory extraction algorithm is used, like [8, 9], that does not induce so much noise in the resulting 3D coordinates. In light of these results it is our conclusion that our original hypotheses that i) *motion segmentation in 3D is more accurate than in 2D*, as well as that ii) *additional invariants lead to more robust motion segmentation*, hold. The combined good performance, fast execution speed, and stability under increasing data complexity, establish our method as a very attractive and viable solution to the problem of motion segmentation of sparse 3D trajectories.

Acknowledgements

This work has been supported by Vinnova through a grant for the project iQmatic, by SSF through a grant for the project VPS, by VR through a grant for the project ETT, and through the Strategic Areas for ICT research CADICS and ELLIIT.

References

1. J. P. Costeira and T. Kanade, "A Multibody Factorization Method for Independently Moving Objects," *IJCV*, vol. 29, no. 3, 1998.
2. R. Vidal, "Subspace clustering," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 52–68, March 2011.
3. E. Elhamifar and R. Vidal, "Sparse Subspace Clustering," in *CVPR*, 2009.
4. G. Liu, Z. Lin, and Y. Yu, "Robust subspace segmentation by low-rank representation," in *ICML*, 2010.
5. C.-Y. Lu, H. Min, Z.-Q. Zhao, L. Zhu, D.-S. Huang, and S. Yan, "Robust and efficient subspace segmentation via least square regression," in *ECCV*, 2012, pp. 347–360.
6. F. Lauer and C. Schnörr, "Spectral clustering of linear subspaces for motion segmentation," in *ICCV*, 2009.
7. V. Zografos, L. Ellis, and R. Mester, "Discriminative Subspace Clustering," in *CVPR*, 2013.
8. S. Hadfield and R. Bowden, "Kinecting the dots: Particle based scene flow from depth sensors," in *ICCV*, 2011, pp. 2290–2295.
9. J. Quiroga, F. Devernay, and J. Crowley, "Scene flow by tracking in intensity and depth data," in *CVPR Workshops*, 2012.
10. D. Mateus and R. Horaud, "Spectral methods for 3-D motion segmentation of sparse scene-flow," in *WMVC*, 2007.
11. A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *NIPS*, 2001, pp. 849–856.
12. S. Perera and N. Barnes, "Maximal cliques based rigid body motion segmentation with a RGB-D camera," in *ACCV*, 2012.
13. P. Lenz, J. Ziegler, A. Geiger, and M. Roser, "Sparse scene flow segmentation for moving object detection in urban environments," in *Intelligent Vehicles Symposium*, 2011, pp. 926–932.
14. J. Klappstein, T. Vaudrey, C. Rabe, A. Wedel, and R. Klette, "Moving object segmentation using optical flow and depth information," in *PSIVT*, 2008, pp. 611–623.
15. S. Ghuffar, N. Brosch, N. Pfeifer, and M. Gelautz, "Motion segmentation in videos from time of flight cameras," in *IWSSIP*, 2012.
16. Y. Wang and S. Huang, "An effient motion segmentation algorithm for multibody RGB-D SLAM," in *Proceedings of Australasian Conference on Robotics and Automation*, 2013.
17. J. Stuckler and S. Behnke, "Efficient Dense 3D Rigid-Body Motion Segmentation in RGB-D Video," in *BMVC*, 2013.
18. A. Teichman, J. Lussier, and S. Thrun, "Learning to segment and track in RGBD," *Automation Science and Engineering, IEEE Transactions on*, vol. 10, no. 4, pp. 841–852, 2013.
19. E. Herbst, X. Ren, and D. Fox, "Object segmentation from motion with dense feature matching," in *Workshop on Semantic Perception, Mapping and Exploration (ICRA)*, 2012.
20. I. Weiss, "Geometric invariants and object recognition," *Inter*, vol. 10, no. 3, pp. 201–231, 1993.
21. L. V. Gool, T. Moons, E. Pauwels, and A. Oosterlinck, "Vision and Lie's approach to invariance," *Image and Vision Computing*, vol. 13, no. 4, pp. 259–277, 1995.
22. H. Schulz-Mirbach, *Anwendung von Invarianzprinzipien zur Merkmalgewinnung in der Mustererkennung*, ser. Fortschritt-Berichte VDI : Reihe 10, Informatik, Kommunikation ; Nr. 372. VDI-Verl. Dusseldorf, 1995, iSBN 3-18-337210-X.
23. ——, "Invariant features for gray scale images," in *DAGM*, 1995.
24. V. M. Govindu, "A tensor decomposition for geometric grouping and segmentation," in *CVPR*, vol. 1, 2005, pp. 1150–1157.

25. G. Chen and G. Lerman, "Spectral curvature clustering (SCC)," *IJCV*, vol. 81, no. 3, pp. 317–330, 2009.
26. J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IROS*, 2012.
27. L. Spinello and K. O. Arras., "People detection in RGB-D data," in *IROS*, 2011.
28. S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," in *ICCV*, 2007.