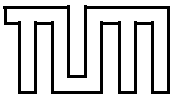
KTH Computer Science
and Communication

Motion Segmentation of RGB-D Videos via Trajectory Clustering

JULIUS ADORF

Master's Thesis at Volumental and CVAP
Supervisor at Volumental: Miroslav Kobetski
Supervisor at KTH: Stefan Carlsson
Examiner at KTH: Stefan Carlsson



Motion Segmentation of RGB-D Videos via Trajectory Clustering

JULIUS ADORF

Master's Thesis
Accredited at Technische Universität München
Supervisor at TUM: Jürgen Sturm
Examiner at TUM: Daniel Cremers

Abstract

Motion segmentation of RGB-D videos can be a first step towards object reconstruction in dynamic scenes. The objective in this thesis is to find an efficient motion segmentation method that can deal with a moving camera. To this end, we adopt a feature-based approach where keypoints in the images are tracked over time. The variation in the observed pairwise 3-d distances is used to determine which of the points move similarly. We then employ spectral clustering to group trajectories into clusters with similar motion, thereby obtaining a sparse segmentation of the dynamic objects in the scene. The results on twenty scenes from real-world datasets and simulations show that while the method needs more sophistication to segment all of them, several dynamic scenes have been successfully segmented at a processing speed of multiple frames per second.

Segmentering av rörelse i RGB-D-filmer
genom gruppering av punktbanor

Referat

Segmentering av rörelse kan vara ett första steg mot rekonstruktion av objekt i dynamiska scener. Målet i den här rapporten är att hitta en effektiv segmenteringsmetod som inte bara skiljer åt föremål som rör sig i scenen utan också hanterar videoinspelningar som görs med rörliga RGB-D-kameror. Metoden vi väljer följer distinktiva punkter i scenen genom tiden. Sedan beräknas likheten mellan de spårade punktbanorna genom att analysera parvis avstånd i 3-d. Spectral clustering används för att gruppera punktbanorna. Vi utvärderar metoden på totalt tjugo olika inspelningar därav sexton är från äkta scener och fyra är från animerade scener. Utvärderingen visar att metoden inte är tillräckligt avancerad för att kunna hantera alla scener. Samtidigt levererar metoden förväntade resultat i flera scener och behöver en bråkdel av en sekund för varje bild.

Bewegungssegmentierung von RGB-D-Videos
durch Gruppierung von Trajektorien

Zusammenfassung

Bewegungserkennung kann ein erster Schritt zur Rekonstruktion von Objekten in dynamischen Szenen sein. Das Ziel dieser Masterarbeit ist eine effiziente Methode zur Segmentierung von RGB-D-Videos zu finden, welche nicht nur sich bewegende Objekte separiert, sondern auch mit Videos zurecht kommt, die von sich bewegenden Kameras aufgenommen wurden. Zu diesem Zwecke verwenden wir eine Methode, die markante Punkte durch das Video verfolgt. Die Veränderung der Abstände zwischen jeweils zwei Punkten gibt Aufschluss darüber, welche Punkte sich nicht auf dem gleichen starren Körper befinden können. Schließlich wird spektrales Clustering angewandt, um die Punkte hinsichtlich ähnlicher Bewegung zu gruppieren. Die Methode testen wir mit zwanzig sowohl echten als auch animierten dynamischen Szenen. Die Ergebnisse zeigen, dass die Methode zwar nicht raffiniert genug ist, um mit allen Szenen zurechtzukommen, aber es trotz des einfachen Ansatzes schafft, die Bewegung in vielen Szenen mit einer Rate von mehreren Bildern pro Sekunde korrekt zu separieren.

To my parents.

Acknowledgements

While working on this thesis, I have become indebted to many people. I dedicate this thesis to my parents, as they were continuously supporting me throughout my whole studies and the thesis work. Next in line are my grandparents from whom I learned much and whose thoughts were with me wherever I was.

I wrote this thesis at Volumental, a 3-d reconstruction company in Stockholm. I especially thank Miroslav Kobetski and Rasmus Göransson for the brainstorming sessions, hints, guidance, and for the trust. Thanks to all other employees and founders at Volumental who inspired and supported me.

Many others have contributed to work. I thank João Pedro Alvito who helped me to record video sequences in the Smart Mobility Lab at KTH Royal Institute of Technology. Cudos to my brother Gabriel who introduced me to the Blender software suite. Michael Gschwandtner helped me with Blensor. Peter Gschirr and Martin Kreuzeder helped me to remove some of the edges in my writing.

I thank my student advisor Angelika Reiser at Technische Universität München (TUM) for her efforts in the double-degree program with KTH, thereby indirectly laying foundation to this project. Many students have profited from her reliable advice, and I make no exception. That I was able to write my thesis at KTH was also made possible by Jürgen Sturm and Daniel Cremers whom I thank for the supervision at TUM. I thank Stefan Carlsson for the supervision at KTH. I acknowledge that my work on this master thesis has been supported by the German Academic Exchange Service (DAAD).

Finally, my life in Sweden has been greatly enhanced by Steven Jörsäter, without whom I should have missed some of the best parts of Sweden and its culture in both summer and winter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Imagine that a computer could discover the whereabouts of all moving objects in all frames of a video with both color and depth information, and describe the motion of the objects accurately such that it becomes possible to learn the 3-d appearance of the moving objects. This would enable automatic 3-d reconstruction of moving objects in a dynamic scene. Researchers around the world are curious on how to achieve this, and there is an interest in business to make the achievements available to a wider public. Such an undertaking can benefit from recent advances in several areas. First, the availability of relatively low-cost depth sensors simplifies the acquisition of depth measurements. Second, the advances in hardware help process the data in a timely fashion. Third, open source libraries help researchers and developers to put theory into practice.

Motion segmentation in dynamic scenes remains a challenging problem despite of the advances. As compared to static scenes, any change observed in the images of a dynamic scene is not necessarily caused by camera motion. Human vision is sophisticated enough to detect small and large, dependent and independent motion of differently sized objects in the scene even if the human observer is moving on his or her own. Humans have the advantage that they may rely on a vast semantic knowledge about the world and can integrate multiple cues for grouping objects together. In contrast, computer vision is not as far in its ability to separate the sources of observed motion in such a generic way.

This thesis explores a possible first step towards the long term goal of 3-d reconstruction in dynamic scenes: the segmentation of moving objects. Of particular interest is whether the depth information provided by RGB-D cameras can be exploited for motion segmentation. Much research in motion segmentation in the last decades has focused on segmenting sequences of 2-d images. With depth sensors becoming increasingly available, it is natural to ask whether we can take advantage of the depth images in a RGB-D video for the purpose of motion segmentation. Depth as an extra dimension adds to the already large amounts of data contained in videos.

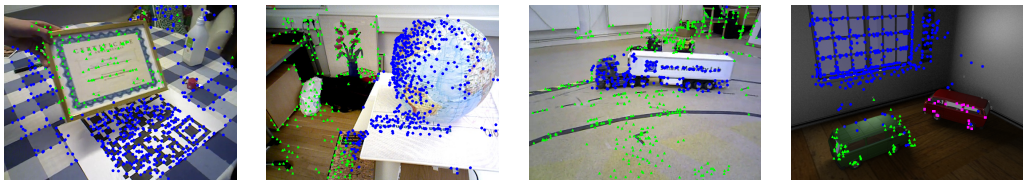


Figure 1.1: Results of motion segmentation in four dynamic scenes. Moving objects in the scene are detected. The keypoints are labeled according to which moving object they belong to.

Care has to be taken in order to make the segmentation method sufficiently fast to be interesting for real applications.

1.2 Problem statement

A *static scene* consists of a single rigid body and does not include any kind of motion. In contrast, a *dynamic scene* contains some motion. Dynamic scenes can contain different *kinds* of motion: rigid, non-rigid, and articulated motion [1]. Under rigid motion, objects do not change shape. In contrast, objects under non-rigid motion can deform. Finally, articulated motion is the composition of two *dependent* motions [1]. The dynamic scenes can be either observed by a *moving* camera or a *fixed* camera, A fixed camera means in the context of this work that the camera does not move relative to the room in which the dynamic scenes are set.

In this thesis, we address the motion segmentation problem in dynamic scenes with rigid motions, observed by a moving camera. Thus, we assume that all motion in the scene is rigid but explicitly avoid the strong assumption of a fixed camera. Given a video with color and depth images, the goal is to find a sparse segmentation of objects that move rigidly at least once in a scene. Figure 1.1 shows examples of how the results of the desired motion segmentation look like.

1.3 Selected approach

The selected approach builds upon the assumption that all motion in the dynamic scene is rigid. The method has been inspired by ideas from Brox and Malik [2]. We adopt an approach based on clustering point trajectories. First, selected keypoints are tracked in the images of the video with the help of a point tracker based on optical flow. This results in point trajectories that capture how individual points move over time. The pairwise Euclidean distances between two points over time reveal which points cannot lie on the same rigid object. We use this fact to define pairwise similarity between trajectories based on the difference of the maximum and the minimum observed distance. The framework is flexible enough to allow other definitions of similarity. In order to find clusters of points with coherent motion, the similarity graph is partitioned with spectral clustering.

1.4 Thesis outline

In the introduction, we motivated why handling dynamic scenes is beneficial for 3-d reconstruction and described the problem tackled in this thesis. In Chapter 2, we review literature related to dynamic scenes, point out the distinguishing features of our work, refine upon the problem statement given in the introduction, and describe the assumptions made in our approach in more detail. In Chapter 3, we present background theory helping the reader to understand the following chapters about the method and the evaluation. In Chapter 4, we describe the essence of the method, which is then evaluated in Chapter 5. In Chapter 6, we conclude with the key findings and suggestions for future work.

Chapter 2

Related work

The existing literature on motion segmentation is vast. This is a hint that the problem is both difficult to solve and that a solution is valuable [1]. Among the researchers that have reviewed methods on motion segmentation are Megret and DeMenthon [3], who categorize existing techniques until 2002 in a survey on video segmentation algorithms. Zappella covers research on motion segmentation until 2008 in his master thesis [4], and reviews trends in a survey [1] in 2009. These sources provide a framework for comparing the assumptions, strengths, and weaknesses of motion segmentation methods. This framework is used in the following to review mostly recent research that is not already covered by these surveys.

2.1 General approaches

The assumptions on the scene and the camera motion control the difficulty of the problem. These assumptions differ from application to application, and it is vital to understand the implications. In this thesis, we assume that the motion in the scene is rigid and that the camera might move. Furthermore, we assume that dynamic objects can appear, temporarily stop, and disappear from the scene.

In contrast, in surveillance scenarios, one can often assume a fixed camera. In such a case, the motion segmentation problem turns into the problem of subtraction of largely static background. For example, Kim et al. [5] describe a system that learns a model of the background and can then use the model to detect moving objects.

The idea of building a model of the background can also be found in applications with moving cameras. Izadi et al. [6] present an extension to the KinectFusion [7] pipeline. Here, a dense surface of the background is reconstructed from the input of a RGB-D camera. Once this model is built, dynamic objects can be detected by determining whether newly arriving data fits to the model of the background or not. This method is reviewed in more detail later in this chapter. A similar approach to motion segmentation is taken by Keller et al. [8], who use point clouds to model both the background and dynamic objects. In order to subtract the

background with moving cameras, Sheik et al. [9] fit a subspace to trajectories on the background by random sample consensus [10], thereby assuming that the background is the predominant structure in the scene. In summary, there is a group of approaches [5, 6, 7, 8, 9] that first construct a model and then identify moving objects as outliers with respect to that model. Zhou et al. [11] avoid the problem of the training phase by outlier detection by using a low-rank approximation of the video frames.

Assuming the background to be predominant in the scene, or assuming the objects to appear in a certain order might not always be viable. There are motion segmentation methods that treat all the objects in the scene equally. To these belong the trajectory grouping methods by Brox and Malik [2], and by Perera and Barnes [12]. Since the ideas for the method presented in this thesis effectively stem from these papers, they are discussed in more detail.

Besides the differences in the assumptions made on the input, there are different requirements on the output of motion segmentation. There seems to be an agreement that motion segmentation is about obtaining a pixel-wise annotation of the moving objects in each frame. Trajectory-based methods often result only in a sparse annotation of the moving objects, where the keypoints are labeled with the object they belong to [12, 13, 14, 15, 16]. There are efforts to obtain dense segmentations by increasing the density of the tracked points [2, 17, 18]. If this is not enough, "densification" [3] can be done as a post-processing step [19]. In contrast, statistical approaches can lead directly to a dense segmentation [1, 20].

So far, different assumptions on the input and requirements on the output of motion segmentation have been treated. Both the assumptions and the requirements are determined by the intended application of the motion segmentation algorithm. When these are clear, it is time to have a closer look on *how* motion can be segmented in various settings. For this purpose, Megret and DeMenthon [3] provide a way to categorize the existing approaches. Particularly helpful appears the division of methods into three bins: (1) methods that focus on segmenting single frames first and care about time later, (2) methods that explore time first and obtain a spatial segmentation later, and (3) methods that jointly estimate the motion in both time and space.

Spatial priority

Focusing on the frame-wise motion first is a natural extension to image segmentation. Given two successive frames in a video, the question becomes on how to find regions of coherent motion between these frames. For the robotic application of singulating objects in cluttered scenes [21], this is already sufficient for segmenting objects based on motion cues [22, 23], especially if the robot is allowed to interact with the scene [24, 25, 26]. Commonly, such approaches use or relate to optical flow [27], scene flow [23, 28, 29], or employ feature matching [26, 30] in order to detect the changes between pairs of images. However, these approaches offer no natural way to maintain a temporally consistent segmentation.

Temporal priority

An alternative is to follow points over time first, and decide how to cluster the points into groups with coherent motion in a later stage. This is the trajectory grouping category of methods [3]. Here, trajectories describe local motion over a long period. The trajectories can be found with optical flow [17, 18, 31, 32] or by matching feature descriptors [33, 34, 35, 36]. The long-term observations seem to provide a natural way to build temporal consistency into the method right from the beginning. The disadvantage is that drift in the tracker can cause these trajectories to cross motion boundaries, meaning that they are susceptible to become corrupted over time [37]. We have adopted the trajectory grouping approach in this work, building in particular on the ideas of Brox and Malik [2], who use spectral clustering to group trajectories based on pairwise similarities. Prior to their work, many other methods have been developed on the idea of trajectory grouping. Assuming an affine camera, the trajectories of points belonging to the same rigid motion lie in a linear subspace [38] such that finding these subspaces solves the motion segmentation problem. The local subspace algorithm of Yan and Pollefeys [13] is a particular method that is treated in more detail in this chapter. The benchmark by Tron and Vidal [39] summarizes a few of these methods, and Zappella [1, 4, 40] covers them well. These methods are mathematically elegant but have the weakness that they usually require complete trajectories. With both moving objects and a moving camera, this requirement limits the use cases considerably. However, there are efforts to overcome the problem of incomplete trajectories [15, 41].

Joint segmentation

Finally, there is the category of methods that segment motion jointly in space and time. Statistical methods seem to be suitable, as they allow to combine the unknowns with the prior knowledge about spatial and temporal constraints in a single framework. Cremers and Soatto [20] formulate motion segmentation as a problem of Bayesian inference, which is solved with variational methods. They represent the motion boundaries either with splines or level sets. A recent method by Schoenemann and Cremers [42] segments motion into layers such that a cost for encoding the video is minimized.

2.2 Exemplary approaches

2.2.1 Clustering of long-term trajectories

Brox and Malik [2] cluster long-term point trajectories in 2-d videos. Their basic idea is to detect motion between pairs of points over time, arrange the collected evidence in a similarity graph, and then use spectral clustering to partition the similarity graph into clusters of similarly moving points. Their work is relevant for this thesis, as the approach allows for a moving camera, handles missing data, aims

for temporally consistent segmentations, and has reasonable complexity with regard to the implementation.

The authors do not assume a static camera. However, they assume that the objects are moving mostly translational with respect to the camera. The assumption is necessary because motion on the image plane is ambiguous. Follow-up work by Ochs and Brox [43] addresses this issue by considering triplets instead of pairs, though at the expense of increased computational costs [44]. This gives rise to the question whether it is possible to extend the approach to 3-d trajectories to get rid of this ambiguity. This thesis differs from their work mostly through the use of 3-d data and a different definition of similarity.

The graph-partitioning approach to clustering handles incomplete trajectories naturally. Non-overlapping trajectories can still be related to each other via transitive links in the similarity graph. Brox and Malik [2] do not report how much missing data can be handled. This is an important question as it influences the choice of the keypoint tracker. They use the large displacement optical flow tracker on the GPU by Sundaram et al. [17], which has been reported to improve in accuracy and quantity over a GPU-based version of the Kanade-Lucas-Tomasi tracker [45]. Yet, the choice of a more advanced point tracker comes at the expense of computational costs. Sundaram et al. [17] report a runtime of 3-4 seconds per frame in 2010. For the longest of the videos used in this thesis, extrapolation shows that tracking alone would accumulate to an hour on the same machine and the same image resolution at 640×480 pixels. Likely due to better hardware, Ochs et al. [44] report a runtime of less than two seconds in 2013. Despite of the advances in hardware, a fast, serial point tracker based on the OpenCV library [46] is used in this work instead because it runs at about half real-time frame rate on our machine.

The trajectories provide temporal links between the video frames. Brox and Malik [2] claim that the ability to obtain "temporally consistent segmentations" is one of the main benefits of the approach. With regard to the survey of Megret and DeMenthon [3], their method falls into the category of trajectory grouping approaches that gather information in time first, and segment in space later. Analyzing along the temporal dimension first before taking further steps makes the method offline, as it needs to see all video frames. Brox and Malik [2] motivate their work by arguing that the long-term trajectories reveal more about the moving objects in the scene than the flow between two scenes. The same position is taken in this thesis.

The way we define similarity between trajectories differs from the work by Brox and Malik [2]. They concentrate on identifying the instant where the motion between points seen on the image plane is maximum. We experiment with both the maximum overall difference in point distance, and alternatively the variance as a measure for dissimilarity, formulated in a generic framework where alternative definitions can be plugged in. Which method works best in which scenarios has to the best of our knowledge not been treated in depth yet and remains an open question.

Follow-up work by Ochs et al. [19, 44] explores the method further. It includes a densification step [3] for obtaining pixel-wise motion segmentation rather than the sparse segmentation obtained by labeling the trajectories. The densification

step is particularly interesting for possible 3-d object reconstruction after motion segmentation because it identifies the regions in the image and in the depth data that shall be included in the 3-d model and its texture. Fragkiadaki et al. [47] treat the discontinuities between clusters explicitly and obtain a dense segmentation from the trajectory clusters via Gabriel graphs [48].

2.2.2 Clustering trajectories with maximal cliques

Perera and Barnes [12] build upon the fact that *none* of the distances between points on a rigid object can ever change. They group trajectories by finding maximal cliques in a thresholded similarity graph constructed from the sample standard deviation between pairs of points.

The maximal cliques correspond to groups of points that are compatible with a rigid motion hypothesis. They solve the maximum coverage problem in order to determine the final segmentation from the, not necessarily disjoint, maximal cliques. The problems of enumerating all maximal cliques in a graph, and the maximum coverage problem are NP-hard [49, 50].

In contrast to many existing motion segmentation algorithms, Perera and Barnes [12] use RGB-D videos as input data. They evaluate their method on synthetic data and real datasets. However, the latter consists only of image sequences with a handful of images with less than 150 keypoints. The RGB-D videos targeted in this thesis contain fifty or more frames, and the quantity of keypoints needs to be larger. While theoretically elegant, it is therefore not clear to see how this method scales to the problem treated in this thesis.

Perera and Barnes [12] also suggest to use the sample standard deviation in order to summarize changes of distances between pairs of 3-d points. These aspects make their research relevant for this thesis, claiming that 3-d data can be used for motion segmentation. We generally follow their notation where possible.

There are many parallels between the maximal clique based approach in [12] and the spectral clustering approach in [2]. Both construct a similarity graph. The maximal clique based approach requires an early decision about which edges in the graph link points on the same rigid object. Hence, noise is addressed early in the algorithm while the uncertain information is kept in spectral clustering almost until the end. The approach taken in this thesis can be seen as a fusion of the ideas in [2] and [12]. The construction of the similarity graph from 3-d trajectories follows [12] in certain aspects, whereas the grouping step is performed via spectral clustering in a similar way as done in [2].

2.2.3 Clustering trajectories by local subspace affinity

The local subspace affinity (LSA) method by Yan and Pollefeys [13] exploits the fact that trajectories of points on a rigidly moving object lie in an up to four-dimensional linear subspace under affine projection [38]. The trajectories are then grouped by

estimating the subspace for each trajectory locally. Spectral clustering partitions the similarity graph based on the principal angles between a pair of estimated subspaces.

LSA belongs to a class of methods that make use of the measurement matrix $\mathrm{W} \in \mathbb{R}^{2F \times N}$ (see [15, 38]). This matrix consists of the stacked 2-d coordinates u, v of all N points in all F frames.

$$\mathrm{W} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1N} \\ v_{11} & v_{12} & \cdots & v_{1N} \\ \vdots & \vdots & \vdots & \vdots \\ u_{F1} & u_{F2} & \cdots & u_{FN} \\ v_{F1} & v_{F2} & \cdots & v_{FN} \end{pmatrix} \in \mathbb{R}^{2F \times N} \tag{2.1}$$

This matrix is formed by projections of the 3-d trajectories with an affine camera. The idea now is to find trajectories that lie in the same subspace. To this end, LSA estimates the matrix rank K and projects each of the N columns onto the K-dimensional unit hypersphere with the help of singular value decomposition. The linear subspace for each trajectory is estimated from the local neighborhood on the sphere. Finally, Yan and Pollefeys [13] use principal angles between the estimated linear subspaces for defining similarity between the subspaces. The similarity graph is partitioned via spectral clustering.

While mathematically elegant, the downside of LSA is that it does not provide a built-in mechanism to treat incomplete trajectories. This is a too strong limitation given the RGB-D videos considered in this thesis. Also, estimating the rank of matrix K and the dimensions of the linear subspaces has been reported as difficult [39]. Local subspace affinity is studied in more detail in the master thesis by Zappella [4], which provides good references to related work on the subject.

2.2.4 Outlier-based segmentation with KinectFusion

So far, we have presented approaches that work on sparse to semi-dense trajectories. KinectFusion by Newcombe et al. [7] is a depth-based real-time approach to the simultaneous localization and mapping (SLAM) problem. Depth readings from a depth sensor are integrated frame by frame into a global model of the surface.

Izadi et al. [6] extend the system to deal with dynamic scenes in order to support interaction. This involves a user interacting with objects in front of the camera. They also state the 3-d reconstruction of a single object as the goal, motion being the cue for segmentation. Hence, their work is relevant to this thesis.

The extension to KinectFusion is based on the assumptions that the background is the dominant structure in the scene, and that parts of the background have already been integrated into the surface model prior to interaction. The idea behind these assumptions is that the camera motion can still be recovered from the dominant background and that dynamic objects reveal themselves as outlier data when fitted to the global model of the surface. In the interactive scenario, users might not only interact with the scene but the system might interact with the user. This

allows for giving the user feedback and instructions on how to make sure that these assumptions hold. These assumptions are not made in this thesis, giving all rigid structures equal treatment and imposing no specific order in which the objects have to appear in RGB-D videos.

2.2.5 Object-centered reconstruction

Shin et al. [51] introduce a bottom-up approach targeted solely at reconstructing moving objects in a dynamic scene. Their idea is based on co-recognizing objects that are common in a set of unordered images by matching image patches between each image pair. The correspondences are then used to obtain a dense segmentation of the objects in the images prior to 3-d reconstruction of the method.

Unfortunately, the method scales quadratically with the number of frames. The reported running times of the reference implementation are too long with respect to the practical orientation of this thesis.

2.3 Benchmarks

The field of RGB-D video segmentation is relatively new and seems to lack standard, large-scale benchmark datasets. No suitable public benchmark could be found that provides RGB-D videos with ground truth segmentation of each moving object in a dynamic scene. In the following, three publicly available benchmarks are presented, but none of them is really suitable for this thesis.

2.3.1 TUM RGB-D Benchmark

The TUM RGB-D Benchmark dataset [52, 53, 54] is a collection of static and dynamic scenes recorded with RGB-D cameras. The benchmark is intended for the evaluation of SLAM systems. Each recording provides color and depth images. The datasets are annotated with the ground truth camera path relative to the background in the scene. The dynamic scenes contain motion such as people moving. The benchmark targets researchers who want to test the camera tracking and mapping capabilities of a SLAM system. There is no pixel-wise annotation of the moving objects in the image sequences, which limits the usefulness of the dataset in the context of motion segmentation, at least as long as the motion segmentation algorithm is not embedded into a system for robust camera tracking.

2.3.2 Hopkins 155 Dataset

The Hopkins 155 Dataset [39] provides 155 videos together with point trajectories. Most of scenes consist of rigidly moving checkerboard objects, recorded with handheld cameras. There are also scenes with non-rigid motion and articulated motion. The trajectories are labeled with ground truth according to which dynamic object

they belong to. However, the Hopkins 155 Dataset does not contain outlier or incomplete trajectories, has less than 500 trajectories per image sequence, and does not provide any depth data as input, which makes it unsuitable for the evaluation of RGB-D video segmentation methods. An update to the dataset adds the capability to insert synthetic outliers into the dataset, and provides 16 sequences with incomplete trajectories and outliers from [15, 55].

2.3.3 Freiburg-Berkeley Motion Segmentation Dataset

The Freiburg-Berkeley Motion Segmentation Dataset [2, 44] contains 59 videos with pixel-wise ground truth. It features mostly non-rigid motion but includes some of the car sequences from the Hopkins 155 Dataset. As with the Hopkins 155 Dataset, no depth data is included. It is thus also not suitable for the evaluation in this thesis.

Chapter 3

Background

In this chapter, we present concepts that the method builds upon. Readers familiar with the topics discussed in this section may skip to the next chapter. In Section 3.1, we list the depth sensors used in this work for recording RGB-D videos, and continue to the tracking of keypoints in Section 3.2. In Section 3.3, we provide the basic terminology and notation of graph theory for convenience before giving an introduction to spectral clustering in Section 3.4.

3.1 RGB-D cameras

The RGB-D sensors used in this work are the Kinect for Windows (Table 3.1, [56]) and the Asus Xtion Pro Live (Table 3.2, [57]). Both cameras are based on the same sensor technology and provide registered depth and color images. They are active sensors based on structured light. This means that they project a near-infrared pattern onto the surface and reconstruct depth from the observations of an infrared sensor. While the cameras are relatively cheap, they also produce a considerable amount of noise. A particular property of the cameras is that the noise in the depth measurements increases quadratically with the depth [58].

Range	0.8 m - 4 m (default) or 0.4 m - 3 m (near)
Field of view	57° horizontal, 43° vertical

Table 3.1: Specifications of the Kinect for Windows RGB-D camera.

Range	0.8 m - 3.5 m
Field of view	58° horizontal, 45° vertical

Table 3.2: Specifications of the Asus Xtion Pro Live RGB-D camera.

3.2 Point trackers

Point trackers follow scene points over time. They essentially solve a correspondence problem: the task of a point tracker is to identify the coordinates of a point in each camera image. The output of a point tracker is a trajectory for each point. The trajectory describes the coordinates of the point in the frames of the video.

One class of approaches to point tracking is based on optical flow. The optical flow describes the displacements of points between successive images. These methods assume that the scene motion between the two images is somewhat limited, such that corresponding points can be found close to each other in successive images. An example for such a flow-based point tracker is the pyramidal implementation [32] of the Lucas-Kanade method [31] available in OpenCV [46, 59]. The pyramidal implementation allows the tracker to deal with larger displacements than it is possible with the fixed window in the plain Lucas-Kanade method. Recent approaches estimate large-displacement optical flow using variational methods [18, 60], which can be used to compute point trajectories with the help of parallelization on the GPU [17].

An alternative approach is to extract keypoints from two images, describe the keypoints based on their local appearance, and use nearest neighbor search to match the keypoints. Research on keypoint extraction and keypoint descriptors is vast, for a survey see [36]. Most well-known are probably the SIFT [33] features and their application to panorama stitching [61], where images with potentially large displacements need to be registered. Not all image registration techniques are applicable in motion segmentation, though. If the registration method assumes a static scene, it can exploit the constraints of multiple view geometry [62]. In dynamic scenes, methods can only assume locally coherent motion.

3.3 Graph theory

Graphs are commonly used in this thesis both in the theoretical discussion as well as in the implementation. There is plenty of introductory and advanced literature about graphs in computer science and mathematics, see [49, 63, 64]. This section lists the definitions used in this work for convenience. All discussed graphs are undirected, and either weighted or unweighted.

A simple undirected graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of undirected edges $\mathcal{E}$. Two vertices $i, j \in \mathcal{V}$ are *adjacent* if and only if they are connected by edge $\{i, j\} \in \mathcal{E}$. An edge $\{i, j\} \in \mathcal{E}$ is called *incident* with the vertices $i, j \in \mathcal{V}$. The graph contains no loops, i.e. $\{i, i\} \notin \mathcal{E}$. The graph can be represented by a symmetric adjacency matrix $E \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ whose elements $e_{ij} = e_{ji}$ indicate whether the vertices i, j are adjacent.

A weighted undirected graph assigns a weight $w_{ij} \in \mathbb{R}$ to each edge. Two vertices $i, j \in \mathcal{V}$ in a weighted undirected graph are called adjacent if and only if the weight w_{ij} is non-zero. The weighted undirected graph can be represented by a symmetric

adjacency matrix $W \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ with elements $w_{ij} = w_{ji}$. The degree d_i of a vertex $i \in \mathcal{V}$ is computed by the summing the weights of all incident edges.

$$d_i = \sum_{j=1}^{|\mathcal{V}|} w_{ij} \tag{3.1}$$

The *volume* [65] of a set of vertices $\mathcal{A} \subseteq \mathcal{V}$ is the sum over all vertex degrees.

$$\mathrm{vol}(\mathcal{A}) = \sum_{i\in\mathcal{A}} d_i \tag{3.2}$$

The *cut* describes the sum of weights of all edges connecting disjoint vertex sets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$, and is a measure of how strongly subgraphs are connected.

$$\mathrm{cut}(\mathcal{A}, \mathcal{B}) = \sum_{i\in\mathcal{A}, j\in\mathcal{B}} s_{ij} \tag{3.3}$$

3.4 Spectral clustering

Spectral clustering comprises a class of methods that use the eigenvectors of a modified similarity matrix for clustering. We will approach these methods from the perspective of partitioning similarity graphs. Spectral clustering cuts a graph by embedding data points in a low-dimensional space where clusters can be found with a simpler clustering method such as k-means. As such, the term "spectral clustering" is slightly misleading because it does not assign labels without the help of a clustering algorithm. Rather, spectral clustering generates features that serve as input for a clustering algorithm.

Spectral clustering methods have been treated extensively in literature. The interested reader is referred to the survey by Luxburg [65] for a more comprehensive tutorial on various spectral clustering techniques. Other standard references are the articles by Shi and Malik [66] and Ng et al. [67]. Spectral graph theory is treated by Chung [68]. Spectral clustering is explained in context with other clustering methods in the survey by Filippone et al. [69]. There is a relation to weighted kernel PCA by Alzate and Suykens [70]. In general, spectral clustering can be interpreted and derived in different ways. The remainder of this section concentrates on a particular variant called *normalized spectral clustering*, and highlights the aspects that make spectral clustering attractive for our problem.

The key idea is to translate the clustering problem into a graph problem. Let us assume that we have a dataset with N data points. How these data points look like and how they are described is irrelevant. In order to be able to use spectral clustering, we only need a *similarity measure* that assigns a non-negative similarity to each pair of data points. This makes it possible to define a *similarity graph*, in which vertices represent data points and each edge is weighted by the similarity between the incident vertices. The similarity graph is represented by its weighted

adjacency matrix $\mathrm{S} \in \mathbb{R}^{N \times N}$ with entries $s_{ij} \geq 0$, which we call *similarity matrix* whenever convenient.

The clustering problem can now be formulated as the problem of partitioning the similarity graph. Let us consider the case of two clusters first. In line with the usual clustering objectives, the goal is to divide the graph such that the vertices within the same cluster are connected by edges with large weights (high intra-cluster similarity), and such that the weights of the edges in the cut are small (low inter-cluster similarity). Additionally, we would like to balance the number of data points in each cluster in order to avoid clusters consisting of single outliers in the data. The *normalized cut* [71] is desirable because it optimizes both for a low value of the cut and balanced clusters. For a partition $\{\mathcal{A}, \mathcal{B}\}$ of the vertex set $\mathcal{V}$, the normalized cut is defined as

$$\mathrm{Ncut}(\mathcal{A}, \mathcal{B}) = \mathrm{cut}(\mathcal{A}, \mathcal{B}) \left(\frac{1}{\mathrm{vol}(\mathcal{A})} + \frac{1}{\mathrm{vol}(\mathcal{B})} \right) \tag{3.4}$$

In order to achieve a low value for $\mathrm{Ncut}(\mathcal{A}, \mathcal{B})$, the term $\mathrm{cut}(\mathcal{A}, \mathcal{B})$, which captures the notion of inter-cluster similarity, needs to be minimized, and the terms $\mathrm{vol}(\mathcal{A})$, $\mathrm{vol}(\mathcal{B})$ need to be maximized at the same time. Unfortunately, finding the normalized cut is NP-complete [71]. A brute-force solution is infeasible given that there are $\left\{ {N \atop 2} \right\} = \sum_{n=1}^{N-1} \binom{N}{n} = 2^{N-1} - 1$ possibilities to partition a set into two subsets [72]. For problem instances with $N > 2000$ as encountered in this thesis, there is a more than 600-digit number of possibilities to choose from. Fortunately, by relaxing the problem into the real domain it turns out that the normalized cut can be approximated in polynomial time by finding the eigenvectors of a modified similarity matrix.

The first step is to form the *unnormalized graph Laplacian* $\mathrm{L} \in \mathbb{R}^{N \times N}$ (see [65])

$$\mathrm{L} = \mathrm{D} - \mathrm{S} \tag{3.5}$$

where $\mathrm{D} \in \mathbb{R}^{N \times N}$ is the degree matrix of the similarity graph containing the vertex degrees in the diagonal, i.e.

$$\mathrm{D}_{ij} = \begin{cases} \sum_{k=1}^{N} s_{ik} & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} . \tag{3.6}$$

The next step in approximating the normalized cut is to solve the generalized eigenproblem

$$\mathrm{L}\mathbf{f} = \lambda \mathrm{D}\mathbf{f} \quad . \tag{3.7}$$

Let $\lambda_1, \lambda_2, \ldots, \lambda_N$ denote the eigenvalues with $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$ and let $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N \in \mathbb{R}^N$ denote the corresponding eigenvectors. None of the eigenvalues is negative. The smallest eigenvalue λ_1 is always zero and the corresponding eigenvector $\mathbf{f}_1$ is the vector of ones. Hence, the first eigenpair is of no interest. The

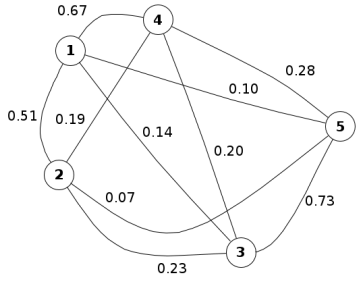


Figure 3.1: A toy similarity graph.

second eigenvector however reveals important information about the cluster structure in the data, playing the role of a real-valued cluster indicator. This vector can be turned into a hard labeling by using k-means which boils down to thresholding. Equation 3.8 shows the graph Laplacian for the toy graph in Figure 3.1 and the second generalized eigenvector (example taken from [73]).

$$\mathrm{L}_{\text{toy}} = \begin{pmatrix} 1.41 & -0.51 & -0.14 & -0.67 & -0.10 \\ -0.51 & 1.00 & -0.23 & -0.19 & -0.07 \\ -0.14 & -0.23 & 1.29 & -0.20 & -0.73 \\ -0.67 & -0.19 & -0.20 & 1.34 & -0.28 \\ -0.10 & -0.07 & -0.73 & -0.28 & 1.18 \end{pmatrix}, \mathbf{f}_{\text{toy},2} \approx \begin{pmatrix} 0.44 \\ 0.41 \\ -0.50 \\ 0.23 \\ -0.58 \end{pmatrix} \tag{3.8}$$

Moving on to the general case of k clusters, there are several propositions on how to do so-called multiway spectral clustering. One of them is to take $k-1$ eigenvectors, and stack them horizontally in a matrix $\mathrm{F} = [\mathbf{f}_2\ \mathbf{f}_3\ \cdots\ \mathbf{f}_k]$. The i-th row in this matrix serves as a feature vector for the i-th data point and the matrix can be passed to k-means for label assignment. This works because the eigenvectors beyond the second eigenvector contain relevant information about the cluster structure in the data. It is illustrative to rewrite Equation 3.7 compactly with the eigenvalues arranged in a diagonal matrix $\Lambda = \mathrm{diag}(\lambda_2, \lambda_3, \ldots, \lambda_k)$ as

$$\mathrm{LF} = \mathrm{D}\mathrm{F}\Lambda \quad . \tag{3.9}$$

Eigenvectors are defined only up to scale. This can be neglected in the case of two clusters. In the multiway case, each column of of matrix F can be rescaled independently with Equation 3.9 still being satisfied. However, any k-means algorithm based on Euclidean distance will be affected by the scale of the eigenvectors as these contain the values along one dimension. Thus, scale matters in the case of multiple clusters.

Spectral clustering as such is a general clustering technique that has been applied to various problems in different fields. While implementing a basic version of spectral clustering with k-means can be done in a few lines of Matlab code (Listing 1), there are more questions to be solved in a practical implementation.

```
function labels = spectral_clustering_with_kmeans(S, num_clusters)
    D = diag(sum(S, 2));
    L = D - S;
    [F, ~] = eig(L, D);
    labels = kmeans(F(:, 2:(num_clusters+1)), num_clusters);
end
```

Listing 1: A basic implementation of spectral clustering with k-means in Matlab. Yet, the simplicity of the code is deceiving. It is non-trivial to set up the similarity matrix S. The number of clusters should also be determined in the procedure rather than specified by the user.

Chapter 4

Method

In this chapter, we describe the implemented motion segmentation system. The goal of the system is to segment the moving objects in each frame of a 3-d video. A conceptual overview of the method is given in Section 4.1. The method consists of roughly three consecutive steps (Figure 4.1). First, points are tracked in the color images in order to follow points in the scene through time and space (Section 4.2). The trajectories are compared pairwise in order to determine which trajectories move similarly and are thus likely to belong to the same rigid object (Section 4.3). This analysis results in a similarity graph which is then partitioned into several components that correspond to the moving objects (Section 4.4).

4.1 Concept

The proposed method relies at its core on the fact that two scene points that move closer to each other or away from each other cannot lie on the same rigid object. Hence, observing the distance between two points gives a clue about which points must lie on different rigid objects. The key idea is to group the points based on the observation of pairwise distances over time. In practice, there are two major difficulties that need to be tackled: noise and missing data. Noise in the observations might suggest motion when there is none. Partial observations might lead to motion going undetected.

4.1.1 Trajectories

Points can be tracked in order to observe how the distance between them changes over time. Point tracking results in trajectories describing the motion of the points through the scene. In the ideal case, the trajectories span all the frames in the video. However, in many scenes observed by a single camera, the trajectories can be *incomplete* [2, 15] and span only a certain part of the video. This is problematic because we can only measure distances between two points if the trajectories overlap in time. Moreover, only trajectories that overlap in at least two frames are of

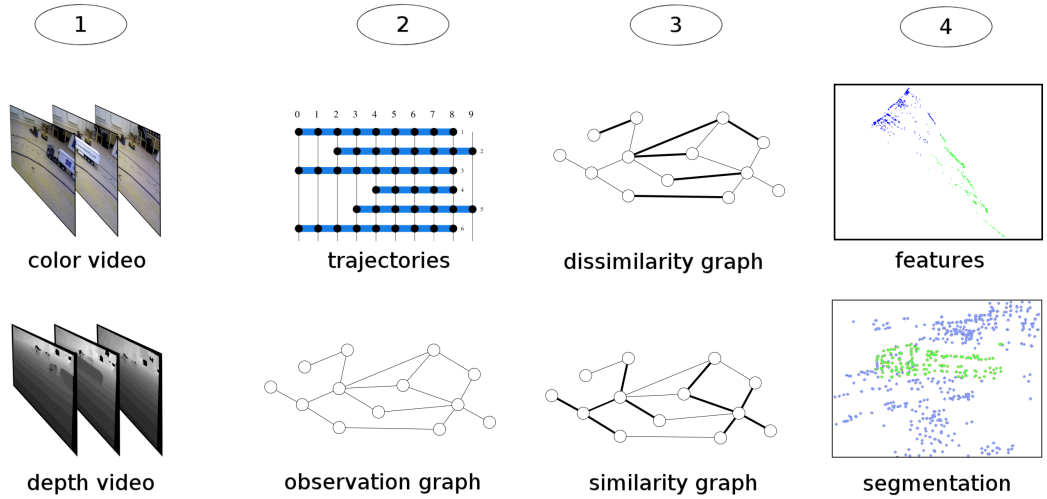


Figure 4.1: Overview of the motion segmentation algorithm. The color and depth images from a RGB-D video are the input data (1). Points are tracked on the color images in order to obtain trajectories. The observation graph indicates for each pair of trajectories whether the trajectories overlap in time (2). We assign high dissimilarity to trajectory pairs if the distances between the tracked points were observed to change. A kernel turns the dissimilarity graph into a similarity graph (3). We use spectral clustering to partition the similarity graph. The eigenvectors of a modified similarity matrix form features. k-means obtains labels from these features, providing the final motion segmentation (4).

interest because we want to track change in the distance. This can be formalized in an *observation graph* where vertices represent trajectories and two vertices are adjacent if and only if the trajectories overlap in at least two frames.

4.1.2 Scenarios without noise and full observation

If all trajectories are complete, then the observation graph is complete. The converse does not hold. Let us assume for now that the distances of all tracked points can be observed at all times, and that the observations are noiseless. It is illustrative to see how the motion segmentation problem can be solved in such a simplified scenario. Let us start by defining another undirected graph on the trajectories, the *rigidity graph*. In the rigidity graph, two vertices are adjacent if and only if the observed distance between the points remains constant. Note that, as illustrated by Figure 4.2, constant distance does not necessarily imply that the two points are on the same rigid object. Constant distance just means that there exists a rigid motion that explains the motion of both points.

In order to find groups of points that follow the same rigid motion, we can search for maximal cliques in the graph [12]. The maximal cliques are not necessarily

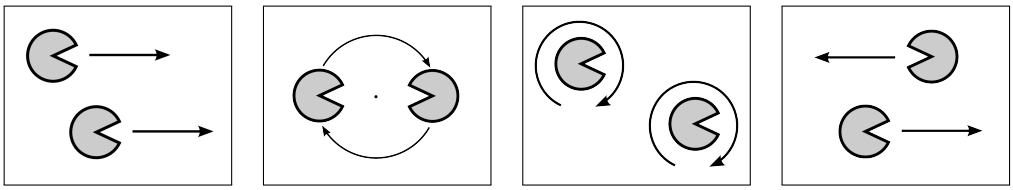


Figure 4.2: Different types of motion of two rigid objects.

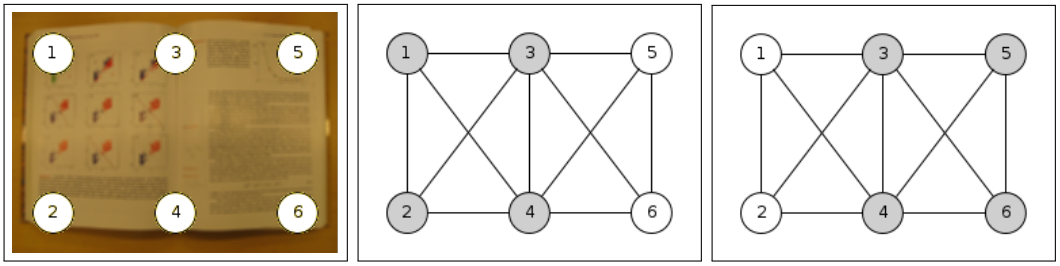


Figure 4.3: The rigidity graph for an articulated object has intersecting maximal cliques. In the depicted case, the intersection is formed by points 3 and 4 on the axis.

disjoint, not even in the ideal noiseless case with complete trajectories. An example for this is given in Figure 4.3. Six keypoints are selected on a book: four on the corners of the book, and two on the axis where the book can be folded together. The two diagrams in Figure 4.3 (right) show the two maximal cliques contained in the rigidity graph of a scene where the book has been folded at least once. This means that finding the maximal cliques alone is not sufficient to partition the trajectories into groups undergoing the same rigid motion, even when there is no noise.

4.1.3 Scenarios with noise and partial observation

In reality, we have to deal with noisy scenarios and incomplete trajectories. Perera and Barnes [12] treat noise in their maximal clique based approach to motion segmentation. They construct the rigidity graph by only including edges if the standard deviation of the pairwise distances remains below a certain threshold. We avoid the thresholding and choose a method that does not force us to threshold statistics on the pairwise distances before partitioning the trajectories.

Rather, we keep noisy information under the assumption that the signal might still be strong enough to be valuable for clustering the trajectories. This is one of the reasons why we choose to partition a *similarity graph*, such as it is done by Brox and Malik [2]. Again, vertices represent trajectories but in contrast to the observation graph and the rigidity graph defined earlier, the similarity graph is an undirected weighted graph. Edges with large weights denote that two points are moving similarly. Then subgraphs with large weight correspond to groups of trajectories that move similarly. This is illustrated in Figure 4.4. We use spectral

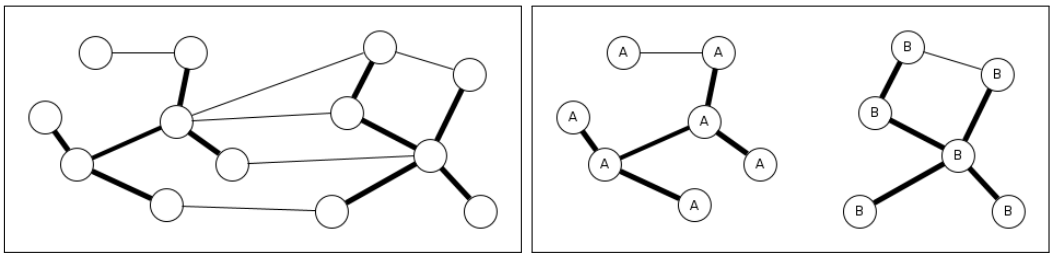


Figure 4.4: Partitioning the similarity graph yields clusters of points that move similarly. Bold edges denote strong similarity. The connected similarity graph is partitioned into two clusters by finding a cut that ideally leads to clusters with high intra-cluster similarity, low inter-cluster similarity, and clusters of somewhat similar size.

clustering (see Section 3.4) in order to find these groups.

4.1.4 Towards pose estimation

Keypoints on the same rigid object can be used to estimate its 6 d.o.f. motion separately of the other objects in the scene. The clustering of points is not an end by itself. Rather, the segmentation should enable us to use the multiple view constraints for estimating the motion of single rigid objects. The remaining part of this chapter describes the pipeline which takes a RGB-D video as input and produces a sparse segmentation of the dynamic objects as output. The pose estimation itself has not been tackled and remains future work.

4.2 Point tracking

Point tracking is the first step in the pipeline. As stated in the concept (Section 4.1), the idea is to track points, construct a similarity graph for the trajectories, and segment the trajectories. The tracking of points constitutes an important step because all further tasks depend on the trajectories. The point tracker has kindly been provided by Volumental for this purpose. The implementation of the point tracker is therefore not part of this thesis.

The tracker builds upon the Lucas-Kanade optical flow implemented in the OpenCV framework [32, 46, 59]. It is relatively fast and is able to produce enough keypoints to cover larger parts of the scene. This is important compared to visual SLAM algorithms for static scenes that only need enough keypoints to recover the camera motion.

The result of the point tracking is a set of trajectories $\{\mathbf{x}_i\}_{i=1}^{N}$ where each trajectory is a sequence of image coordinates.

$$\mathbf{x}_i = (\mathbf{x}_i^{a_i}, \mathbf{x}_i^{a_i+1}, \ldots, \mathbf{x}_i^{a_i+l_i-1}) \tag{4.1}$$

The first frame of trajectory $\mathbf{x}_i$ is denoted by a_i and the length of the trajectory by l_i. The point tracker used in this thesis creates trajectories describing a point in l_i successive frames, as indicated by Equation 4.1. This simplifies reasoning, but is not necessarily true for other point trackers, and not a necessary prerequisite for the method either. Assuming that the color images and the depth images are registered, we can obtain the trajectory $\mathbf{X}_i = (\mathbf{X}_i^{a_i}, \mathbf{X}_i^{a_i+1}, \ldots, \mathbf{X}_i^{a_i+l_i-1})$ in camera coordinates.

4.3 Trajectory statistics

The next step is to define pairwise dissimilarity between trajectories. This is a prerequisite for constructing the similarity graph as described later in Section 4.4.1. Intuitively, the more change observed in the distance between a pair of points, the higher dissimilarity should be assigned to them. We formalize this intuition in the following.

4.3.1 Statistics of scene distances

Let ρ_{ij}^t denote the scene distance between two points with 3-d coordinates $\mathbf{X}_i^t$ and $\mathbf{X}_j^t$ in frame t, following the notation in [12].

$$\rho_{ij}^t = \left|\left|\mathbf{X}_i^t - \mathbf{X}_j^t\right|\right|_2 \tag{4.2}$$

By measuring the scene distance between two points at all observable times $t_1, t_2, \ldots, t_n \in \mathcal{T}_{ij}$, we obtain a time series $(\rho_{ij}^{t_1}, \rho_{ij}^{t_2}, \ldots, \rho_{ij}^{t_n})$. Defining dissimilarity can be seen as finding an appropriate statistic for this time series. At this time, it would be a good idea to assume a certain statistical model for the time series. Instead, let us just assume that the distances ρ_{ij}^t are independent and identically distributed. Then, we can say that all distances in the time series are distributed like the random variable ρ_{ij}.

The idea is to use statistics of the variable ρ_{ij} that summarize variability in the series of distance measurements. There are many statistics available for measuring statistical dispersion. Any of them might be considered for computing dissimilarity. In this work, we are exploring the range and the variance.

The range of the sample is the difference between the largest and the smallest value in the sample[1]. If the range is large, then the points were once observed to be close together at one point of time, and far away of each other at another point of time. This should be a strong indicator that they cannot belong to the same rigid body. Intuitively, the range "memorizes" the largest motion seen. Conversely, if two points never move, and there is little noise in the measurements, then the range is close to zero. However, it is possible that a single outlier dominates the range. The sample range is computed by

[1]For an alternate definition in statistics for the range, see Evans et al. [74].

$$\hat{\mathrm{r}}\left(\rho_{ij}\right) = \max_t \rho_{ij}^t - \min_t \rho_{ij}^t \tag{4.3}$$

Alternatively, the variance can serve as a measure of dissimilarity between trajectories. The variance is more robust towards outliers than the range but it becomes smaller if two points remain static for a long time after having moved once. Therefore, the variance assigns higher dissimilarity to points that move constantly throughout the observed period. The (uncorrected) sample variance of the distances between two points is

$$\hat{\mathrm{var}}\left(\rho_{ij}\right) = \frac{1}{|\mathcal{T}_{ij}|} \sum_t \left(\rho_{ij}^t - \hat{\mu}\left(\rho_{ij}\right)\right)^2 . \tag{4.4}$$

Perera and Barnes [12] use the sample standard deviation instead. The use of statistical tools shall not hide the fact that these measures are defined in a pragmatic rather than principled manner. However, the use of statistical terms is beneficial for generating ideas, predicting the effects of measures, and the analysis in the evaluation.

4.3.2 Statistics of image distances

Similar to the definitions of dissimilarity based on scene distance, we can make use of distances in the image plane. This is particularly useful if depth data is not available. It might also be useful if depth data turns out to be unreliable. Working in the image plane comes at a cost though, since the distance between the projections $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$ depends on the camera pose (Figure 4.5). Let d_{ij}^t denote the Euclidean distance between the projections $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$ of two scene points on the image plane.

$$d_{ij}^t = \left|\left|\mathbf{x}_i^t - \mathbf{x}_j^t\right|\right|_2 \tag{4.5}$$

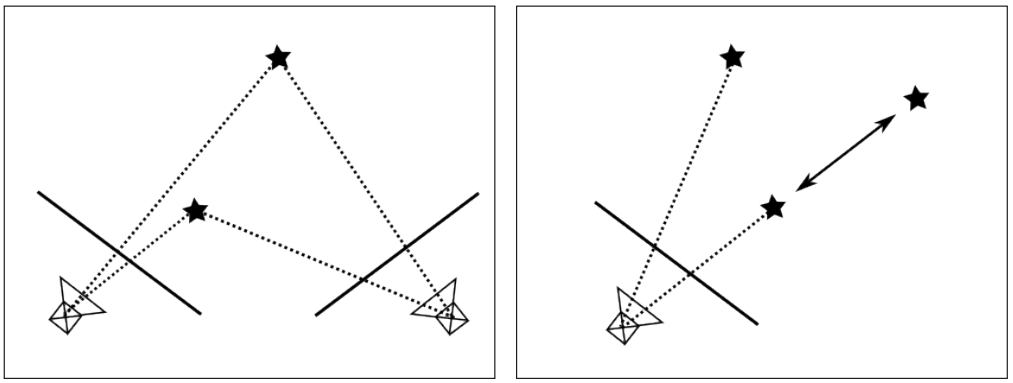
Changes in the distance of the projections do not necessarily imply that the distance between the scene points has changed. Conversely, if the distance of the projections remains constant, then the scene points have not necessarily remained static. These ambiguities cannot occur when observing the scene distance directly. This was one of the motivations behind this work.

Just as with the scene distances, we use the sample range and the sample variance as two more candidates for defining dissimilarity.

$$\hat{\mathrm{r}}\left(d_{ij}\right) = \max_t d_{ij}^t - \min_t d_{ij}^t \tag{4.6}$$

$$\hat{\mathrm{var}}\left(d_{ij}\right) = \frac{1}{|\mathcal{T}_{ij}|} \sum_{t \in \mathcal{T}_{ij}} \left(d_{ij}^t - \hat{\mu}\left(d_{ij}\right)\right)^2 \tag{4.7}$$

Brox and Malik [2] define dissimilarity in the image differently, taking global considerations such as the magnitude of the motion at a particular time into account.



(a) The projections of two points depend on angle and distance of the camera.

(b) Points on optical rays project to the same location on the image plane.

Figure 4.5: Relations between distances of scene points and their projections.

4.3.3 Computational complexity

Given N complete trajectories in T frames, a straightforward algorithm for computing any of the statistics in Equations 4.3, 4.4, 4.6, and 4.7 walks over all $\binom{N}{2}$ pairs of trajectories and computes the statistic for each pair. Computing the range or variance for a pair of trajectories is linear in the length of the trajectories, e.g. has computational complexity $\mathrm{O}(T)$. Thus, the total costs for computing dissimilarity between all trajectories are in $\mathrm{O}(TN^2)$. For example, computing the sample range in a video with $T = 50$ frames and $N = 1000$ complete trajectories leads to $50 \cdot \binom{1000}{2} \approx 25 \cdot 10^6$ computations of Euclidean distance.

4.4 Trajectory clustering

Trajectory clustering aims at finding clusters of points with coherent motion. The approach taken in this work is based on partitioning a similarity graph. Vertices in this graph represent trajectories and the weighted edges describe the pairwise similarities between trajectories. Clusters are found by cutting the graph into components. The standard objectives of clustering apply: trajectories within the same cluster should be as similar as possible, and the similarity between clusters should be as small as possible. Also, the cluster sizes should be balanced to a certain extent.

4.4.1 Similarity graph

Section 4.3 presented four ways to define the dissimilarity between two trajectories. However, spectral clustering requires pairwise similarities rather than pairwise dissimilarities. In this work, a Gaussian-shaped similarity kernel [65] is used in order to turn dissimilarities into similarities.

$$\mathrm{s} : \mathbb{R} \to \mathbb{R},\ x \mapsto \exp\left(-\frac{x^2}{\sigma^2}\right) \tag{4.8}$$

Different parameterizations of the Gaussian-shaped similarity function in Equation 4.8 are found in literature, such as the one in [75] that shows the connection to the heat kernel. Choosing to divide by σ^2 simplifies interpretation. This way, it is possible to think of x and σ having the same unit.

Any of the dissimilarities defined in Equations 4.3, 4.4, 4.6, 4.7 can be plugged into the kernel in order to obtain the similarity matrix $\mathrm{S} \in \mathbb{R}^{N \times N}$, but the scene distance range $\hat{\mathrm{r}}\left(\rho_{ij}\right)$ is of most interest in this work and used by default.

$$s_{ij} = \begin{cases} \mathrm{s}\left(\hat{\mathrm{r}}\left(\rho_{ij}\right)\right) & \text{if } |\mathcal{T}_{ij}| \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

In the case where there is no or insufficient overlap in order to compute the dissimilarity between two trajectories, the similarity is defined to be zero. Brox and Malik [2] mention that two trajectories that do not overlap are still related via transitive links in the similarity graph. This is an important property as demonstrated in the evaluation. Unfortunately, a similarity value of zero does not carry any information about whether two points were observed to move throughout the whole video sequence or whether there were no observations to indicate otherwise. In the former case, we have strong evidence for the points being on different rigid objects whereas in the latter case, we are just uncertain.

The similarity graph should be connected. Otherwise spectral clustering would result in finding the components of the graph [65]. It can happen that short trajectories in areas of missing depth end up as separate components in the similarity graph. Therefore, we keep only the largest component. The components are identified by searching the observation graph. Furthermore, the 3% of trajectories with the lowest mean similarity to all other trajectories are discarded. The purpose is to identify some outlier trajectories early.

4.4.2 Generalized eigensystem

The core step of spectral clustering consists of computing the eigenvectors. This consists of finding the first n_f eigenvectors corresponding to the smallest positive eigenvalues in the generalized eigenproblem $\mathrm{L}\mathbf{f} = \lambda \mathrm{D}\mathbf{f}$ where $\mathrm{L} = \mathrm{D} - \mathrm{S}$ (see Section 3.4). In the implementation, we use the eigensolver for generalized self-adjoint eigenproblems in the linear algebra package Eigen [76]. It is suitable because the graph Laplacian L is real symmetric (thus also self-adjoint) and the degree matrix D is positive definite. This is a solver for dense matrices and computes all eigenvectors and eigenvalues. Listing all eigenvalues and eigenvectors has computational complexity $\mathrm{O}(N^3)$. As long as the number of frames T is small enough compared to the number of trajectories N, solving the eigenproblem is the theoretical bottleneck in the presented motion segmentation algorithm.

As only the first $n_f \ll N$ eigenvectors are needed, determining the full eigenspace is not necessary. There exist efficient iterative algorithms for solving large (sparse) eigensystems of the type in Equation 3.7 such as in the software package ARPACK [77], which was chosen for the job. Using an iterative solver is a common motive in applications of spectral clustering, see e.g. [2, 66].

4.4.3 K-Means

Following the recipe outlined in Section 3.4, we take the first n_f eigenvectors corresponding to the smallest non-zero eigenvalues. By default, we choose $n_f = k - 1$ where k is the number of clusters. As to computational costs, k-means runs so fast that its runtime is negligible compared to the previous steps of computing the dissimilarity and the eigenvectors.

4.4.4 Model selection

In order to make a system interesting for real applications for end users and convenient to use by experts, its parameters should be automatically selected, taking only a RGB-D video as bare input and segment the motion without further guidance. While this is difficult to achieve, model selection is an important issue.

The parameters introduced so far in the system are the number of clusters k, the bandwidth parameter σ for the similarity, and the number of eigenvectors n_f that are used as features in k-means. The implemented system supports automatic selection of the bandwidth σ.

Similarity bandwidth σ

If one has to choose the bandwidth manually, then it makes sense to choose the bandwidth σ to reflect the scale of the dissimilarities. The first implemented heuristic chooses the bandwidth σ simply to be the mean of the dissimilarities. Besides this mean heuristic, [65] also suggests another heuristic for the bandwidth σ. Here, the edge with maximum weight ϵ is found in a minimum spanning tree of the dissimilarity graph. Then the bandwidth is set to $\sigma = \sqrt{2}\epsilon$, where the factor $\sqrt{2}$ is included for compatibility with the definitions in [65]. In the implementation, Kruskal's algorithm together with union-find [49] is used for constructing a minimum spanning tree efficiently. The costs for computing these heuristics is therefore low.

Chapter 5

Evaluation

In this chapter, we analyze the performance of the method. We begin with an explanation of the evaluation method in Section 5.1 in order to introduce the performance measures for different stages of the algorithm. In Section 5.2, we present the datasets used for analysis in Section 5.3.

5.1 Evaluation method

The purpose of this evaluation is to understand the properties of the proposed method, give a proof of concept, and show its limitations. Examples are particularly useful for this purpose. Hence, the focus is on the qualitative evaluation, which explores the cases when the method works, inspects the black box parts of the method, and shows cases in which the method fails. Even though the evaluation is mostly qualitative in nature, we quantify several aspects of the datasets and the algorithm. In the following, we describe the external measures used to quantify the error of the point tracker and the trajectory clustering stage.

5.1.1 Tracking performance

Point tracking is the first stage in the algorithm. Tracking produces the trajectories, which make up all input seen by the subsequent clustering step. It is therefore interesting to know how well the chosen point tracker actually performs. The first measure we use is the *corruption rate* t_c for measuring the amount of corrupted trajectories defined in Equation 5.1. This fraction can be determined if the ground truth segmentation is known.

$$t_c = \frac{\text{number of corrupted trajectories}}{N} \tag{5.1}$$

However, this measure has shortcomings. Firstly, a trajectory is corrupted whenever the point tracker crossed the boundary of two dynamic objects at least in one frame. This is counter-intuitive because the measure does not distinguish corrupted

trajectories from randomly generated trajectories. This sharp criterion is also troublesome if t_c is computed from unreliable ground truth. Secondly, it weights all trajectories equally regardless of their lifetimes.

In order to address these shortcomings, we derive an experimental, entropy-based measure that is meant to be more informative. The key idea is to model the tracking as a random process and measure the uncertainty in this process. Let the i-th trajectory be composed of l_i points. Let us assume that each of the points lies on a dynamic object given by the output of the i.i.d. discrete random variables $Y_{ij} \in \{1, 2, \ldots, C\}$, where C is the number of dynamic objects and $1 \leq j \leq l_i$. The entropy [78] for a random variable Y with probability distribution p is

$$\mathrm{H}[Y] = -\sum_{y} \mathrm{p}(y) \log_2 \mathrm{p}(y) \quad . \tag{5.2}$$

The i-th trajectory is represented by the joint entropy of $\mathbf{Y}_i = (Y_{i1}, Y_{i2}, \ldots, Y_{il_i})$. The joint entropy the sum of all individual entropies since the involved random variables were assumed to be statistically independent. We now define the (normalized) entropy of the trajectories t_h as

$$t_h = \frac{1}{L \log_2 C} \sum_{i=1}^{N} l_i \, \mathrm{H}[\mathbf{Y}_i] \tag{5.3}$$

where $L = \sum_{i=1}^{N} l_i$ is the sum of all trajectory lifetimes. This measure addresses both shortcomings of the simpler corruption rate. The entropy $\mathrm{H}[\mathbf{Y}_i]$ is maximum if the distribution p_i of $\mathbf{Y}_i$ assigns the same probability to all dynamic objects and zero if $\mathrm{p}_i = 1$ for a single dynamic object (see [78]). Trajectories with a longer lifetime have more influence on t_h. The division by $\log_2 C$ ensures that $0 \leq t_h \leq 1$ because the entropy H has minimum value zero and maximum value $\log_2 C$ [78].

5.1.2 Analysis of trajectory statistics

In the previous section, we defined the error measures for the trajectories. In this section, we continue to the question on how to assess the quality of the dissimilarities between the trajectories, i.e. the 3-d range, 3-d variance, 2-d range, and 2-d variance. The construction of the dissimilarity graph in Section 4.3 is driven by the idea that trajectories should be assigned high dissimilarity if the tracked points lie on different moving rigid objects. The use of the sample range and the sample variance feels intuitively right, but these statistics have not been derived in a principled manner. The question is whether their use is justified. We are trying to answer this question by comparing the dissimilarity graphs with the ideal case. In the long run, it is probably best to have a large benchmark for deciding which statistics works best. Yet, in this thesis with limited data for evaluation, analyzing the intermediate stages might be helpful to assess why the system performs well in certain scenarios and has difficulties in other.

5.1.3 Clustering performance

The quality of the segmentation can be measured by comparing the clusters estimated by the algorithm with the clusters given by ground truth. For this purpose, the *adjusted Rand index* [79] is used in this work as an external quality measure.

Ground truth

The adjusted Rand index requires ground truth. We use pixel-wise segmentations of the dynamic objects in the video frames as ground truth. These *label images* provide the true labels for the trajectories. The label images for the real-world scenes used in this evaluation (Section 5.2) have been created manually. Only every eighth frame has been annotated. The obtained ground truth data is "dense in space and sparse in time" [2]. We annotated either the depth frames or the color frames in the RGB-D videos using mostly the flood filling tool in GIMP [80]. While this approach is reasonably simple, it remains tedious and leads to inaccurate segmentations at the boundaries of the objects where neither the depth images (lateral noise) nor the color images (shadow, blur) are helpful cues for the human eye.

Adjusted Rand index

The adjusted Rand index measures agreement between two partitions of a set. It is corrected for chance, which means that it measures improvement over a randomly obtained partition (see [79] for details). The adjusted Rand index ranges between -1 and 1, where larger values indicate better agreement.

The true labels for the trajectories are obtained from the label images. It can happen that label images from two or more video frames specify different labels for the same trajectory, in which case we ignore these trajectories when computing the adjusted Rand index. Such is the case if the trajectory is corrupted, or the label images are not pixel-accurate segmentations of the dynamic objects.

5.2 Datasets

We used four datasets for evaluation. By using many datasets of different character, we hope also to explore many different properties of the method. This section describes the scenes contained in all four datasets. Three of them have been created as part of this work. These are the Office Dataset (Section 5.2.1), the Smart Mobility Lab Dataset (Section 5.2.2), and the Simulated Dynamic Scene Dataset (Section 5.2.3). The fourth dataset is the TUM RGB-D Benchmark [54] (Section 5.2.4). All the videos have a resolution of 640×480 pixels.

5.2.1 Office Dataset

The Office Dataset consists of altogether eight scenes in an office environment. Figure 5.1 shows one particular frame for each of the eight videos. Table 5.1 provides

Scene	Frames	Short description
`certificate`	123	handheld moving camera; motion mostly in a plane
`poster`	113	fixed camera; motion mostly in a plane
`prml`	67	fixed camera; motion along the optical axis
`momo`	129	fixed camera; stop and go
`globe_s`	157	handheld fixed camera; rotation
`globe_m`	123	handheld moving camera; rotation
`two_books`	59	fixed camera; two objects
`static`	236	handheld moving camera; static scene

Table 5.1: Characteristics of the scenes from the Office Dataset. The scenes are designed to contain various kinds of motion for both the camera and the objects in the scene.

an overview about the scenes. The idea behind this dataset is to explore scenes containing motion with various qualities.

`certificate` A textured planar object moves first mostly fronto-parallel to the camera (Figure 5.1a) and then away from the camera. The camera itself is moving along a circle.

`poster` A large poster is waved in a fronto-parallel plane of a fixed camera (Figure 5.1b). The background is at a depth of up to 3 m. The foreground object is large in comparison to those in other scenes.

`prml` A book is moved away along the optical axis of a fixed camera. This scene complements the `poster` scene. The `poster` scene shows translational motion whereas the `prml` scene contains motion along the optical axis.

`momo` A book is moved fronto-parallel of a fixed camera, stopping for a second in the middle of the video. This scene asks for a solution of the temporary stopping problem.

`globe_s` A globe is rotated on a lazy susan in front of a fixed camera about around 180 degrees. This is a prototypical scene for incomplete trajectories, as points on the globe appear and disappear.

`globe_m` A globe is rotated on a lazy susan in front of a moving camera. This is a version of `globe_s` with the additional challenge of a moving camera.

`two_books` Two books are moved in a fronto-parallel plane in front of a fixed camera. This scene is a test case for the ability to segment multiple objects.

`static` The camera is moved sideways in a static scene. This scene is mostly included for debugging purposes.

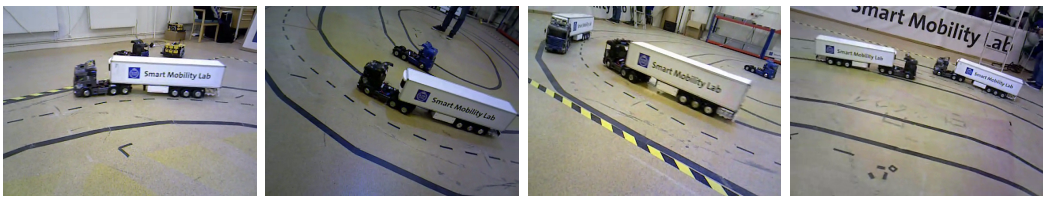
(a) `certificate` (b) `poster` (c) `prml` (d) `momo`

(e) `globe_s` (f) `globe_m` (g) `two_books` (h) `static`

Figure 5.1: The eight scenes from the Office Dataset. The scenes include planar motion, motion along the optical axis, temporary stops, rotation, moving cameras, more than one dynamic object, and no motion at all.

5.2.2 Smart Mobility Lab Dataset

The *Smart Mobility Lab Dataset* contains four videos of model trucks driving around in the Smart Mobility Lab (SML) at KTH. A frame for each video is shown in Figure 5.2, and the properties of the scenes are listed in Table 5.2. The videos exhibit some interesting properties. First, some of the dynamic scenes involve more than one truck. Second, some of the scenes contain moving semi-trailer trucks. Second, the dataset contains videos with both moving and fixed cameras.



(a) `truck_s` (b) `truck_m` (c) `two_trucks_s` (d) `two_trucks_m`

Figure 5.2: Four scenes from the Smart Mobility Lab Dataset.

The scenes in the SML dataset are challenging for our proposed method. The trucks only cover small regions in the video. The trucks are not really rigid bodies but rather articulated objects[1]. Little and repetitive texture on floor and the truck trailers make point-tracking difficult.

`truck_s` A single semi-trailer truck passes in front of a handheld fixed camera. The

[1]British English: *articulated lorry*. American English: *semi-trailer truck*.

truck appears in the scene only after thirty frames of the video and disappears around fifty frames before the video ends.

`truck_m` A single semi-trailer truck passes in front of a handheld moving camera. The truck drives along a curve, which means that the motion observed by the camera is not only translational.

`two_trucks_s` Two semi-trailer trucks pass each other in front of a handheld fixed camera. The trucks are not visible in the whole scene. In the curve, the articulated motion of the semi-trailer trucks is clearly visible.

`two_trucks_m` Two semi-trailer trucks pass each other in front of a moving camera. One of the trucks occludes the other while passing.

Scene	Frames	Short description
`truck_s`	223	handheld fixed camera; one truck
`truck_m`	246	handheld moving camera; two trucks
`two_trucks_s`	227	handheld fixed camera; two trucks
`two_trucks_m`	261	handheld moving camera; two trucks

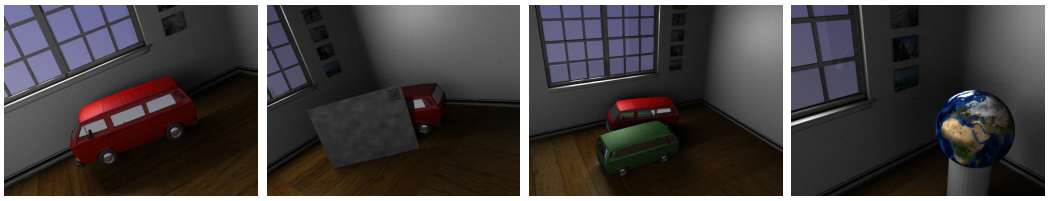
Table 5.2: Characteristics of the scenes from the Smart Mobility Lab dataset. The scenes are designed to show all combinations of a moving vs. fixed camera and one vs. two objects.

5.2.3 Simulated Dynamic Scene Dataset

Apart from real scenes, we also use simulated dynamic scenes (SDS) in the evaluation. The simulated scenes serve as a proof of concept, help understand the system, and are useful to verify the implementation. The simulations enable us to obtain datasets with pixel-accurate ground truth segmentation. Because it is not clear how well the sensor models capture the characteristics of their real counterparts, we do not make any claims about system performance on real scenes. Such claims would require rigorous model validation, which is too costly to be within the scope of this work. Table 5.3 lists the properties of the four simulated scenes. In the following, we describe the four dynamic scenes we modeled, and how they were rendered into a sequence of color and depth images.

Scene	Frames	Short description
`vwt3_m`	50	moving camera
`vwt3_o`	50	like `vwt3_m`, but with occlusion
`two_vwt3`	50	like `vwt3_m`, but with two objects
`earth`	50	fixed camera; rotating sphere

Table 5.3: Characteristics of the scenes from the Simulated Dynamic Scene Dataset.



(a) `vwt3_m` (b) `vwt3_o` (c) `two_vwt3` (d) `earth`

Figure 5.3: Four scenes from the Simulated Dynamic Scene Dataset.

Modeling of dynamic scenes

We used the Blender 3-d animation suite [81] for modeling the static background and the dynamic foreground, for rendering the dynamic scene into a sequence of color images, and for exporting the ground truth segmentation. Blender is well-suited for these tasks. Many models can be found on the web for download, which is convenient for less experienced users. Among these models is a room[2] serving as a building block for the simulated scenes shown in Figure 5.3. The virtual room has white walls and a white ceiling, a wooden floor, and a lattice window. As the walls and the ceiling of the pre-built room model do not provide any texture, we applied a few postcards sparingly to the wall right of the window. A "radio-controlled" model of a Volkswagen T3 serves as a dynamic object in the foreground. Before going into detail on how the scenes were created, we describe the four simulated scenes used in this evaluation:

`vwt3_m` This simulated scene is shown in Figure 5.3a (`vwt3_m`). It features a moving camera and a single object. The camera follows a model car moving on a curve along the walls. The window in the background would be troublesome in reality if a depth sensor is employed that actively uses infrared for measuring depth. In the simulation, we skip over that detail and take advantage of the lattice in the window, which provides some features for the point tracker. Altogether, this scene is remotely similar to the `truck_m` scene from the Smart Mobility Lab Dataset.

`vwt3_o` Figure 5.3b shows this scene. It is equal to the `vwt3_m` scene in all aspects except that for an inserted plane. The plane occludes the car completely for a short moment. The purpose of this variant is to study the effects of occlusion on the segmentation results.

`two_vwt3` This scene is depicted in Figure 5.3c. Again, it is a variant of `vwt3_m` but features two model cars instead of one. The extra car moves in the opposite direction and partially occludes the first car. This scene serves experiments with more than one dynamic object in the scene.

[2]CC BY 3.0 US. Copyright 2003-2014 Andrew Kator & Jennifer Legaz, retrieved 2013 from `http://www.katorlegaz.com/3d_models/arch_interiors/0012/index.php`. The model has been modified.

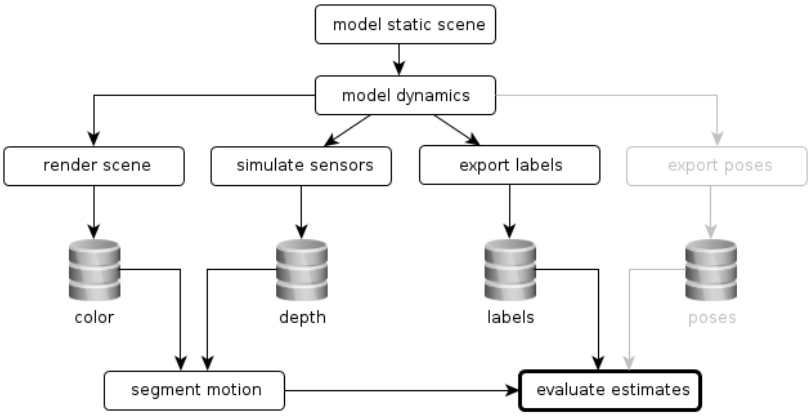


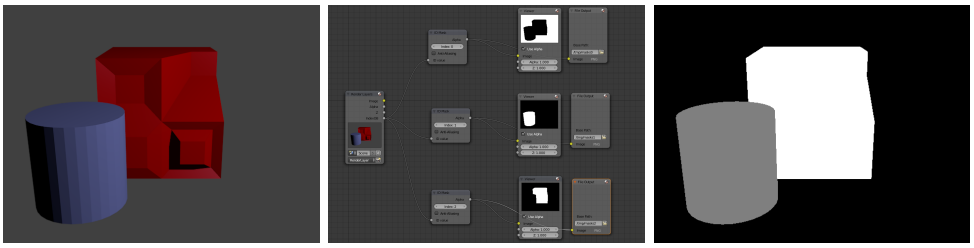
Figure 5.4: Workflow for evaluating dynamic scenes simulated with Blensor. The dynamic scene is first modeled with the Blender 3-d animation suite. The ground truth segmentation of the dynamic objects is exported to disk. Exporting the poses is supported by Blender. The dynamic scene is rendered to a sequence of color images and depth images. The motion in the rendered video is segmented and compared against ground truth.

`earth` Apart from sharing the static background, this scene is different from the other simulated scenes (Figure 5.3d). It shows a globe performing a 360 degree rotation. The scene is observed by a static camera. The idea is to have a simulated scene that is similar to the `globe_s` scene.

Creating these scenes might sound like a daunting task. However, Blender comes with a range of features that limit the costs of simulation. In order to obtain these datasets, we followed the workflow shown in Figure 5.4. In the first step, we setup the static scene, which consisted of loading the pre-built room model into Blender, equipping it with some interior lighting, and UV-mapping some pictures as postcards to the wall. These are all common activities within Blender and are well-documented.

In the second step, we inserted the dynamic objects into the scene, which comprise the detailed Volkswagen T3 model and the earth model. The Volkswagen T3 model has been kindly made available by Gabriel Adorf. The texture on the earth model stems from the Visible Earth catalog by NASA [82]. Blender allows us to animate these objects by inserting location and rotation at keyframes. Blender then interpolates motion between these keyframes. This leads to very smooth camera motion unlike the motion of the handheld cameras in the Office and Smart Mobility Lab datasets. Yet, the camera is in motion all the time in all simulated scenes except the `earth` scene. This is a challenge for the point tracker.

Having the scene built and animated, the third step consists of exporting pixel-correct ground truth segmentation. Blender allows us to assign a pass index to each



(a) Rendered image. (b) Node editor. (c) Label image.

Figure 5.5: Blender supports exporting the ground truth segmentation via the Blender node editor. The objects can be selected and marked with a pass index in Blender. The node editor can be used to generate masks for each pass index. Note that occlusion is automatically handled. In a remaining step, an extra script fuses the separate masks into a single label image.

object. These pass indices are used for masking the objects while the scene is rendered. We take advantage of Blender's node editor (Figure 5.5) to fully automatize the process. The creation of the static scene, the dynamic content, and the ground truth already covered, there is one critical question: how to obtain RGB-D videos from the modeled dynamic scenes in Blender?

Sensor models

The motion segmentation algorithm requires RGB-D videos as input. Thus, we need both a model for a color camera and a model for the depth sensor. These sensor models are described in the following.

Rendering a scene into a color video is supported by Blender out-of-the-box. By default, Blender renders each frame from the current position of the camera. This does not account for the effects introduced by a real camera in motion. Videos from a real moving camera suffer from motion blur. While on the wishlist for extra realistic scenes, we did not take the extra step to introduce motion blur in the scene and used the defaults. More care was taken with the lighting, which also has an impact on the performance of the point tracker. For example, the wall reflects light diffusively while the glass on the model car features specular reflection.

For simulation of the depth sensor, we use Blensor [83]. Blensor extends Blender by adding various depth sensor models such as the Kinect. The interface to the depth sensor models is integrated with Blender such that modeling the dynamic scene, rendering the dynamic scene to color, exporting ground truth segmentation, and the depth sensor simulation can be done all within Blender/Blensor. The depth sensor models generate point clouds in the format defined in the Point Cloud Library [84].

The Kinect sensor model was used for all simulations. This model is supposed to simulate the depth sensors of the real Kinect or the very similar Asus Xtion. According to the Blensor documentation at the time of writing, this sensor model

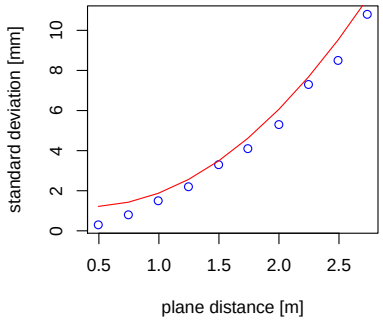


Figure 5.6: Axial noise of Blensor's Kinect sensor model (blue points) compared with the axial noise model (red curve) presented by Nguyen et al. [58]

has not undergone model validation yet. Nevertheless, we expect the sensor model to exhibit similar noise characteristics as its real counterpart. No time could be spent on model validation. But we at least verified the sensor model implementation by testing it against an empirically derived noise model of the Kinect by Nguyen et al. [58]. Their empirical noise model (Equation 5.4) gives the standard deviation σ_z of the axial noise depending on the depth z and angle θ of a plane.

$$\sigma_z\left(z,\theta\right) = 0.0012 + 0.0019(z-0.4)^2, \quad 10^\circ \leq \theta \leq 60^\circ \tag{5.4}$$

Blensor's Kinect sensor model should exhibit a similar relationship between standard deviation in the direction of the optical axis and the depth of the object. In order to verify this, we set up a Blender scene consisting of a single planar surface perpendicular to the camera. We then measured the standard deviation of all depth points from a perpendicular plane fitted to the point cloud. The standard deviation was measured at ten equidistant steps between 0.5 m and 2.75 m. Thanks to perpendicularity, the plane fitting reduces to finding the mean of the depth values of all points. By using a perpendicular plane, we generously ignored the constraint on the angle in Equation 5.4. In summary, we reproduced parts of the experiment in [58], just with a sensor model rather than a real camera. Judging from the results of the experiment in Figure 5.6, the sensor model captures the relation between depth and noise in Equation 5.4 well.

As a side note, we also experimented with depth measurements of tilted planes in recordings with the real Asus Xtion. Plane fitting with RANSAC [10] as is done by [85] turned out to be harmful. The reason is that depth maps of planar surfaces recorded by a Kinect-like sensor look like step functions. Rather than fitting a plane to all the points, RANSAC selects a level set, which likely leads to a bad fit.

Based on these experimental results, we used Blensor's default parameter configuration for all simulated scene datasets. We used Blensor at git commit `a5ac165`.

5.2.4 TUM RGB-D Benchmark

The TUM RGB-D Benchmark [54] contains several dynamic scenes. The dynamic content in these scenes is included as a challenge for camera trackers. The benchmark only provides ground truth for the motion of the camera with respect to the static background that is dominating the recorded scene. The true motion of the moving objects themselves is not known. Furthermore, the scenes contain people who are moving non-rigidly, which conflicts with one of the assumptions made in this thesis. Hence, the TUM RGB-D Benchmark is only of limited use for evaluating our method. Despite of this, we selected four videos from the benchmark that include large motion. One of them is depicted in Figure 5.7. The purpose is to gather some numbers on these videos with hundreds of frames. These four videos are listed in Table 5.4.

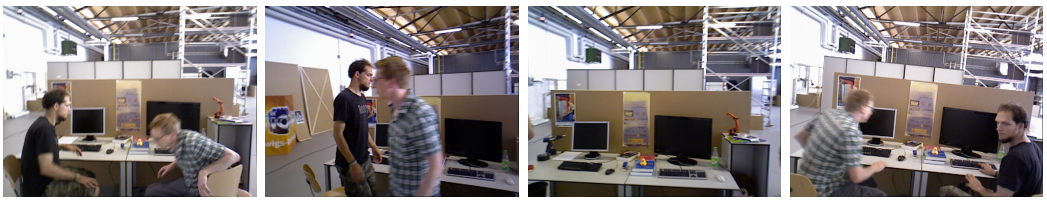


Figure 5.7: Four frames of the video (*freiburg3_walking_halfsphere*) from the TUM RGB-D Benchmark. This is not only a difficult scene for visual SLAM algorithms but also for motion segmentation. The scene features a moving camera and the people are moving non-rigidly, occlude each other, are occluded by other objects, and temporarily disappear.

Scene	Frames	Short description
`fr3_walking_static`	714	fixed camera; walking persons
`fr3_walking_halfsphere`	1018	moving camera; walking persons
`fr3_walking_xyz`	826	translating camera; walking persons
`fr3_walking_rpy`	864	rotating camera; walking persons

Table 5.4: Characteristics of the scenes from the TUM RGB-D Benchmark.

5.3 Analysis

In this section, we describe both qualitative and quantitative results for the various stages of the motion segmentation algorithm. We start by analyzing the results of the point tracker (Section 5.3.1), continue with investigating into the statistics computed on the trajectories (Section 5.3.2), before looking on the overall system (Section 5.3.3).

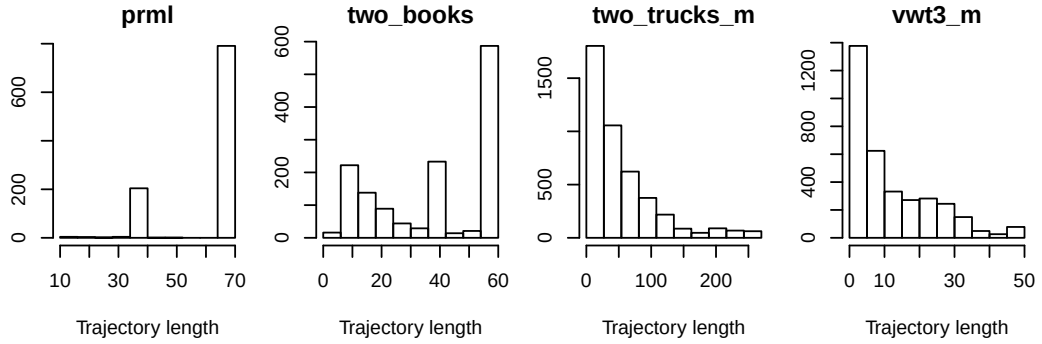


Figure 5.8: Trajectory length histograms for scenes `prml`, `two_books`, `two_trucks_m`, and `vwt3_m`. The distributions reflect scene aspects such as occlusion and camera motion.

5.3.1 Point tracking

The evaluation of the point tracker resulted in several findings. First, the lifetime of the trajectories compared to the length of the videos varies a lot between the different datasets and that the ability to handle incomplete trajectories is important for the presented method. Second, the observation graph consisted of several connected components in some occasions – an issue that can be fixed but is best avoided right from the beginning. Third, the point tracking stage does not only result in a certain quantity of corrupted trajectories but also leads to undesired effects at the boundaries of dynamic objects.

Incomplete trajectories

Missing data comes in different shapes, depending on the characteristics of the scenes. The trajectory lifetimes are summarized by histograms. The histograms for all videos are listed in Figure A.1 in the appendix. Four of those histograms are also shown in Figure 5.8. The histograms for the scenes `prml`, `two_books`, `two_trucks_m`, and `vwt3_m` have different shapes. There are a few things that can be learned from these histograms.

Even in the scenes with fixed camera, the trajectories are incomplete. For example in the `prml` scene, the camera was attached to the table and the book is moved away from the camera in front of a wall where almost no features are found. Nevertheless, the scale of the book in the images changes over time and the images are noisy. The point tracker started to track a large group of points in the middle of the video for the remaining 37 frames. This explains the two groups of trajectories in the histogram (see Figure 5.8), one spanning the whole scene and the other spanning only roughly the last half of the video. The histogram suggests that the tracker might be changed to add new keypoints continuously rather than adding them in large chunks as soon as the number of tracked points falls below the

	t_c	t_h
vwt3_m	0.020	0.018
vwt3_o	0.019	0.010
two_vwt3	0.031	0.014
earth	0.048	0.030
truck_s	0.031	0.021
truck_m	0.13	0.19
two_trucks_s	0.084	0.050
two_trucks_m	0.068	0.046

	t_c	t_h
certificate	0.034	0.027
poster	0.010	0.0085
prml	0.017	0.013
momo	0.12	0.095
globe_s	0.091	0.086
globe_m	0.048	0.052
two_books	0.19	0.12
static	0	0

Table 5.5: Corruption rate t_c and trajectory entropy t_h for all videos with ground truth segmentation.

minimum threshold.

The missing data in the trajectories are cumbersome in theory because they lead to special cases in definitions (see Equation 4.9), and in the implementation. One might therefore ask, why incomplete trajectories should not simply be discarded? To answer this question, let us consider the scenes `two_books`, `two_trucks_m`, `globe_s`, and `earth`. The `two_books` scene shows two books being waved in front of the camera, thereby occluding parts of the background. The trucks in `two_trucks_m` are not visible during the whole scene. The scenes `globe_s` and `earth` show a rotating object, which also leads to incomplete trajectories.

Corrupted trajectories

The point tracker is not always accurate. Here, we quantify the amount of corrupted trajectories using the corruption rate and trajectory entropy as defined in Equations 5.1 and 5.2.

The corruption rate and the trajectory entropy can be reliably and accurately measured on the simulated scenes because of their pixel-accurate ground truth segmentation in all frames. In summary, 1.9 - 4.8 percent of all trajectories produced by the point tracker in the simulated scenes are corrupted. Details are provided in Table 5.5.

There is a type of drift that goes undetected by the corruption rate and the trajectory entropy. The point tracker sometimes follows points that lie on an "invisible extension" close to the boundary of a dynamic object in front of untextured background. Such "hitchhikers" are shown in front of the car in Figure 5.9a, where points were mistakenly classified to belong to the car in scene `vwt3_m`.

Corrupted trajectories correlate with serious errors in motion segmentation, which happens in the `two_books` scene. Here, a book is moved from the left to the right (Figure 5.9b) in front of the camera. On its way to the right, keypoints from the background get "picked up" by the book such that the corrupted trajectories describe points on the background first, and on the foreground later. This issue caused by drift motivates recent work by Ricco and Tomasi [37]. The corrupted

(a) The point tracker mistakenly follows "invisible" points in front of the model car.

(b) Corrupted trajectories can lead to serious problems in the final segmentation.

(c) The ground truth segmentation reveals a large group of corrupted trajectories.

Figure 5.9: The point tracker does not know about the boundaries of objects (a). In another scene, wrong clustering results are obtained (b) in areas with many corrupted trajectories (c).

trajectories are shown in Figure 5.9c, where markers (x) on white background denote corrupted trajectories. It might well be that such corrupted trajectories can be automatically detected using depth data because the depth values are likely to change abruptly at object boundaries.

Runtime

The point tracker works at half real-time speed and manages between 12 and 13 frames per second on average for the given datasets. The frame rate depends on the characteristics of the video, e.g. the simulated dynamic scenes were processed only at 8-9 frames per second. The tracker is running serially.

Conclusions

The point tracker operates solely on the image plane and does not consider depth at all. In some cases, this results in many corrupted trajectories across depth discontinuities. Conversely, the 3-d sample range and the 3-d sample variance operate with depth only, not taking any discrepancies with visible boundaries in the image into account. This means that the point tracker in 2-d and the statistics in 3-d do not work optimally together. Some of these problems can successfully be identified using the corruption rate and the trajectory entropy. These measures have helped to identify the problems with corrupted trajectories at motion boundaries in the `two_books` scene. Pixel-accurate ground truth segmentations for all videos would be helpful to determine the exact amount of corruption also for all real scenes.

Despite of the mentioned issues, an advantage of the point tracker is its speed. While not running in real time, it processes several frames per second, which makes the point tracker interesting for applications.

5.3.2 Trajectory statistics

The concept in Section 4.1 described how to reason about motion in the scene, based on measuring pairwise distances between scene points. In Section 4.3, we defined four statistics on the distances measured for a pair of scene points. The question is how well these statistics work in practice. It turns out that on a coarse scale, the statistics are indeed a useful indicator for separating trajectories that describe points on different rigid objects. On a detailed scale, the analysis revealed that outliers in the dissimilarity graph can cause difficulties for spectral clustering.

Dissimilarity box plots

Are the range and the variance good at distinguishing between points on different rigid objects in practice? Box plots [86] are used in the following to answer this question. These *dissimilarity box plots* summarize the performance of the 3-d range, the 3-d variance, and their 2-d variants.

We explain the idea and interpretation of these box plots using the example in Figure 5.10a. This plot summarizes computed dissimilarities $\hat{\mathrm{r}}(\rho_{ij})$ between trajectories i and j in the `prml` scene. The plot groups trajectory pairs $\{i, j\}$ based on whether the respective scene points lie on the same rigid object or on different rigid objects. Each of the three boxes shows the (standard) lower quartile, the median, and the upper quartile of the values $\hat{\mathrm{r}}(\rho_{ij})$. The upper whisker is omitted. The quartiles and the median help assess whether our expectations hold: $\hat{\mathrm{r}}(\rho_{ij})$ assigns low dissimilarity to pair of points on the same object, and assigns high dissimilarity to pairs of points on different objects.

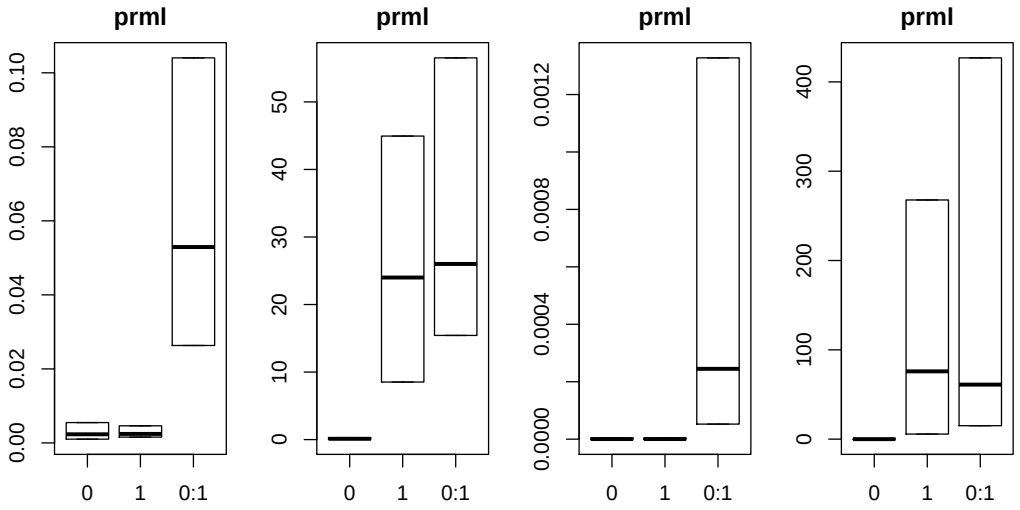
The plots work also in scenes with multiple objects, such as the scene `two_books` (Figure 5.11). The dissimilarity box plots for all annotated videos are listed in Appendix A.2.

Translational motion assumption

The dissimilarity box plots indicate that the 3-d range is superior over the 2-d range in separating the motion in the `prml` scene. The only motion in this scene is a book that moves away from the camera along the optical axis. Both the 3-d range and the 2-d range treat the background (label 0) correctly. The 3-d range, however, detects the change in depth of the book (label 1), whereas the 2-d range is confused by the scale change of the book in the image plane. The box plots in Figures 5.10b and 5.10a make this effect visible.

Missing depth

The 3-d distances can only be computed where depth is available. RGB-D cameras like the Kinect and the Xtion (Section 3.1) do not provide depth information in all regions of the video frames. Depth is missing for example for objects too close or too far from the camera. Another reason is that the infrared projector and the



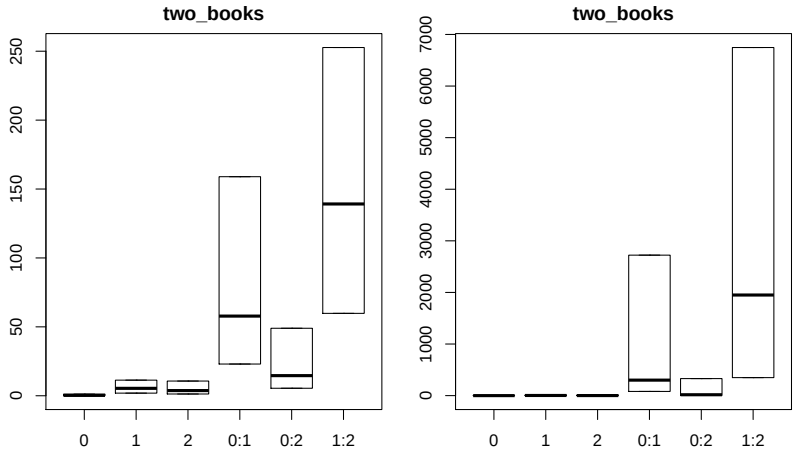
(a) 3-d range (b) 2-d range (c) 3-d variance (d) 2-d variance

Figure 5.10: The four box plots show how the values of the 3-d range, the 2-d range, the 3-d variance, and the 2-d variance are distributed in the scene `prml`. The rigid objects are labeled with 0 (here: background) and 1 (here: dynamic foreground). The boxes summarize the pairwise dissimilarities of points on the same rigid object, and the dissimilarities between points on different rigid objects (indicated by 0:1). The whiskers are omitted due to the lack of space. The concept extends to multiple rigid objects.

infrared sensor of the Kinect and the Xtion are placed next to each other, causing objects in the foreground occlude areas in the background where the infrared sensor cannot measure depth. How much depth is missing depends on the particular scene. Figure 5.12 shows the average percentage of distances that cannot be computed in 3-d due to missing depth. Between 16% and 25% of the 3-d distances cannot be computed in the scenes in the Office dataset and the SML dataset. The scenes in the TUM RGB-D Benchmark are recorded in a large hall, and parts of the scenes are out of range for the depth sensors. In practice, points with depth larger than a certain range shall not be included in the computation, which is a good idea also because the noise in the depth measurements grows quadratically with the depth for the Kinect and the Xtion [58].

Runtime

The computation of the dissimilarities between the up to $N(N-1)/2$ pairs of trajectories over all F frames runs with around 100 frames per second (wall time) on average on all datasets. However, this speed is not achieved because the algorithm is efficient (see Section 4.3.3), but rather because the computation runs in parallel.



(a) 2-d range $\hat{\mathrm{r}}(d)$ (b) 2-d variance $\hat{\mathrm{var}}(d)$

Figure 5.11: The box plots summarize the sample distribution of the 2-d range and the 2-d variance in the scene `two_books`.

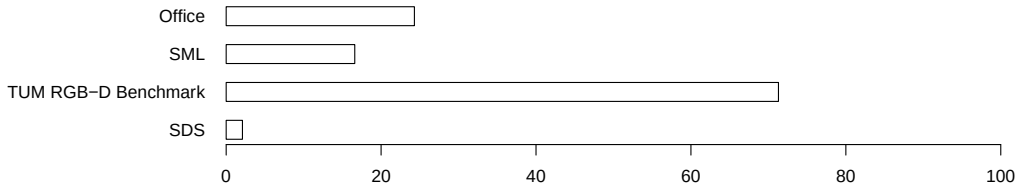
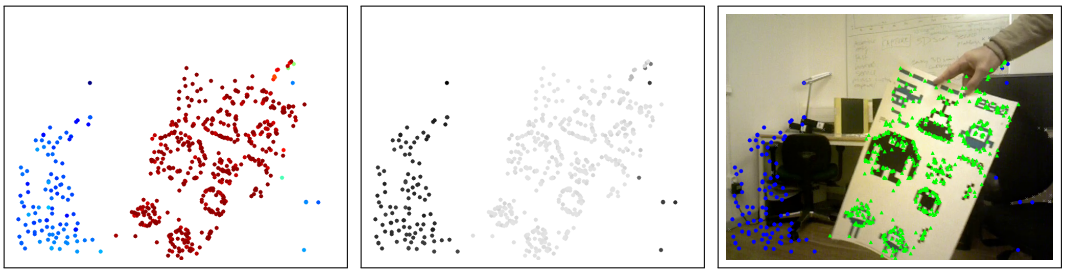


Figure 5.12: Percentage of distances not computable due to missing depth.

The number of comparisons is large. Nearly 52 million comparisons in 2-d were performed on average on the Office dataset. For the longer scenes in the SML dataset, nearly 100 million distance computations in the image plane took place. With several hundreds of frames per video, the TUM RGB-D Benchmark requires roughly 350 million comparisons on average in order to construct the dissimilarity graph. The routines to compute the dissimilarities between pairs of trajectories have to be implemented carefully. It is an advantage if the dissimilarity measures are relatively simple with regards to the implementation, such as the range and the variance.

Conclusions

The dissimilarity box plots show that the median dissimilarity within clusters is lower than the median dissimilarity between clusters for many scenes. This confirms that the conceptual idea in Section 4.1 of rigid points keeping the same distance can be put into practice. In other scenes, the box plots raise doubts on the quality



(a) using jet color map (b) grayscale (c) labels

Figure 5.13: The second eigenvector of the `poster` scene, visualized in color (a) and in grayscale (b). In this simple scene, k-means finds the labels (c) with ease, using the second eigenvector as input alone.

of the trajectories and the dissimilarity measures. The dissimilarity box plots are coarse tools, and do not capture the whole nature of the dissimilarity graphs. The next section shows that a more detailed analysis reveals issues. These issues are not obvious from the dissimilarity box plots.

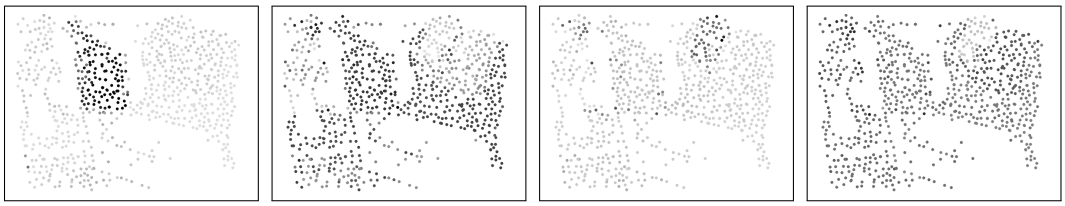
No clear answer can be given to the question whether the 3-d range and the 3-d variance are superior or inferior to their 2-d counterparts. A weakness of the 3-d variants is their inability to handle missing depth whereas the 2-d variants suffer in particular cases from the assumption of translational motion.

5.3.3 Trajectory clustering

In this section, we analyze the last stage in the pipeline, where the similarity graph is constructed and partitioned by spectral clustering. Unless noted otherwise, the 3-d range is used to construct the similarity graph. We present scenes where motion is segmented successfully. These serve as model exemplars on how results and intermediate results look like in the case of success. But the issues can be best discussed on scenes in which the algorithm has problems. These scenes are valuable as they show when and where the algorithm is not sophisticated enough to even segment scenes that it has been trained on. We finish the section with a discussion of the model selection mechanisms, the overall performance, and the runtime of the system.

Cluster structure in the eigenmap

Partitioning the similarity graph (see Section 4.4.1) is a two-step process. The first step consists of mapping the vertices to a low-dimensional embedding [75], which is referred to as the eigenmap. The expectation in spectral clustering is that these vertices are located in this low-dimensional space such that the cluster structure becomes apparent. The second step consists of actually extracting the clusters from this feature space and assign discrete labels to the trajectories.



(a) second (b) third (c) fourth (d) fifth

Figure 5.14: The eigenvectors 2–5 of the `two_books` scene all contain information about the clusters. The second eigenvector tells the first book apart where as the other three eigenvectors separate the other book.

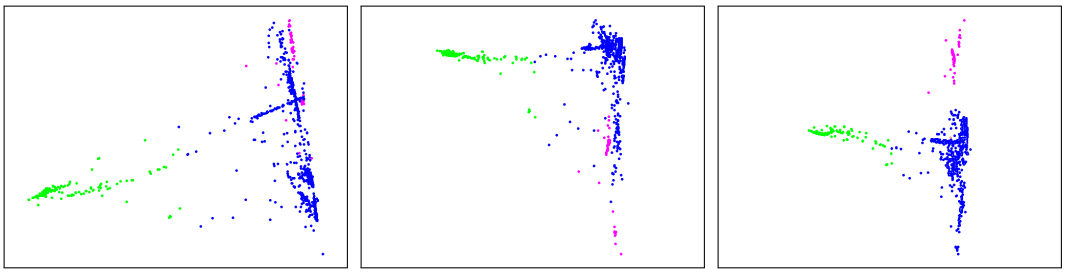
Thus, the cause of problems in the label assignment can be either due to the clusters being hard to distinguish in the eigenmap, or due to an algorithm that is not sophisticated enough to find these clusters. In the following, examples of eigenmaps are presented where the structure of the clusters is (1) well-pronounced, (2) pronounced but distorted by outliers, (3) totally distorted by outliers.

The first question is how well the second eigenvector $\mathbf{f}_2$ separates the clusters in a scene with two clusters. In the ideal case, the second eigenvector would be sufficient and the task of k-means consists of choosing the threshold. This is indeed the case for some scenes. Figure 5.13 shows the second eigenvector and the segmentation for the `poster` scene. There are no difficulties here for k-means to find a good segmentation based on the second eigenvector. The same conclusion can be drawn for scene `prml`, which is depicted in Figure 5.17b.

The second question concerns the eigenvectors corresponding to the third eigenvalue and beyond. What information about the clusters do they reveal? We follow in the footsteps of Brox and Malik [2], and plot the eigenvectors in order to gain insight into what they tell about the boundaries of the dynamic objects. The `two_books` scene serves as an example for the discussion. This scene has been recorded with a fixed camera setup and shows two moving objects. The second eigenvector is sufficient to tell the left book apart (Figure 5.14a). In the third eigenvector, the structure is less pronounced (Figure 5.14b), especially in the area of the corrupted trajectories near the book to the right. The fourth and the fifth eigenvector contain enough information to even overcome the problem with the corrupted trajectories shown in Figure 5.9c. Projections of the feature space illustrate this in Figure 5.15.

Negative impact of outlying trajectories

Unfortunately, the evaluation revealed that outlier trajectories can significantly distort the feature space and cause k-means to form clusters of outliers. Before showing scenes where this happens, we look at the scene `prml`, where the outlier problem is moderate and does not prevent k-means from finding a good segmentation. Figure 5.17a shows the 2-dimensional feature space spanned by the rows of the second and third eigenvector. Here, the labeling has been determined by k-means based on



(a) second and third (b) second and fourth (c) second and fifth

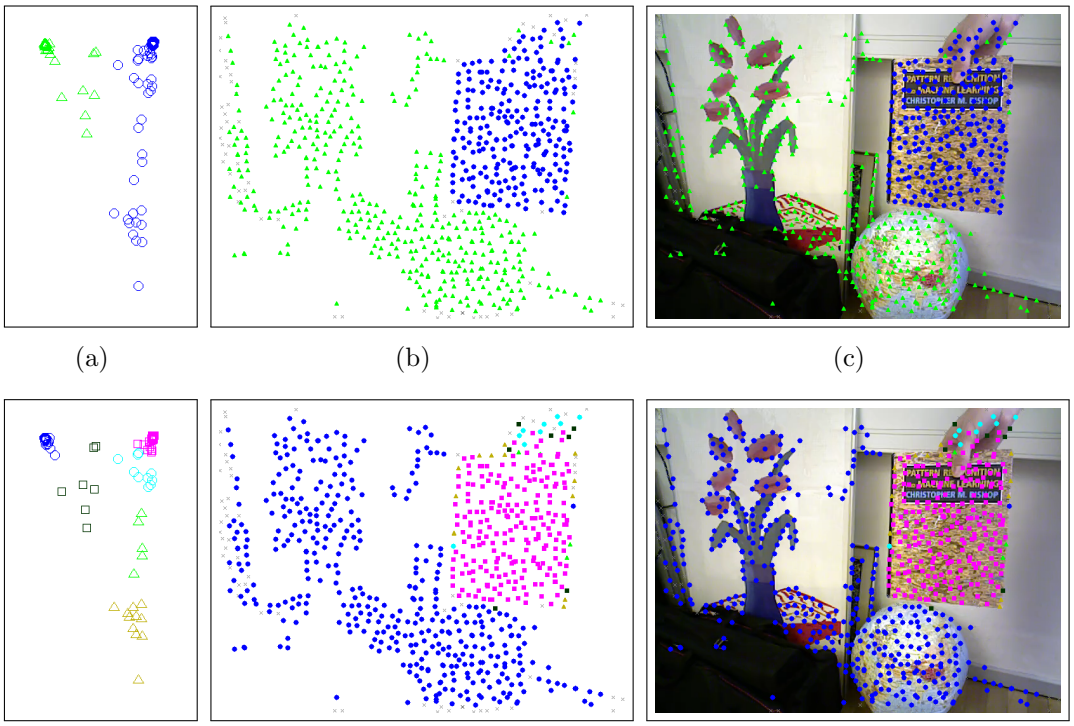
Figure 5.15: Projections of four-dimensional feature space to two dimensions. The feature space is spanned by the rows of the eigenvectors 2-5 of the `two_books` scene. k-means separates the clusters in the four-dimensional space well.

thresholding the second eigenvector only. The large majority of the 933 trajectories are mapped to similar spots in the upper corners of the feature space depicted in Figure 5.17a. Around 30 trajectories are widely spread over the feature space. It is illustrative to show which trajectories these outliers correspond to. To this end, we identified six clusters with k-means (Figure 5.17d). The result of this experiment is easy to interpret: the small outlying clusters belong to regions close to the object boundaries and to the articulated hand (Figure 5.17f). The separation of trajectories on the hand is a desired feature, while the outlying trajectories near the boundaries are undesirable. Most likely, these trajectories are corrupted and contain points both on the far-away wall and the book in the foreground.

The `prml` scene with fixed camera, roughly equal-sized clusters, and around 3% outliers in feature space, the problem with outliers showed to be less problematic than in other, more challenging scenes. For example, in the simulated `earth` scene, we observed that a trajectory got mapped so far away from all other data that k-means formed a single-element cluster. This observation is the reason why the filtering step described in Section 4.4.1 is included.

Even in the relatively easy `prml` scene, a single outlying trajectory has been observed to completely distort feature space, leading to the outlying trajectory forming a cluster by itself. In order to understand why this happens, the outlier trajectory has been analyzed in more detail. The outlier trajectory spanned the whole video and was located in a region of static background. The depth values were piece-wise constant but contained discontinuities. These discontinuities can be traced back to color sensor noise, causing the point tracker describe two points across a depth discontinuity with a single trajectory. Generally, depth sensor noise is also a problem. Both the sample range and the sample variance yield large dissimilarities between the outlier trajectory and the other trajectories. Thus, the outlier trajectory ends up with a low degree in the similarity graph with reasonable choices of bandwidth. Spectral clustering assigns the outlying trajectory a high value in the eigenvectors, as can be seen in the plot of the second eigenvector in Figure 5.16.

Figure 5.16: The second eigenvector in the `prml` scene. The single outlier trajectory is separated from the rest of the scene by k-means. Such artifacts created by noise need to be filtered away.



(a) (b) (c)

(d) (e) (f)

Figure 5.17: The second and third eigenvector are plotted on the x-axis and the y-axis, respectively (a). Thresholding the second eigenvector suffices for finding a good segmentation (b–c). However, noise and articulated motion lead to outliers in feature space (a). These outliers can be segmented with k-means by over-segmenting with a large number of clusters (d). The outlier trajectories are found on the boundary of the book and the articulated hand (e–f).

Bandwidth selection

Choosing a good bandwidth parameter value can be tricky. As stated in Section 4.4.4, it is convenient if the bandwidth can be selected automatically. In order to objectively evaluate the performance of the selection mechanisms, we would need a set of unseen sequences. Instead, we take a shortcut for at least obtaining ballpark estimates on whether the heuristic might be useful. To this end, we determine values for the bandwidth parameters manually in scenes from the Office dataset, the SML dataset, and the SDS dataset. The bandwidth values are then selected with the mean heuristic or with the minimum spanning tree heuristic based on the dissimilarity graph.

Table 5.6 lists the obtained adjusted Rand indices. The experiments have been repeated 25 times to account for the randomized parts in the algorithm. The sample standard deviation of the measured adjusted Rand index was less than $3 \cdot 10^{-4}$ for all scenes except for the `truck_m` scene, where the sample standard deviation was up to $8 \cdot 10^{-2}$. All bandwidth values can be interpreted in meters. Thus, the manually selected bandwidth values are in the range of millimeters or a few centimeters. The selected values span a relatively large range.

Overall performance

Overall, some but not all RGB-D videos could be segmented. The presented method segments the motion well in scenes `certificate`, `poster`, `prml`, `momo`, `truck_s`, `truck_m`, `two_books`, and `vwt3_m`. It gives mixed results for the scenes `globe_s` and `two_vwt3`, and `earth`. The method is not sophisticated enough to provide reasonable motion segmentations for the scenes `globe_m`, `two_trucks_s`, `two_trucks_m`. Neither could the four selected videos from the TUM RGB-D Benchmark be segmented.

Ideally, an algorithm should be powerful enough such that it can be trained perfectly on the training dataset, even though it is the generalization error on unseen instances that needs to be minimized. There remains more work to do in order to make the motion segmentation method sophisticated enough to segment all of the scenes included in the evaluation.

Nevertheless, the eight scenes where the method succeeded comprise scenes with moving cameras, scenes with more than one dynamic object, scenes with temporary stopping, scenes where objects appear and disappear, and scenes with motion in all directions. Segmentations for the scenes listed in Table 5.6 can be found in Figure A.6.

Runtime

In the final part of the evaluation, runtimes are presented for the whole motion segmentation algorithm. The runtimes are given for the scenes listed in Table 5.6, i.e. for those scenes where a reasonable motion segmentation could be obtained.

Scene	Manual	ARI	Mean	ARI	MST	ARI
certificate	0.0050	0.88	0.059	0.29	0.0091	0.91
poster	0.020	0.94	0.090	0.75	0.034	0.94
prml	0.0050	0.98	0.030	1.00	0.0095	1.00
momo	0.010	0.88	0.19	< 0	0.015	0.82
globe_s	0.10	0.71	0.18	0.66	0.032	0.46
two_books	0.050	0.80	0.092	0.72	0.030	0.35
truck_s	0.20	0.84	0.26	0.83	0.12	< 0
truck_m	0.25	0.79	0.61	0.51	0.062	0.32
vwt3_m	0.050	0.98	0.20	0.86	0.034	0.24
two_vwt3	0.10	0.67	0.20	0.67	0.036	0.62
earth	0.20	0.69	0.080	0.34	0.053	0.33

Table 5.6: Selection of the bandwidth parameter σ: manually, with the mean heuristic, and with the minimum spanning tree heuristic. The adjusted Rand index is reported for each bandwidth choice.

The experiments were performed on a up-to-date machine with several cores; the wall times are what matter most to users.

Point tracking takes most of the wall time (Figure 5.18) per frame. The purpose of the filtering (Section 4.4.1) is to identify outlier trajectories in the dissimilarity graph. Conceptually, a subgraph of the dissimilarity graph without the outliers can then be used in further processing. In the implementation however, a shortcut was taken and the dissimilarity graph is completely recomputed without the outliers. Hence, there is duplicate work in the filtering and the statistics step (Figure 5.18). The gain of removing this duplicate work is small though, and not worth the risk of premature optimization [87].

Computing the eigenmap is different. The runtimes reported in Figure 5.18 are obtained with a configuration that computes all eigenvectors. As stated in Section 4.4.2, computing all eigenvectors of a matrix has computational complexity $O(N^3)$. Here, it is worth showing the effects of computing only the first few eigenvectors with an iterative solver instead. Figure 5.19 shows the effects of replacing the full eigenproblem solver with an iterative solver for the three scenes.

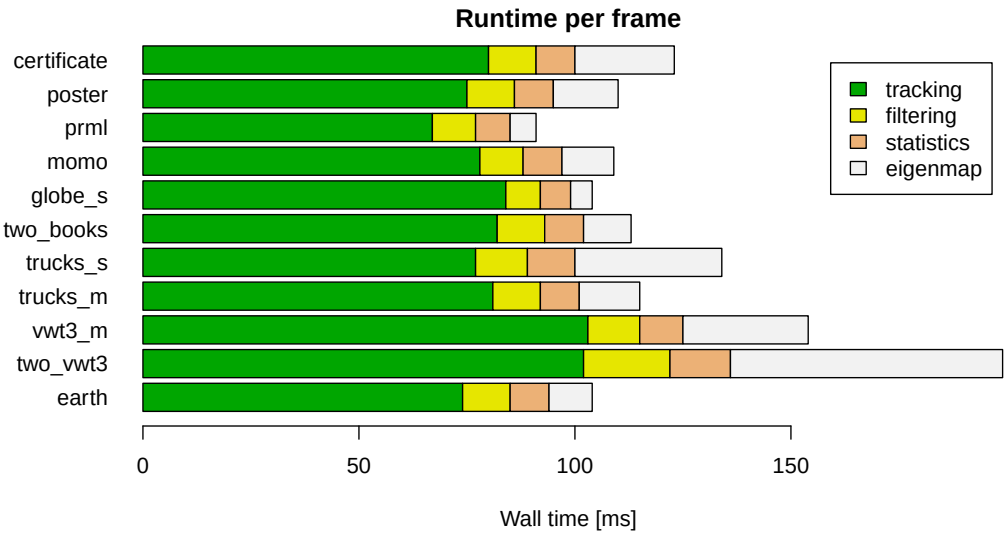


Figure 5.18: Runtime of the motion segmentation system.

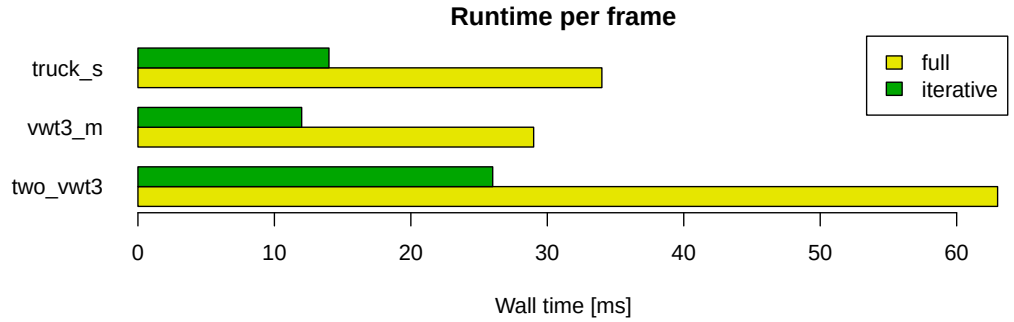


Figure 5.19: Runtime of full and iterative eigenproblem solver.

Chapter 6

Conclusion

In this chapter, we bring the key findings from this work together. Of particular interest is the thesis question: whether it is possible to segment dynamic scenes by making use of depth data from RGB-D videos. This question has two answers, and both are valid in their own sense. The first answer is *yes*, because the discussed method successfully segmented motion in dynamic scenes recorded by a moving camera. The second answer is *no*, because the method is limited in its ability to segment dynamic scenes it has been trained on. The mixed results are put into relation in Section 6.1 by showing what has been done to make it succeed in several cases, and discussing the nature of the limitations. The latter motivate Section 6.2, which lists what might be done in future to overcome the limitations. Overall, the conclusion is that despite of the difficulty of the motion segmentation problem, despite of the lack of strong assumptions such as fixed cameras or dominant background, despite of the simple approaches to each step in the presented method, the system is still able to segment various dynamic scenes.

6.1 Key findings

The good news is that the method successfully segmented dynamic scenes with unconstrained rigid motions in the presence of a moving camera, noise, and occlusion. The method accomplishes this with simple filtering, with textbook statistics on pairwise distances, and a plain vanilla implementation of spectral clustering. All of this is performed with multiple frames per second. This is promising as the various stages of the algorithm leave room for improvement.

On the other hand, it must be clearly stated that the method has not proved sufficient to segment many of the scenes included in the evaluation. This can be attributed to the method being too simple. Each of the stages has its weaknesses, and a chain is only as strong as its weakest link. The first link is the point tracker, which has the tendency to produce corrupted trajectories especially in the areas of motion boundaries. The second link consists of the trajectory statistics. The range and the variance tend to memorize noise. Furthermore, the local pairwise

comparisons in each frame do not attempt to distinguish noise from actual motion. Noise makes spectral clustering, the third link in the chain, map trajectories to locations in feature space that are either far away from the actual clusters, or in between of the actual clusters. K-Means, as the fourth and final link, has difficulties to find the clusters in the presence of noise.

6.2 Future directions

The presented method can be seen as a general framework for motion segmentation. Most of the mentioned links can be replaced with an alternative while keeping the rest of the method unchanged. Many of the ideas of Brox and Malik [2] might be transferred from 2-d to 3-d. The Lucas-Kanade point tracker can be replaced by another point tracker, possibly trading speed for quality. Less drift, longer trajectories, and semi-dense trajectories might yield better input for the subsequent stages in the method. An explicit model of sensor noise might be included when building the similarity graph. This might include other measures of statistical dispersion or temporal filtering to tackle noise. Finally, k-means might be replaced by a more sophisticated algorithm for label assignment. The findings by Brox and Malik [2] that the second eigenvector alone does not convey enough information have been confirmed in this thesis in a slightly different context. This suggests that their choice of replacement for k-means might also work in this context.

The modularity of the approach has advantages. Weak parts can be replaced. Complexity can be added where needed. However, the lack of an off-the-shelf benchmark dataset with RGB-D videos and pixel-accurate ground truth makes it difficult to decide where to start adding complexity, i.e. to determine the weakest link. In this thesis, we have taken the first steps and created several RGB-D videos, both including real-world scenes and simulated scenes. However, the creation of a large corpus of videos with depth and ground truth together with a workbench for evaluation is a task complex enough for a separate master thesis. A benchmark with annotated RGB-D videos might drive research in motion segmentation using depth data in a similar way as the Hopkins-155 database has done for 2-d trajectory-based motion segmentation.

Bibliography

[1] L. Zappella, X. Llado, and J. Salvi, "New Trends in Motion Segmentation," in *Pattern Recognition* (P.-Y. Yin, ed.), ch. 3, InTech, 2009.

[2] T. Brox and J. Malik, "Object Segmentation by Long Term Analysis of Point Trajectories," in *Computer Vision – ECCV 2010*, Lecture Notes in Computer Science, pp. 282–295, 2010.

[3] R. Megret and D. DeMenthon, "A Survey of Spatio-Temporal Grouping Techniques," tech. rep., INSA Lyon; University of Maryland, 2002.

[4] L. Zappella, "Motion Segmentation From Feature Trajectories," 2008.

[5] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, "Real-Time Foreground–Background Segmentation using Codebook Model," *Real-Time Imaging*, vol. 11, pp. 172–185, June 2005.

[6] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pp. 559–568, ACM, 2011.

[7] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time Dense Surface Mapping and Tracking," in *Proceedings of the 2011 IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, IEEE, Oct. 2011.

[8] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion," in *Proceedings of the 2013 International Conference on 3D Vision*, IEEE, June 2013.

[9] Y. Sheikh, O. Javed, and T. Kanade, "Background Subtraction for Freely Moving Cameras," in *Proceedings of the 2009 IEEE International Conference on Computer Vision*, pp. 1219–1225, IEEE, Sept. 2009.

[10] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[11] X. Zhou, C. Yang, and W. Yu, "Moving Object Detection by Detecting Contiguous Outliers in the Low-Rank Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 3, pp. 597–610, 2013.

[12] S. Perera and N. Barnes, "Maximal Cliques Based Rigid Body Motion Segmentation with a RGB-D Camera," in *Computer Vision – ACCV 2012* (K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, eds.), Lecture Notes in Computer Science, pp. 120–133, Springer, 2013.

[13] J. Yan and M. Pollefeys, "A General Framework for Motion Segmentation: Independent, Articulated, Rigid, Non-rigid, Degenerate and Non-degenerate," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), Lecture Notes in Computer Science, pp. 94–106, 2006.

[14] A. Goh and R. Vidal, "Segmenting Motions of Different Types by Unsupervised Manifold Clustering," in *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–6, IEEE, June 2007.

[15] S. R. Rao, R. Tron, and R. Vidal, "Motion Segmentation via Robust Subspace Separation in the Presence of Outlying, Incomplete, or Corrupted Trajectories," in *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, June 2008.

[16] M. Fradet, P. Robert, and P. Pérez, "Clustering Point Trajectories with Various Life-Spans," in *2009 Conference for Visual Media Production*, pp. 7–14, IEEE, Nov. 2009.

[17] N. Sundaram, T. Brox, and K. Keutzer, "Dense Point Trajectories by GPU-Accelerated Large Displacement Optical Flow," in *Computer Vision – ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), Lecture Notes in Computer Science, pp. 438–451, Springer, 2010.

[18] T. Brox and J. Malik, "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 500–513, Mar. 2011.

[19] P. Ochs and T. Brox, "Object Segmentation in Video: A Hierarchical Variational Approach for Turning Point Trajectories into Dense Regions," in *Proceedings of the 2011 IEEE International Conference on Computer Vision*, pp. 1583–1590, IEEE, Nov. 2011.

[20] D. Cremers and S. Soatto, "Motion Competition: A Variational Approach to Piecewise Parametric Motion Segmentation," *International Journal of Computer Vision*, vol. 62, pp. 249–265, Nov. 2005.

[21] J. Adorf, "Object Detection and Segmentation in Cluttered Scenes through Perception and Manipulation." Bachelor thesis, Technische Universität München, 2011.

[22] E. Herbst, P. Henry, X. Ren, and D. Fox, "Toward Object Discovery and Modeling via 3-D Scene Comparison," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, pp. 2623 – 2629, IEEE, 2011.

[23] E. Herbst, X. Ren, and D. Fox, "RGB-D Flow: Dense 3-D Motion Estimation Using Color and Depth," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*, pp. 2276 – 2282, IEEE, 2013.

[24] J. Kenney, T. Buckley, and O. Brock, "Interactive Segmentation for Manipulation in Unstructured Environments," in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, pp. 1377–1382, IEEE, May 2009.

[25] E. Herbst, X. Ren, and D. Fox, "Object Segmentation from Motion with Dense Feature Matching," in *Workshop on Semantic Perception, Mapping and Exploration at the 2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012.

[26] L. Chang, J. R. Smith, and D. Fox, "Interactive Singulation of Objects from a Pile," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, pp. 3875–3882, IEEE, 2012.

[27] M. Unger, M. Werlberger, T. Pock, and H. Bischof, "Joint Motion Estimation and Segmentation of Complex Scenes with Label Costs and Occlusion Modeling," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, no. 2, pp. 1878–1885, IEEE, June 2012.

[28] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke, and D. Cremers, "Stereoscopic Scene Flow Computation for 3D Motion Understanding," *International Journal of Computer Vision*, vol. 95, pp. 29–51, Nov. 2011.

[29] C. Vogel, K. Schindler, and S. Roth, "3D Scene Flow Estimation with a Rigid Motion Prior," in *Proceedings of the 2011 IEEE International Conference on Computer Vision*, pp. 1291–1298, IEEE, 2011.

[30] P. Bhat, K. Zheng, N. Snavely, A. Agarwala, M. Agrawala, M. Cohen, and B. Curless, "Piecewise Image Registration in the Presence of Multiple Large Motions," in *Proceedings of the 2006 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2491–2497, IEEE, 2006.

[31] B. D. Lucas, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of Imaging Understanding Workshop*, vol. 130, pp. 121–130, 1981.

[32] J.-Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker," Tech. Rep. 2, Intel Corporation, 2000.

[33] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[34] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," in *Computer Vision – ECCV 2006*, vol. 3951 of *Lecture Notes in Computer Science*, pp. 404–417, Springer, 2006.

[35] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF : Binary Robust Independent Elementary Features," in *Computer Vision – ECCV 2010*, vol. 6314 of *Lecture Notes in Computer Science*, pp. 778–792, Springer, 2010.

[36] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

[37] S. Ricco and C. Tomasi, "Dense Lagrangian Motion Estimation with Occlusions," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1800–1807, IEEE, June 2012.

[38] C. Tomasi and T. Kanade, "Shape and Motion from Image Streams under Orthography: a Factorization Method," *International Journal of Computer Vision*, vol. 9, pp. 137–154, Nov. 1992.

[39] R. Tron and R. Vidal, "A Benchmark for the Comparison of 3-D Motion Segmentation Algorithms," in *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, June 2007.

[40] L. Zappella, *Manifold Clustering for Motion Segmentation.* PhD thesis, University of Girona, 2011.

[41] C. Julia, A. D. Sappa, F. Lumbreras, J. Serrat, and A. Lopez, "Rank Estimation in 3D Multibody Motion Segmentation," *Electronic Letters*, vol. 44, no. 4, pp. 4–5, 2008.

[42] T. Schoenemann and D. Cremers, "A Coding-Cost Framework for Super-Resolution Motion Layer Decomposition," *IEEE Transactions on Image Processing*, vol. 21, pp. 1097–110, Mar. 2012.

[43] P. Ochs and T. Brox, "Higher Order Motion Models and Spectral Clustering," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 614–621, IEEE, June 2012.

[44] P. Ochs, J. Malik, and T. Brox, "Segmentation of Moving Objects by Long Term Video Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Dec. 2013.

[45] C. Zach, D. Gallup, J.-m. Frahm, and C. Hill, "Fast Gain-Adaptive KLT Tracking on the GPU," in *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, no. 1, IEEE, 2008.

[46] "OpenCV (Open Source Computer Vision)." http://www.opencv.org/, 2014.

[47] K. Fragkiadaki, G. Zhang, and J. Shi, "Video Segmentation by Tracing Discontinuities in a Trajectory Embedding," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1846–1853, IEEE, June 2012.

[48] K. R. Gabriel and R. R. Sokal, "A New Statistical Approach to Geographic Variation Analysis," *Systematic Biology*, vol. 18, no. 3, pp. 259—-278, 1969.

[49] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 3rd ed., 2009.

[50] U. Feige, "A Threshold of ln n for Approximating Set Cover," *Journal of the ACM*, vol. 45, pp. 634–652, July 1998.

[51] Y. M. Shin, M. Cho, and K. M. Lee, "Multi-Object Reconstruction from Dynamic Scenes: An Object-Centered Approach," *Computer Vision and Image Understanding*, vol. 117, pp. 1575–1588, Nov. 2013.

[52] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, D. Cremers, R. Siegwart, and W. Burgard, "Towards a benchmark for RGB-D SLAM evaluation," in *Proceedings of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systens Conference (RSS)*, pp. 1–2, 2011.

[53] J. Sturm, W. Burgard, and D. Cremers, "Evaluating Egomotion and Structure-from-Motion Approaches Using the TUM RGB-D Benchmark," in *Workshop on Color-Depth Camera Fusion in Robotics at the 2012 IEEE/RJS International Conference on Intelligent Robot Systems*, IEEE, 2012.

[54] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, IEEE, Oct. 2012.

[55] R. Vidal, R. Tron, and R. Hartley, "Multiframe Motion Segmentation with Missing Data Using PowerFactorization and GPCA," *International Journal of Computer Vision*, vol. 79, pp. 85–105, Nov. 2008.

[56] "Technical Specifications: Kinect for Windows." http://msdn.microsoft.com/en-us/library/jj131033.aspx, 2014.

[57] "Technical Specifications: Asus Xtion Pro Live." http://www.asus.com/Multimedia/Xtion_PRO_LIVE/#specifications, 2014.

[58] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking," in *Proceedings of the 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pp. 524–530, IEEE, Oct. 2012.

[59] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[60] B. K. Horn and B. G. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, pp. 185–203, Aug. 1981.

[61] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, 2006.

[62] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd ed., 2004.

[63] V. I. Voloshin, *Introduction to Graph Theory*. New York: Nova Science Publishers, 2009.

[64] R. Diestel, *Graph Theory*. Heidelberg: Springer, 4th ed., 2010.

[65] U. von Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[66] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[67] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, pp. 849–856, MIT Press, 2001.

[68] F. R. K. Chung, *Spectral Graph Theory*, vol. 92 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, Feb. 1997.

[69] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A Survey of Kernel and Spectral Methods for Clustering," *Pattern Recognition*, vol. 41, pp. 176–190, Jan. 2008.

[70] C. Alzate and J. A. K. Suykens, "Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 335–47, 2010.

[71] J. Shi, J. Malik, and C. S. Division, "Normalized Cuts and Image Segmentation," in *Proceedings of the 1997 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 731–737, 1997.

[72] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*. Addison-Wesley, 1st ed., 1989.

[73] J. Adorf, "Seminar Report on Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA [C . Alzate and J . Suykens , 2010]." Seminar report, Technische Universität München, 2012.

[74] M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions*. Wiley, 3rd ed., 2000.

[75] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[76] G. Guennebaud and J. Benoit, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[77] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. 1998.

[78] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[79] L. Hubert and P. Arabie, "Comparing Partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[80] "GIMP – The GNU Image Manipulation Program." http://www.gimp.org, 2014.

[81] "Blender." http://www.blender.org, 2014.

[82] "Visible Earth. A catalog of NASA images and animations of our home planet." http://visibleearth.nasa.gov/, 2014.

[83] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree, "BlenSor: Blender Sensor Simulation Toolbox," in *Advances in Visual Computing*, Lecture Notes in Computer Science, pp. 199–208, Springer, 2011.

[84] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, pp. 1–4, IEEE, May 2011.

[85] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications.," *Sensors (Basel, Switzerland)*, vol. 12, pp. 1437–54, Jan. 2012.

[86] L. Råde and B. Westergren, *Mathematics Handbook for Science and Engineering*. Springer, 2004.

[87] D. E. Knuth, "Structured Programming with go to Statements," *ACM Computing Surveys*, vol. 6, pp. 261–301, Dec. 1974.

Notation

F	Number of frames.
N	Number of trajectories.
C, k	Number of clusters.
u, v	Image coordinates.
$\mathbf{P}_i$	Point in the scene.
$\mathbf{x}_i$	2-d trajectory.
$\mathbf{X}_i$	3-d trajectory.
$\mathbf{x}_i^t$	2-d image coordinates of the projection of $\mathbf{P}_i$ at time t.
$\mathbf{X}_i^t$	3-d camera coordinates of point $\mathbf{P}_i$ at time t.
l_i	Length of trajectory.
ρ_{ij}^t	Euclidean distance between points $\mathbf{P}_i$ and $\mathbf{P}_j$ at time t.
d_{ij}^t	Euclidean distance between projections $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$.
$\mathcal{T}_{ij}$	Temporal overlap of two trajectories.

$\hat{\mathrm{r}}$	Sample range.
$\hat{\mu}$	Sample mean.
$\hat{\mathrm{var}}$	Sample variance.

G	Graph.
$\mathcal{A}, \mathcal{B}, \mathcal{V}$	Sets of vertices.
$\mathcal{E}$	Set of edges.
d_i	Degree of a vertex.
vol	Volume of a graph.
cut	Cut in a graph.
Ncut	Normalized cut in a graph.
D	Degree matrix.
S	Similarity matrix.
L	Graph Laplacian.
F	Matrix of eigenvectors.
λ_i	i-th smallest eigenvalue.
$\mathbf{f}_i$	Eigenvector for eigenvalue λ_i.

σ	Parameter for Gaussian similarity function.
n_f	Number of eigenvectors to compute.

p	Probability mass function.
H	Entropy.
t_c	Trajectory corruption rate.
t_h	Trajectory entropy.

Abbreviations and Acronyms

ARI	Adjusted Rand index.
GPU	Graphical processing unit.
KTH	Kungliga Tekniska högskolan.
LSA	Local subspace affinity.
MST	Minimum spanning tree.
NASA	National Aeronautics and Space Administration.
PCA	Principal component analysis.
RANSAC	Random sample consensus.
RGB-D	Red, green, blue - depth.
SDS	Simulated dynamic scenes.
SLAM	Simultaneous localization and mapping.
SML	Smart Mobility Lab.
TUM	Technische Universität München.

Appendix A

Plots

A.1 Trajectory lengths

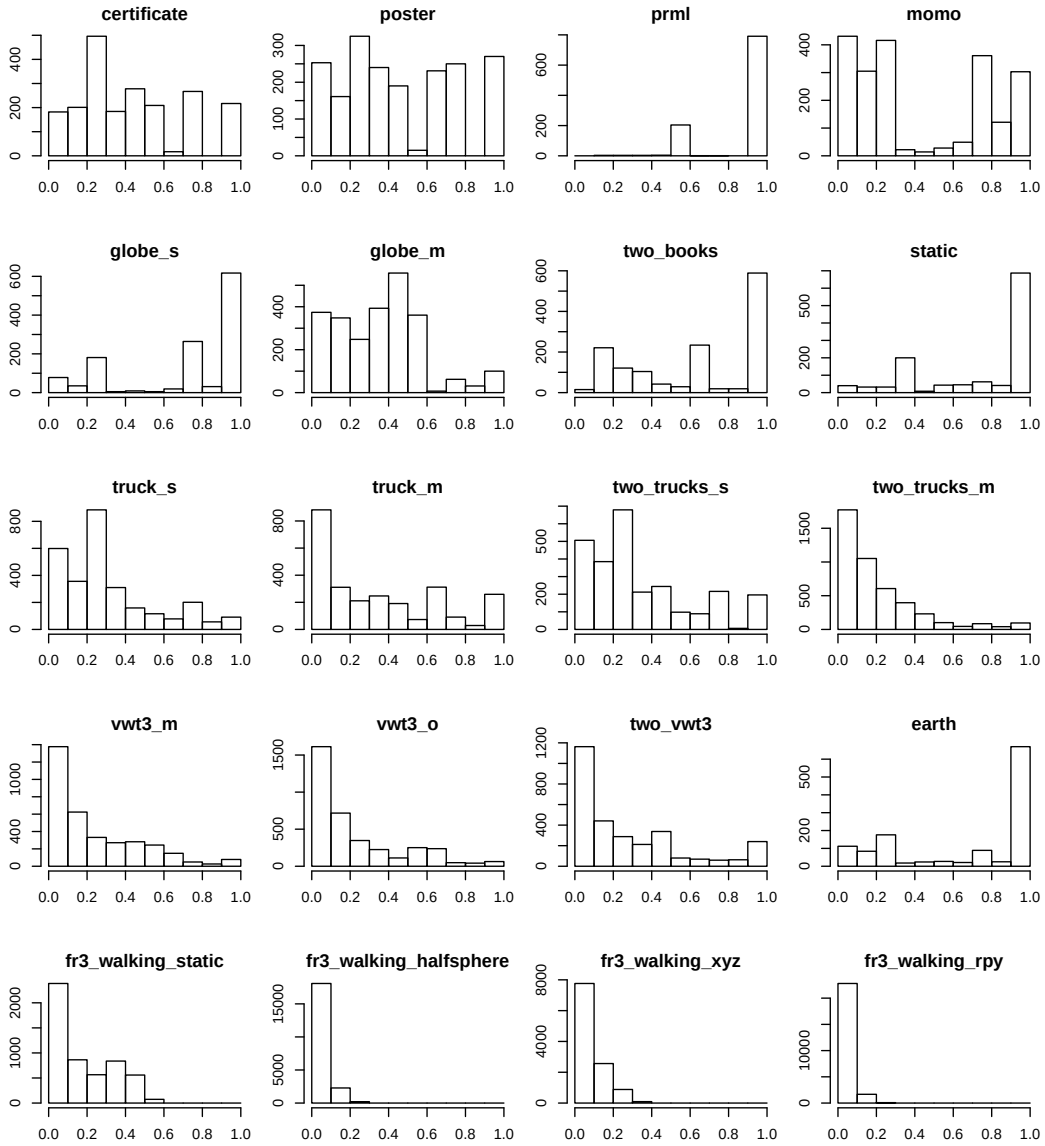


Figure A.1: Trajectory length histograms for all datasets. The x-axis denotes the fraction of the video in which the trajectory was alive. The number of trajectories in each bin is given by the y-axis.

A.2 Dissimilarity box plots

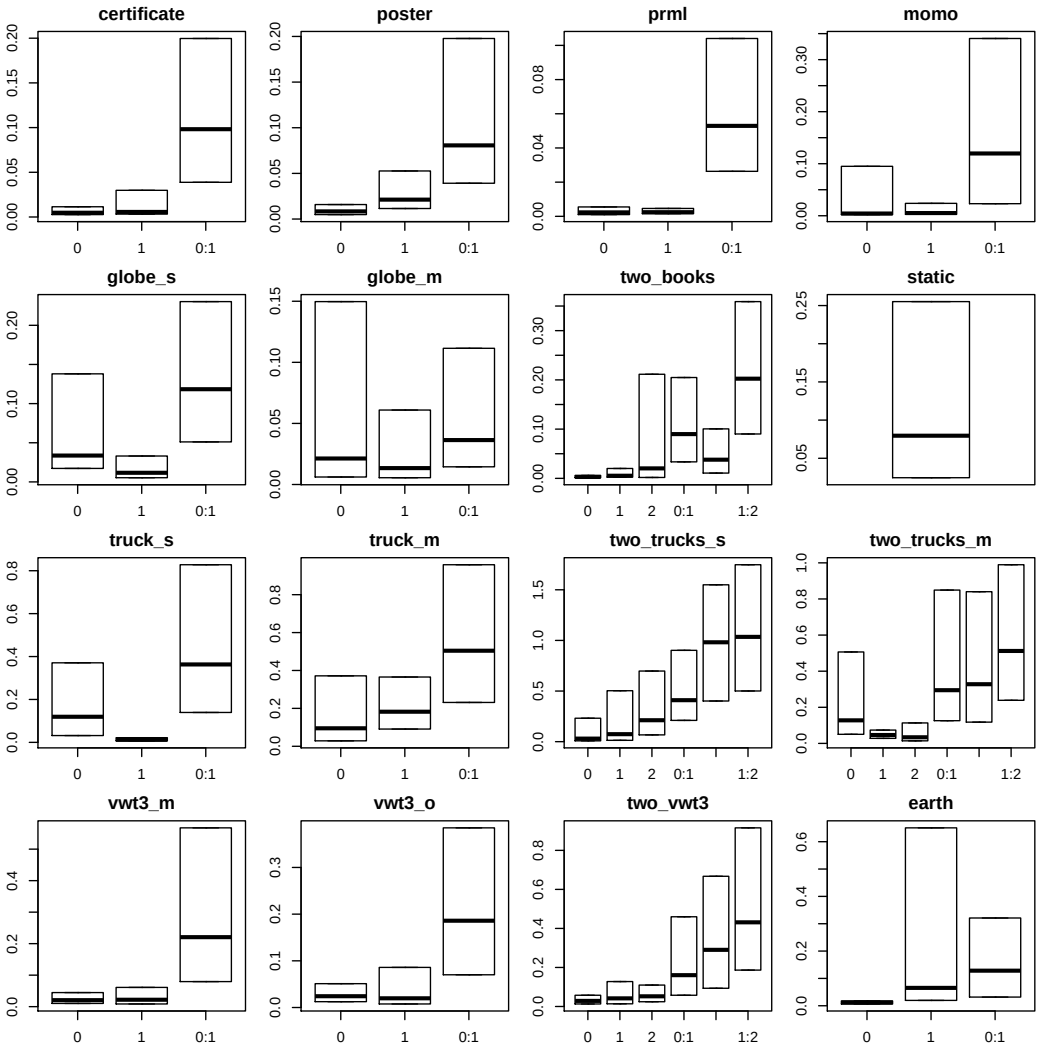


Figure A.2: 3-d range box plots for all annotated videos.

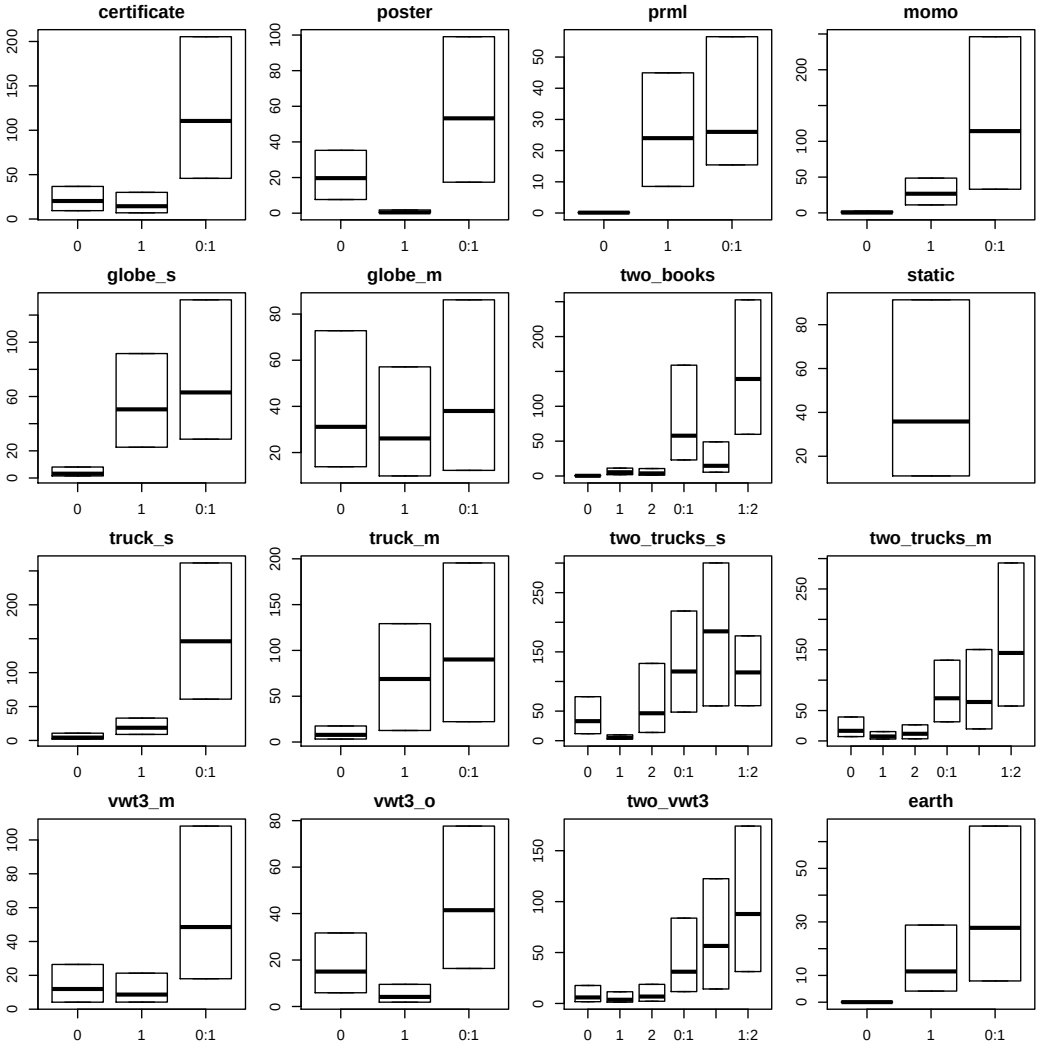


Figure A.3: 2-d range box plots for all annotated videos.

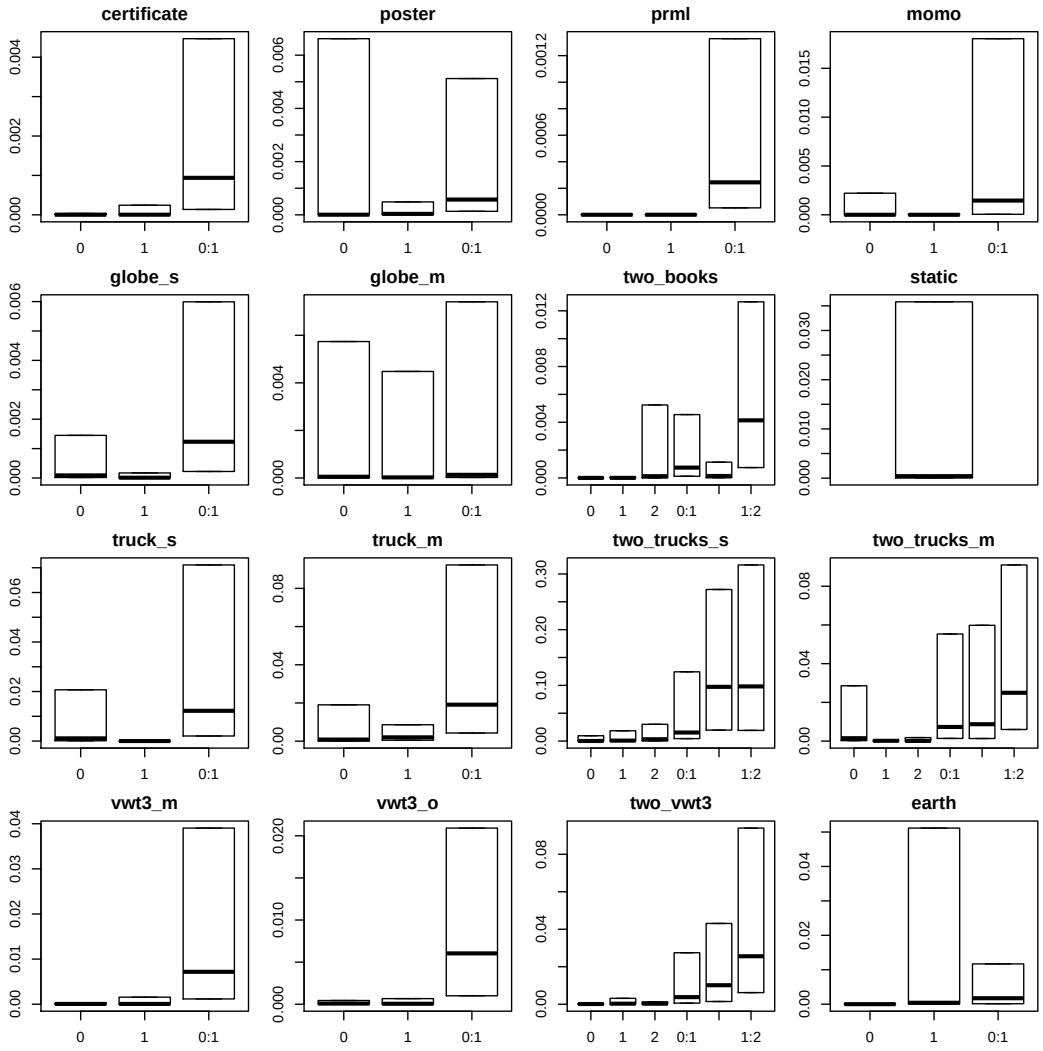


Figure A.4: 3-d variance box plots for all annotated videos.

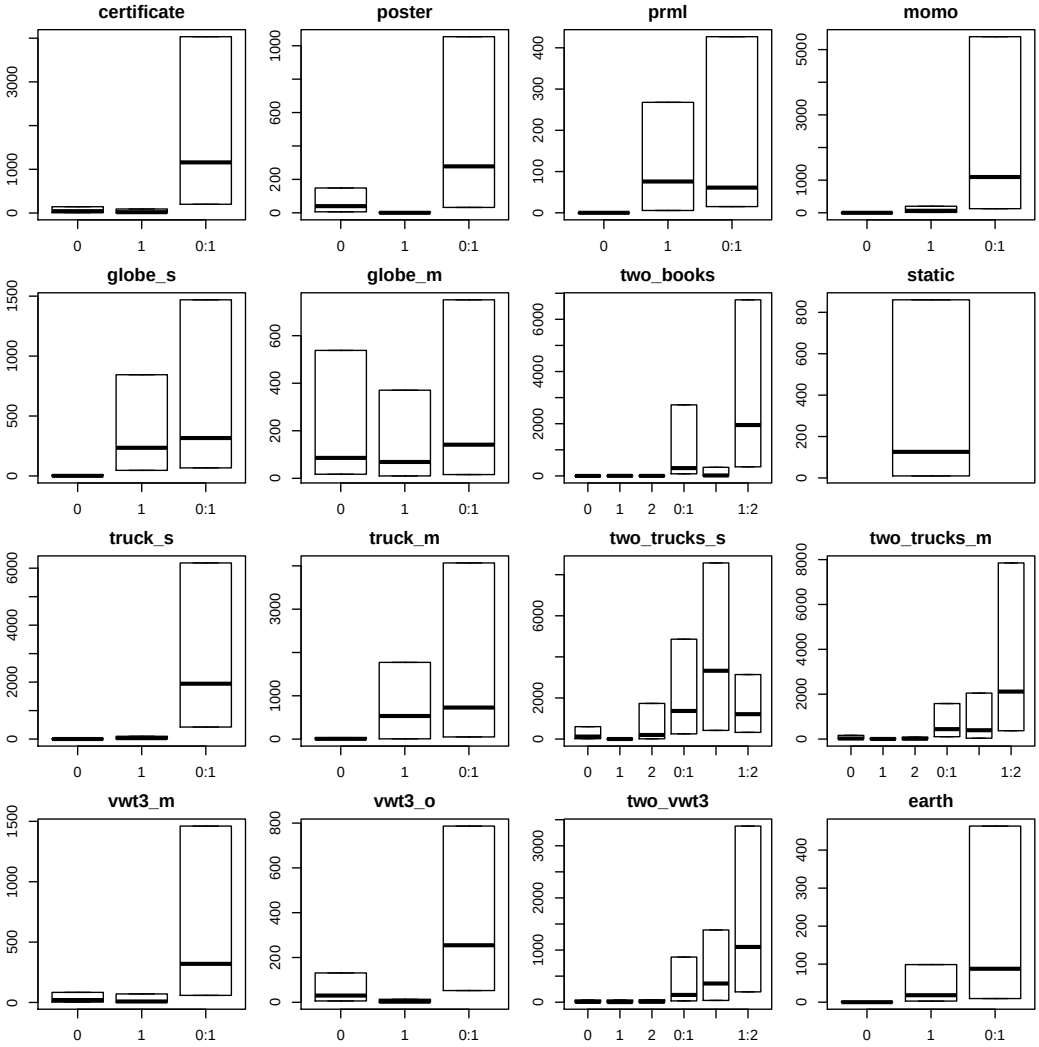
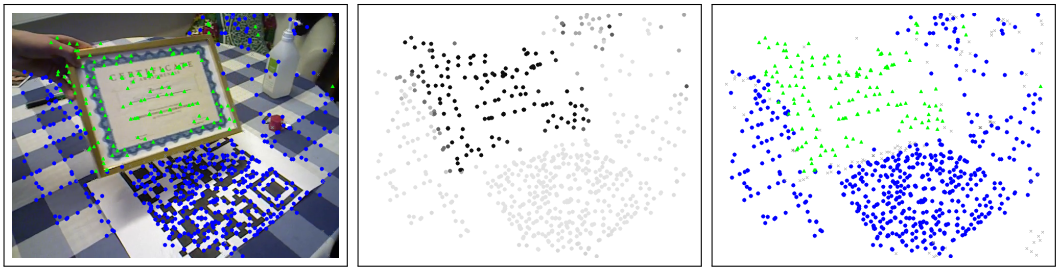


Figure A.5: 2-d variance box plots for all annotated videos.

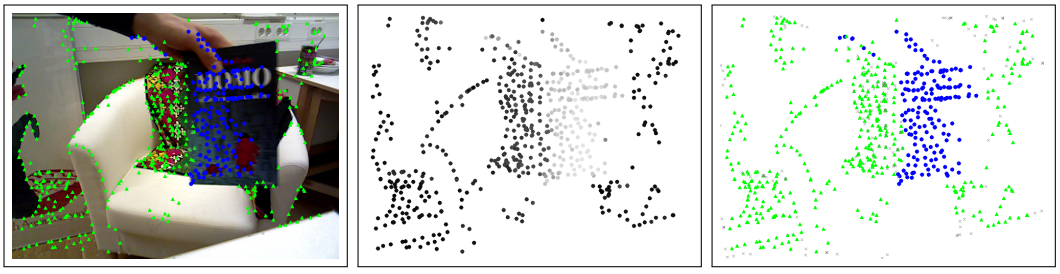
A.3 Segmentations



(a) `certificate`

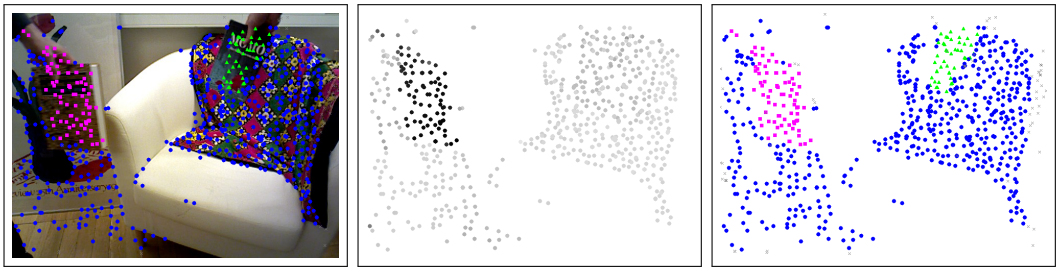
(b) `poster`

(c) `prml`

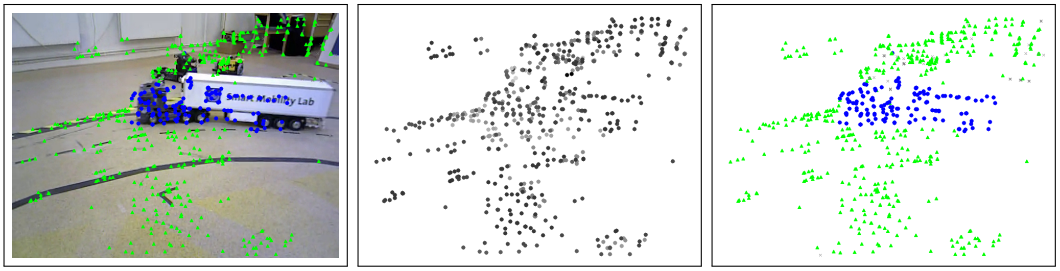


(d) `momo`

(e) `globe_s`

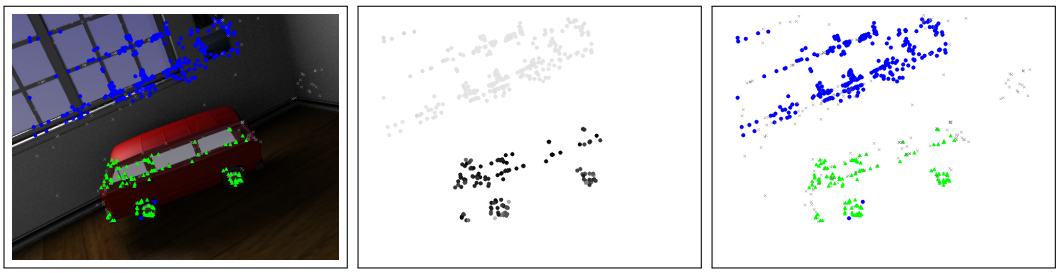


(f) `two_books`



(g) `truck_s`

(h) `truck_m`



(i) `vwt3_m`

(j) `two_vwt3`

(k) `earth`

Figure A.6: Motion segmentation results for selected scenes. The left-most images show a video frame with the labeled trajectories. The center images show the second eigenvector in grayscale. The right-most images show the labels alone.