TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Bildverarbeitung und Mustererkennung

Efficiency by Sparsity: Depth-Adaptive Superpixels and Event-based SLAM

David Weikersdorfer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 23. September 2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 30. Mai 2014 angenommen.

Abstract

While the automation of many industries is already a reality, most tedious tasks in every-day life take place in complex and uncontrollable environments which require sophisticated models and intelligent reasoning. Typical hard problems are the detection of objects or the navigation in an unknown environment. In this thesis sparse models are used to develop efficient solutions to both of these problems. Depth-Adaptive Superpixels are a novel image segmentation technique for combined colour and depth sensors which creates excellent oversegmentations and full segmentations for static images or video streams. Event-based SLAM is a novel event-based simultaneous localization and mapping algorithm which uses a biologically inspired dynamic vision sensor to dramatically decrease the computational complexity and thus enabling applications in small robotic systems. Both methods do not only have a very high quality but can also be executed with much less computation power than conventional algorithms.

Zusammenfassung

Während die industrielle Automatisierung bereits weit fortgeschritten ist entziehen sich viele müselige Aufgaben im täglichen Leben noch der Automatisierung, da sie in komplexen und unkontrollierbaren Umgebungen statt finden und so raffinierte Modelle und intelligente Algorithmen erfordern. Typische Beispiele sind die Erkennung von Objekten oder die Navigation in einer unbekannten Umgebung. In dieser Doktorarbeit werden sparse Modelle verwendet um effiziente Lösungen zu beiden Problemen zu entwicklen. Tiefenadaptive Superpixel sind eine neuartige Methode für die Segmentierung von kombinierten Farb- und Tiefenbildern, die exzellente Übersegmentierungen und Objektsegmentierungen von statischen Bildern und Bildsequenzen berechnen. Ereignisbasierter SLAM ist eine neuartige Methode für simultane Lokalisierung und Kartenerstellung die einen biologisch inspirierten Bildsensor verwendet um die Berechnungskomplexität dramatisch zu reduzieren und so eine Anwendung in kleinen robotischen Systemen ermöglicht. Beide Methoden erzeugen nicht nur qualitativ hochwertige Ergebnisse sondern kommen gleichzeitig auch mit deutlich weniger Rechenkraft als konventionelle Algorithmen aus.

Contents

1 Introduction

Automation plays a crucial role in current research and development projects: humanoids shall aid in the care for elderly people, autonomous robots are developed to scout disaster areas and autonomously driving cars have the potential to increase safety and make driving more enjoyable. For all these tasks the ability to analyse the operation environment for people, obstacles, or objects and places of interest is a fundamental requirement. Additionally autonomous machines must be able to navigate towards theses points of interest without getting stuck or lost, and without hurting humans or hitting obstacles. This very basic set of skills has been mastered by humans and even the smallest animals, but still poses a lot of questions and open problems for machines.

The scientific field of computer vision tackles theses problems and has made substantial progress towards solutions in the last decades. Image segmentation has been an active field of study with extraordinary results and many approaches have been developed to detect objects in images taken by colour cameras. While these methods often give very good results, studying pure colour images alone is limiting, as it is often difficult to extract geometric information from projections of 3D objects. The introduction of combined colour and depth (RGB-D) sensors has been a major step towards the analysis of complex three dimensional scenarios.

Towards autonomous navigation and exploration, numerous methods have been proposed for simultaneous localization and mapping (SLAM) or for analysing and understanding complex three-dimensional environments. While it is a difficult computational problem to navigate in 3D without any depth information, the problem can be simplified by using a distance sensors like a laser range finder. Robust mathematical models have been developed to fuse information into an environment map and to allow localization within the created map.

In practical applications not only the potential performance of a method in solving the task at hand is relevant, but also its resource efficiency. Autonomous robots are limited with respect to size, weight and electrical power consumption and thus computational resources, like CPU and RAM, have to be used efficiently. This problems becomes increasingly important for flying exploration vehicles or embedded applications.

An approach which has the potential to provide not only functional but also efficient solutions are sparse data models. Sparse models have been used to great success for example in machine learning [61, 90, 14] or for dimensionality reduction in general [92, 49]. In contrast to dense models which use and represent all available information, sparse models focus on the salient or relevant information. This principle allows to concentrate attention to important aspects instead of trying to understand every piece of information. Sometimes a sparse approach can also yield additional insight by identifying and extracting hidden patterns in complex data.

The leading principle of this thesis is ”*Efficiency through sparsity*”. Efficiency can be viewed from two sides: an algorithm gets more efficient if it runs faster while giving the same quality of results, or if it gives better results within the same runtime. ”Sparsity” can be realized in very different forms and here it will be mainly used to reduce the dimensionality of the visual sensor input with only minimal loss of information. This work demonstrates that the intelligent choice of sparse models actually results in algorithms which are both faster, and at the same time give a higher quality.

Efficiency through sparsity will be demonstrated within two main topics: superpixels and event-based vision. The first part concentrates on the segmentation of RGB-D images and video streams by using a sparse image representation called superpixels. Here similar pixels are grouped into small segments, called ”superpixel”, which results in a much more compact image representation which can greatly simplify the analysis of images and video streams with subsequent high-level algorithms. In the second part a different approach to sparse models is chosen by using an event-based dynamic vision sensor which provides a sparse representation directly in hardware. Such sensors do not provide a series of dense images, but a continuous stream of only pixel locations, where pixels are only reported when a change in illumination in this particular pixel has been registered. Novel computer vision algorithms have to be developed for this kind of sensor which work directly on this sparse data stream and benefit from the sparse representation of dynamic changes.

1.1 Depth-Adaptive Superpixels

Classically an image is represented by a rectangular array where each entry, called ”pixel”, represents one measurement point. Images are captures by CMOS chips which measure the intensity of red, blue and green light for each pixel and periodically map a complete measurement of all pixels into computer memory for further processing. Today the trend goes towards increasingly bigger image resolutions with high-definition resolutions of 1080×1920 pixels being already at the lower end of the resolution spectrum.

But often a higher image resolution is not required for successfully understanding the contents displayed in the image. On the contrary, a high number of pixels overloads many sophisticated methods which, as a result, often use down-sampled images with a very low resolution of e.g. only 120 to 160 pixels. But the blind down-sampling of an image is often not a good choice as important

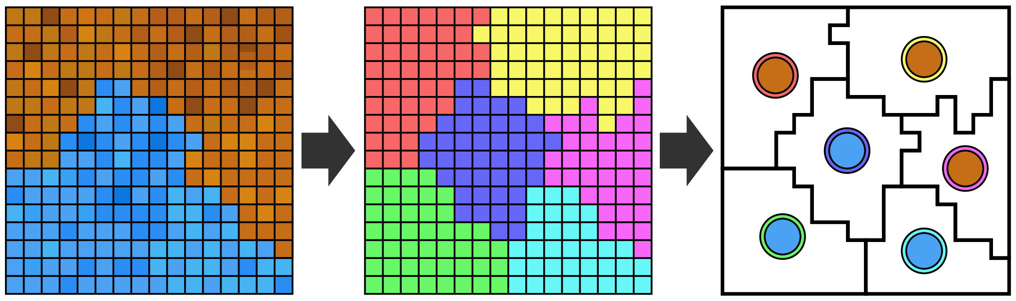


Figure 1.1: **From left to right:** Dense pixel grid, oversegmentation (using false colours) and boundary-preserving superpixels symbolized with circles at segment barycentre with segment mean colour.

information like rugged segment edges may be lost, while other redundant regions of an image are not compressed enough. A very successful method to intelligently compress the information of an image are image oversegmentation techniques [5, 48, 63, 64, 68, 72, 81, 88] which have entered the focus of study in the recent years.

An image oversegmentation clusters pixels into segments, so called ”superpixel”, where each superpixel is a good representation of all the pixels it contains (see fig. 1.1). Oversegmentations do not try to immediatelly fully understand an image, instead they form a sparse, intermediate layer of information. This sparse representation preserves segment boundaries and most of the relevant image information, and can be used by high-level algorithms for example to complete the image segmentation process or for various other applications like object tracking or saliency detection.

In the recent years combined colour and depth sensors have been a great success as they simplify many applications in computer vision. However up to now oversegmentation algorithms have focused on pure colour images. In this work a novel oversegmentation technique, *Depth-Adaptive Superpixels*, is presented which extends the principle of superpixels to combined color and depth images. The method forms the basis of two high-level algorithms, *Depth-Adaptive Superpixel Segmentation* and *Temporal Depth-Adaptive Superpixels*, which compute full RGB-D image and RGB-D video segmentations.

1.2 Event-Based SLAM

Many results in computer vision focus on frame-based video cameras which are essentially just very fast photo cameras producing still images every few milliseconds. Typical framerates for normal consumer products are 24 to 60 frames per seconds at high-definition resolutions. However many motions in every day like a car moving on the road, a waving hand or a falling object require higher framerates for reliable object tracking. While professional high speed cameras can provide

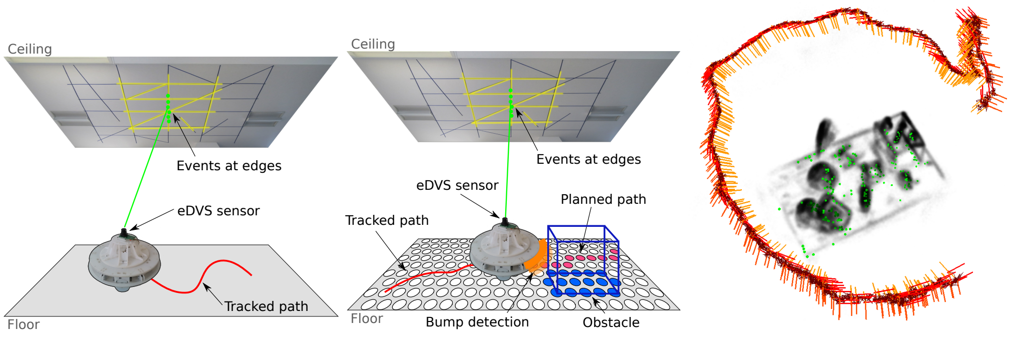


Figure 1.2: **Left:** Event-based Simultaneous Localization and Mapping; **Middle:** Autonomous Exploration with event-based SLAM; **Right:** Event-based 3D SLAM.

several thousand frames per seconds, high frame rates imply expensive hardware, low resolution and an increased noise level. Additionally, all frames need to be processed by the computer – for example the PrimeSense device has a bandwidth of 26 MB/s for the colour image alone (see table 6.1 for details). A high bandwidth requires high computation power, high electrical power and is infeasible for many embedded realtime applications.

Dynamic vision sensors chose a different approach to encode the dynamic changes in a scene and only reports data for pixels which actually change their brightness values. Such an approach provides a completely new view on tracking and self-localization. Instead of working with full dense images, only a sparse stream of pixel locations with changed data needs to be processed. However this new field of event-based computer vision requires a revisiting of established principles and the development of new ideas to adapt to the nature of dynamic vision sensors. Dynamic vision sensors have already been used successfully in several applications and are actively investigated in current research.

Of particular interest in many computer vision applications are object tracking, self-localization and mapping, but up to know no general tracking or mapping algorithm for dynamic vision sensor has been proposed. In this work a novel particle filter algorithm, *Event-based Particle Filter*, is presented which the framework of Condensation to dynamic vision sensors. The algorithm is highly efficient and thus suitable for realtime applications and embedded applications. *Event-based Particle Filter* is extended to a simultaneous localization and mapping algorithm – the *Event-Based SLAM* algorithm – by using a novel approach to generate environment maps. Both algorithms demonstrate how the efficient usage of a sparse representation of dynamic changes can result in highly efficient tracking algorithms, compared to state-of-the-art which already often requires GPGPU hardware to barely run in realtime.

1.3 Publications

Parts of the material discussed in this thesis was published and presented at peer-reviewed computer vision and robotics conferences:

- David Weikersdorfer, David Gossow and Michael Beetz: *Depth-Adaptive Superpixels.* Proceedings of the International Conference on Pattern Recognition (ICPR), Tokyo, Japan, 2012
- David Weikersdorfer and Jörg Conradt: *Event-Based Particle Filtering for Robot Self-Localization.* Proceedings of the International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, China, 2012
- David Weikersdorfer, Alexander Schick and Daniel Cremers: *Depth-Adaptive Supervoxel for Efficient RGB-D Video Analysis.* Proceedings of the International Conference on Image Processing (ICIP), Melbourne, Australia, 2013
- David Weikersdorfer, Raoul Hoffmann and Jörg Conradt: *Simultaneous Localization and Mapping for event-based Vision Systems.* Proceedings of the International Conference on Computer Vision Systems (ICVS), St. Petersburg, Russia, 2013
- Raoul Hoffmann, David Weikersdorfer and Jörg Conradt: *Autonomous Indoor Exploration with an Event-Based Visual SLAM System.* Proceedings of the European Conference on Mobile Robots (ECMR), Barcelona, Spain, 2013
- David Weikersdorfer, David Adrian, Daniel Cremers and Jörg Conradt: *Event-based 3D SLAM with a depth-augmented dynamic vision sensor.* **SUBMITTED** at International Conference on Robotics and Automation (ICRA), Hong Kong, 2014

1.4 Outline and Contributions

This thesis is structured as follows (see fig. 1.3): This introduction, Depth-Adaptive Superpixels (part I), Event-based SLAM (part II), and Conclusions and appendix. Part I and II are self-contained and can be studied in isoloation.

At the beginning of the first part in §2 the *Adaptive Superpixels* (*ASP*) oversegmentation algorithm is introduced which extends the state-of-the-art algorithm SLIC to non-constant density functions. In order to run in realtime a fast Poisson disc sampling algorithm is developed (§2.3) and a density-dependent compactness term is introduced for a local iterative cluster algorithm (§2.4). *ASP* forms the basis of the *Depth-Adaptive Superpixels* (*DASP*) oversegmentation algorithm for RGB-D images (§3). *DASP* transports the concept of uniform and compact superpixels from 2D to 3D and can also be viewed as an oversegmentation algorithm for single-view point clouds. For the full segmentation of RGB-D images a new graph based method is developed which results in the *Depth-Adaptive Superpixel Segmentation* (*s-DASP*) algorithm (§4). It is demonstrated how a local similarity measure on a graph of superpixels can be globalized with the help of spectral graph theory to yield good image segments. *DASP* is additionally extended to the RGB-D video stream segmentation method *Temporal Depth-Adaptive Superpixels* (*t-DASP*) which uses strands of superpixels and a graph structure on theses strands to provide very good results for RGB-D video stream analysis (§5).

In the second part, the *Event-based Particle Filter* (*EB-PF*) algorithm for dynamic vision sensors is introduced (§6). This algorithm is a novel approach to object tracking and self-localization using only a sparse stream of pixel events. *EB-PF* is extended to a novel simultaneous localization and mapping algorithm – the *Event-Based SLAM* (*EB-SLAM*) algorithm – by using an intelligent and surprisingly simple method to represent and generate environment maps (§7). The algorithm is applied in an autonomous exploration scenario where a small robot in the style of a vacuum cleaner creates a map of an unknown environment (§7.4). In §8 the *EB-SLAM* algorithm is adapted and reformulated for the three-dimensional case leading to an highly efficient 3D SLAM algorithm (*EB-SLAM-3D*).

All proposed methods are further introduced and motivated in their respective chapters. At the end of each chapter a thorough evaluation of the proposed methods is presented. All evaluations include a comparison to ground truth data and if applicable a comparision to comparable state-of-the-art methods.

The thesis is concluded in §9 with a revision of the presented methods and remarks on possible future work. In the appendix a listing of quality metrics for superpixels (§A) and supplementary evaluation results (§B) can be found.

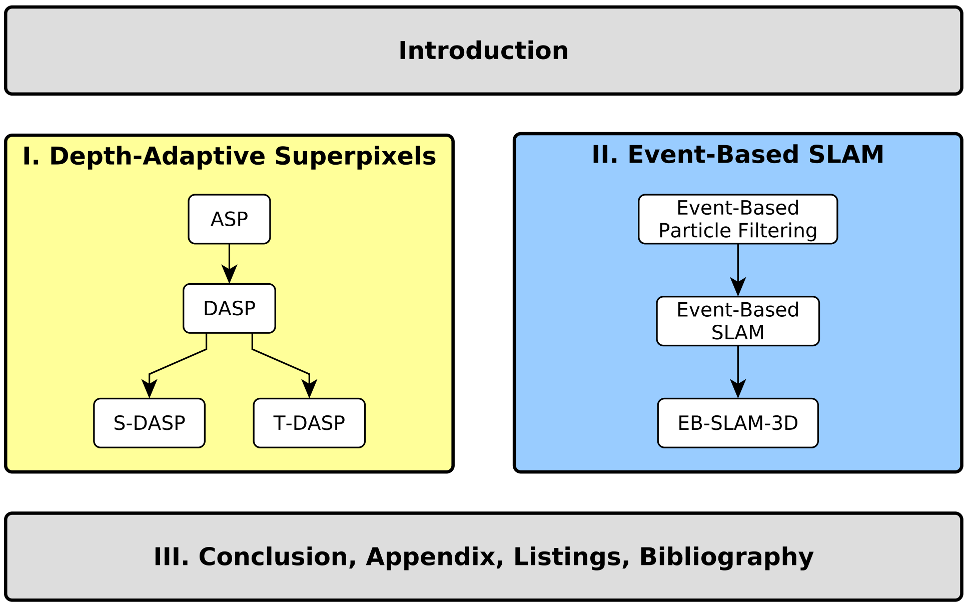

Introduction
I. Depth-Adaptive Superpixels
ASP
DASP
S-DASP
T-DASP
II. Event-Based SLAM
Event-Based Particle Filtering
Event-Based SLAM
EB-SLAM-3D
III. Conclusion, Appendix, Listings, Bibliography

Figure 1.3: Outline of this thesis

Part I

Depth-Adaptive Superpixels

2 Adaptive Superpixels

Adaptive Superpixels is an oversegmentation technique which distributes superpixels according to an arbitrary, user defined density function. This method forms the theoretical basis of *Depth-Adaptive Superpixels*.

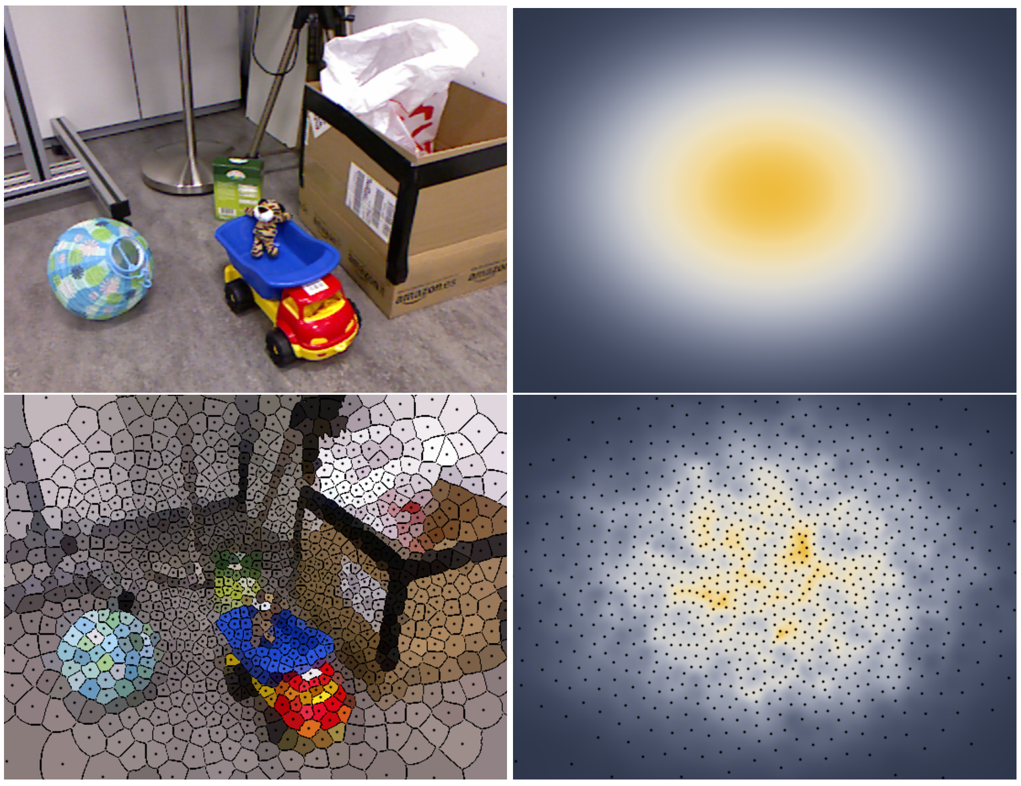


Figure 2.1: **Top**: Input colour image and user density function. **Bottom left**: Adaptive Superpixels and superpixel centres. **Bottom right**: Density actually realized by superpixels.

2.1 Superpixels

2.1.1 Image segmentation

Detecting objects in a camera image or video stream is one of the most important requirements for many practical applications of computer vision. The problem is mostly formulated as a labelling task where each pixel in the image is assigned label indicating that it belongs to a certain semantic group like tree, sky, house, cow, ... The semantic is highly dependent on the context, scope and application and in many scenarios a vast amount of implicit and explicit knowledge is required to define it sufficiently well to be practical. Example applications are identification of tumours in medical images, detecting humans on the street or differentiating objects on a supermarket shelf.

The goal of image segmentation is the division of an image into a set of non-overlapping segments with certain properties. Following [64] segments shall fulfil two criteria: **intra-segment similarity** and **inter-segment dissimilarity** (see fig. 2.2).. Intra-segment similarity describes the property that all pixels from one segment belong to the same semantic group, while inter-segment dissimilarity indicates that pixels from any two segments belong to different semantic groups

Mathematically, an image segmentation is a partition which is defined as follows:

Definition 1. Let Ω be a set and $\mathfrak{P} = \{S_1, \dots, S_n \mid S_i \subset \Omega\}$ a set of subsets of Ω. $\mathfrak{P}$ is called a **partition** of Ω iff.
(i) $S_1 \cup \dots \cup S_n = \Omega$ – subsets cover the whole set, and
(ii) $\forall\, i \neq j : S_i \cap S_j = \emptyset$ – subsets are pair-wise disjunct.
Elements $S \in \mathfrak{P}$ are called **segments**.

Due to the variability of visual appearances, the high ambiguity of colour information and the complexity of natural occurring scenes, inter-segment dissimilarity is often an ill-formed problem when no additional models or other form of prior knowledge is available. On the opposite, intra-region similarity is

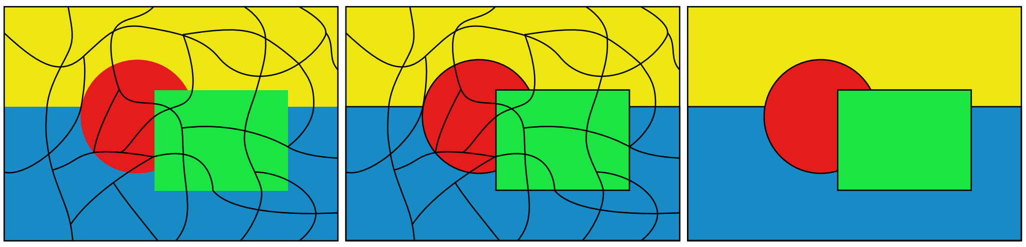


Figure 2.2: **Left:** A partition without special properties. **Middle:** A partition which satisfies intra-segment similarity but not inter-segment dissimilarity. **Right:** A partition which satisfies both intra-segment similarity and inter-segment dissimilarity.

often much easier to establish when no additional information is available. This is partly due to the fact, that intra-region similarity is a non-exclusive, local property which can be established by only considering local neighbourhoods of pixels. The allocation of specific pixels to a segment does influence allocation choices which happen in another far away region of the image. Inter-region dissimilarity, on the other hand, is an exclusive, global property which compares every segment to every other segment to assure that no two segments belong to the same semantic group.

Intra-region similarity is much easier to achieve and a special kind of image segmentation which only satisfies intra-region similarity is called oversegmentation. Oversegmentations are an elegant sparse image representation which can be used to formulate more efficient computer vision algorithms. In this chapter a new oversegmentation called *Adaptive Superpixels* is presented – it will be introduced in §2.2 after a discussions of benefits and applications of oversegmentations and the presentation of state-of-the art oversegmentation methods in the rest of this section. The chapter will then continue with the methodology of *ASP* in §2.3 and §2.4. It is concluded with a short evaluation and the presentation of examples applications in §2.5.

2.1.2 Image Oversegmentation

A segmentation of an image which satisfies intra-region similarity but not necessarily inter-region similarity is called an **oversegmentation**. The word is chosen due to the fact that more segments than semantic groups are used to segment the image and the segmentation job was ”overdone”. Segments of an oversegmentation are commonly called **superpixels**.

The central idea of an oversegmentation is the simplification of image segmentation by using less primitives to represent an image. Instead of considering all pixels of the dense image grid similar pixels are grouped together into superpixels depending on their colour values. Pixels are then represented by the superpixel and the mean colour over all pixels in the superpixel (see fig. 2.3). Good superpixels have a low variance in appearance such that the mean colour is a good representation of individual pixels. This guarantees that only very little information is lost and that superpixels can be used to represent the full image.

Superpixels can also be viewed as an intelligent downsampling of an image in the sense that the downsampling is not done over rectangular regions like for mip maps, but on arbitrarily shaped regions which are chosen depending on the actual content of the image. This especially abets superpixel which respect object edges and do not introduce a smoothing effect typical to downsamplings. Fig. 2.3 compares a possible segmentation from a superpixel algorithm like SLIC [5] to a naive downsampling of the image. Though the naive downsampling uses more than double the number of superpixels, it does not represent the image contents as well as intelligent superpixels. Important borders are not captured

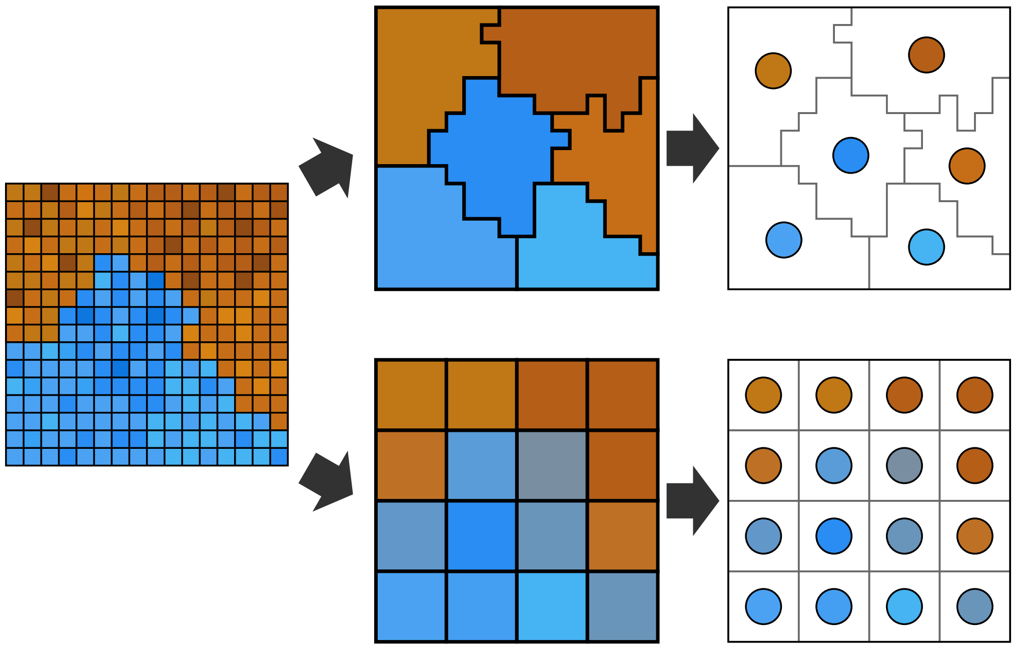


Figure 2.3: **Top:** An intelligent oversegmentation respect edges. **Bottom:** Classical down-sampling yields poor oversegmentations.

and colours are smeared out when building the mean over pixels with strongly different appearance.

When each superpixel remembers the set of pixel locations it is representing, the underlying image can be reconstructed with very high quality. Fig. 2.4 demonstrates the power of superpixels when representing an image by comparing superpixel segementations with different number of pixels. In this example the original image consists of 590000 pixels and is captured well by an oversegmentation with only 8000 superpixels – this is two magnitudes less. More examples of superpixel segmentations for different scenarios are shown in fig. 2.5.

The central advantage of superpixels is their ability to reduce the number of primitives from a high number of pixels to a much lower number of superpixels. This enables complex algorithms which have high runtime requirements and thus superpixels have been used with great success in many applications in computer vision in the recent years: Wang et al. [78] uses a local segmentation into superpixels to train an object appearance model and to track the moving object over time. Cheng et al. [16] uses a superpixel segmentation to compute image saliency with a global method which would be infeasible when applied directly to the full pixel grid. Arabelaez et al. [7] use an image oversegmentation to improve the quality of a local image boundary detector by suppressing noise and by providing an ultrametric contour map which can be used to smoothly vary the resolution of an image segmentation. As presented later in §3, superpixels can be extended to the three-dimensional space to simplify the segmentation of 2.5D point clouds [81].

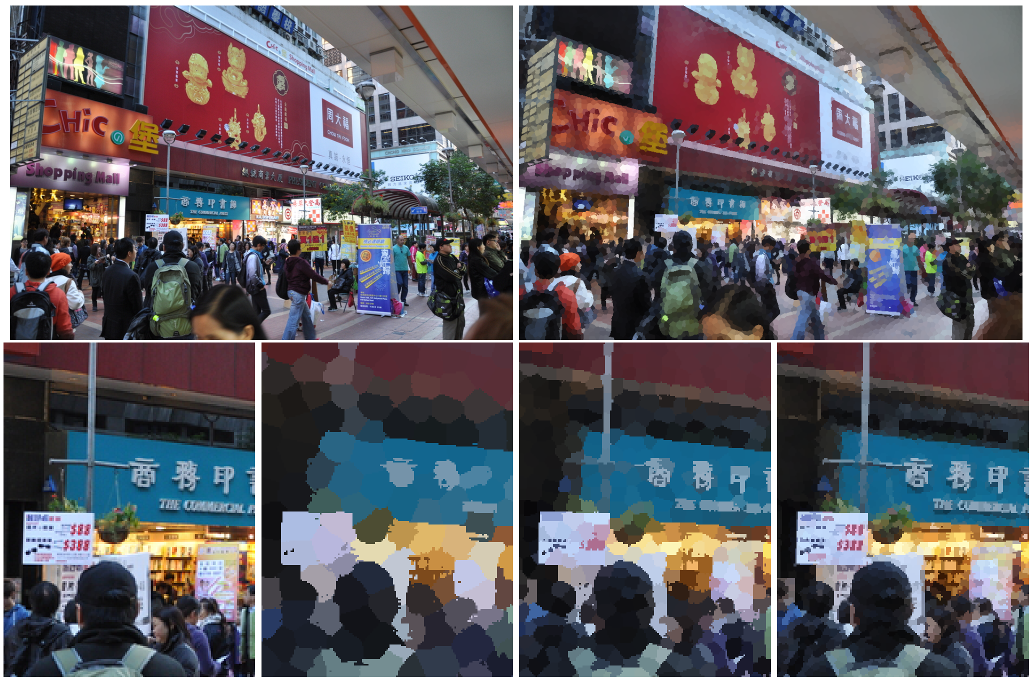


Figure 2.4: **Top row:** Original image and SLIC superpixels [5] with 8000 superpixels. **Bottom row:** Close up with 2000, 8000 and 32000 superpixels.

For many applications several properties of superpixels are advantageous:

Compression Pixels have to be represented well by superpixels, thus the variance in appearance of pixels gathered in one superpixel should be low.

Locality Pixels of one superpixel should not be distributed over the whole image but concentrate on a small local region.

Conservation of edges Following the principle of inter-region similarity superpixels should respect strong edges in the image to increase the probability that the boundaries between semantic groups are preserved.

Connected Often it is not desired that superpixels are not connected in a graph-theoretical sense and have small enclaves completely surrounded by pixels from other superpixels.

Uniform size and coverage This is a stronger property that aims to distribute superpixels of equal size uniformly over the image.

Compactness An even stronger property that additional requires superpixel to have spherical shape and retain the minimum possible isoperimetric quotient.

Runtime As superpixels are in most cases only an intermediate step their computation should not be time consuming.

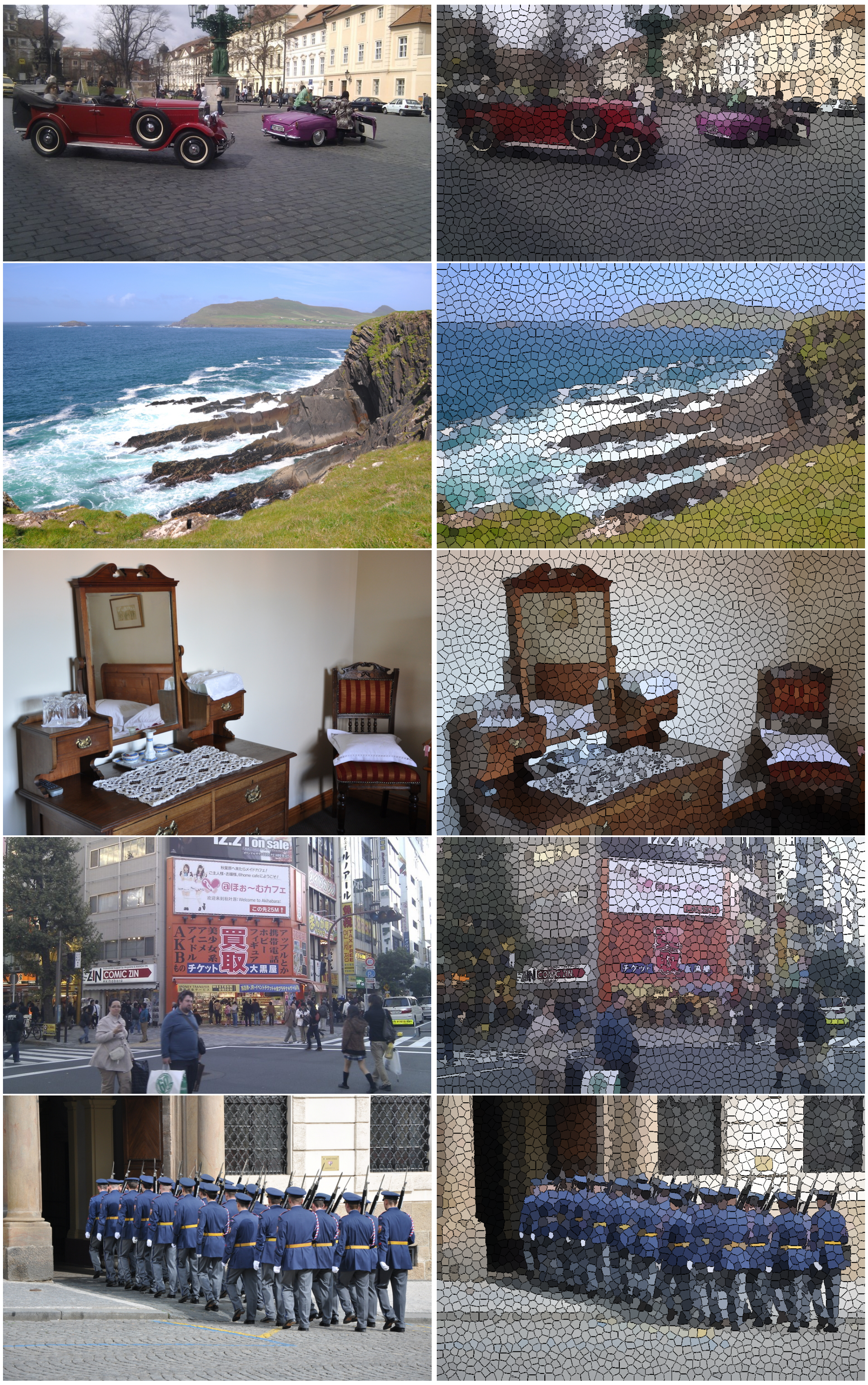


Figure 2.5: Several example images and corresponding superpixels computed with the SLIC algorithm (§2.1.3. The number of superpixels is 2000 for all three examples.

These properties are often conflictive and can not be satisfied all at once. In §2.1.3 several state-of-the-art superpixel methods are presented and their goals with respect to superpixel properties are discussed.

When measuring the quality of superpixels with respect to the presented qualities one has to carefully consider the implications of a specific metric. For example, when trying to only minimize the variance in pixel feature values (i.e. pixel colour), superpixel tend to be extremely distributed over the whole image without guaranteeing local spatial coherence. On the other side, the quality of an intra-segment similarity segmentation can be increased by reducing the size of segments. The smaller segments are the less pixels a segment has to represent, and thus the smaller the variance of pixel information and the higher the probability that they belong to the same semantic group. Indeed, the segmentation where each pixel is a segment on its own is already a segmentation which perfectly satisfies intra-segment similarity. Common metrics to measure the quality of segmentations are collected in §A.1.

2.1.3 State-of-the-art superpixel methods

In the recent years many different superpixel algorithms have been proposed [77, 72, 18, 64, 87, 26, 76, 48, 5, 63, 88, 68] which use different mathematical models and focus on different desirable properties. Here three recent methods are presented in more details:

TurboPixels

TurboPixels [48] is a superpixel method using geometric flows to compute an image oversegmentation. The method is designed to provide compact and connected superpixels which are distributed uniformly over the image. Superpixels are incrementally grown from seed points and boundary growth is slowed down in the vicinity of edges. The geometric flow is formulated such that it is attracted to image edges while still producing smooth superpixel boundaries. The number of superpixels can be controlled explicitly. One of the major disadvantages of TurboPixels is the high computational costs of the algorithm. Reasonably sized images required a runtime of several minutes which makes the algorithm unsuitable for realtime applications.

Homogenous Superpixels from Random Walks

Homogeneous superpixels from random walks [63] uses Markov Clustering (MCL) to compute compact and connected superpixels. Markov Clustering is a generic graph clustering method based on stochastic flow circulation. Originally the method is infeasible for huge graphs like the pixel lattice graph, as it requires the iterative multiplication of the adjacency matrix. Due to this fact the authors present an intelligent pruning method which makes the algorithm more efficient while also adding a compactness constraint. However, the algorithm still requires a runtime of several seconds for small images. Additionally the number of superpixels can not be controlled explicitly, only the size can be controlled indirectly be adapting the pruning method.

Simple Linear Iterative Clustering (SLIC)

SLIC superpixels [5] is a method which produces uniformly distributed and uniformly sized superpixels. The method does not enforce compactness and connectivity rigorously. The algorithm is essentially a k-mean algorithms which uses the spatial distance of pixels as a regularization term to promote compactness. This compactness term establishes a certain degree of compactness and connectivity but does not enforce it. The compactness of superpixels can be controlled by the user with a single parameter. The algorithm can be implemented very efficiently – the typical runtime is in the order of 100 milliseconds. Like with TurboPixels, the number of superpixels can be controlled explicitly.

2.2 Adaptive Superpixels

While there are already many superpixel algorithms, none of them allows an explicit control over the global distribution of superpixels. With some it is possible to specify the total number of superpixels, but it is not possible to specify regions of interest which should have a higher density of superpixels and other less important regions with a lower density. Thus a novel oversegmentation method, *Adaptive Superpixels* (ASP), is introduced which generalizes the idea of the SLIC superpixel algorithm by allowing control not only over the number of superpixels, but also the global distribution of superpixels. In *Adaptive Superpixels* the distribution of superpixels is governed by an user defined density function which indicates the probability that a superpixel shall be placed at a certain pixel. The SLIC algorithm corresponds to the case of a constant superpixel density.

The density function can serve many purposes of which some are investigated in §2.5.2 The important application of *Adaptive Superpixels* in this context is the extension to *Depth-Adaptive Superpixels* in §3, a superpixel algorithm for RGB-D images which combine colour and depth information. There, the density function will be used to distribute superpixels uniformly over the 3D surface thus guaranteeing several desirable properties *in 3D*. Fig. 2.6 shows two examples of density functions and corresponding results from the *Adaptive Superpixels* algorithm. For now it is assumed that the superpixel density is arbitrary and given by the user.

The two steps of the *Adaptive Superpixels* algorithms are as follows:

1. Sample initial superpixel centres from the given density function using *Simplified Poisson Disk Sampling* (*SPDS*) - an efficient Poisson disc sampling method.

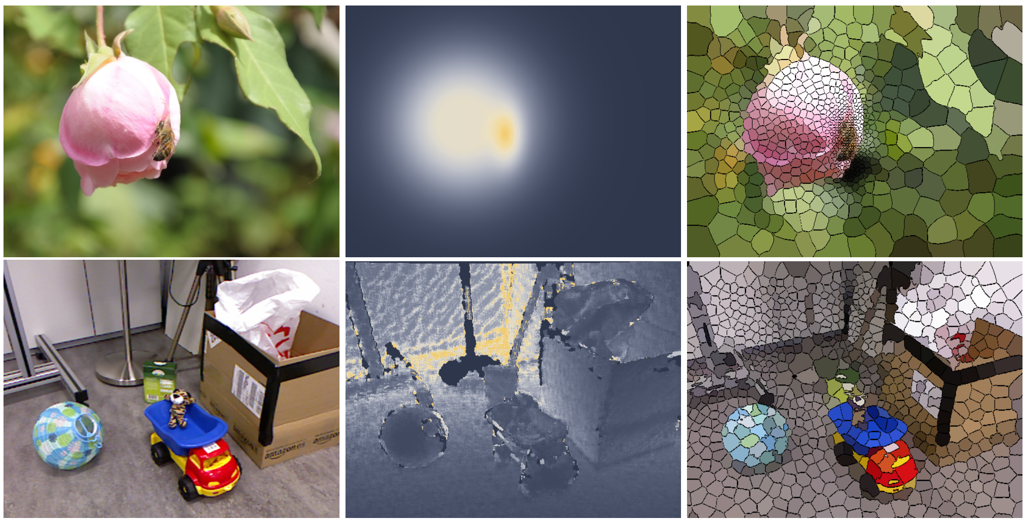


Figure 2.6: **Top:** A closeup to a region of interest, and **bottom:** distribution of superpixels according to pixel depth. Original color image (**left**), user defined density function (**middle**) and adaptive superpixels (**right**) are displayed.

2. Assign pixels to superpixels using *Density-Adaptive Local Iterative Clustering* (*DALIC*) with a density-dependent compactness term in the distance metric.

Poisson disc sampling describes a point distribution where the distance between points is not arbitrary but a fixed constant. For a 2D plane, this corresponds to many flat discs arranged such that they fill the plane. Often it is desirable to use discs with soft edges in order to squeeze them a bit together and get slightly irregular distributions. Poisson Disc sampling will be investigated in more detail in §2.3.

Density-Adaptive Local Iterative Clustering is a variation of k-means clustering with two distance measure where one is used as a regularization term. The other measure is the local feature distance measure used for computing point-cluster similarity. The regularization term governs the reach of a cluster point and guarantees that points are only assigned to local clusters. This term results in compact clusters and a highly efficient clustering algorithm with a runtime linear in the number of pixels and independent of the number of clusters. Details will be presented in §2.4.

During the first step an approximation to the global positioning of superpixels is computed by using a simplified multi-layer Poisson Disc sampling method. The distribution of cluster centers shall follow the given density function and Poisson disc sampling guarantees that clusters are evenly distributed according to the desired density. In the context of *Adaptive Superpixels*, an approximation is sufficient as superpixel placement will be further influenced during the second step. Here pixels are iteratively assigned to superpixels by considering only a local neighbourhood dependent on the local density values. Both steps work hand in hand to assure that the final positions of superpixel centres are distributed according to the given density function while assuring that the generated superpixels are a good representation of the given image.

Using Poisson disc sampling for the initial placement of superpixel seeds is an important requirement to guarantee a probably correct placement of superpixels defined by the density function. Additionally the quality of generated superpixels is not only higher, but it can also be achieved with a notable smaller number of iterations during *DALIC*. In fact, the isoperimetric quotient and the uniform distribution error under the given superpixel density (see §A.1) is per definition already almost perfect for the initial seed points. The density-dependent compactness term in the distance function ensures that superpixels are globally distributed corresponding to the density function. Both, initial density-dependent Poisson distribution of superpixels and density-dependent metric in nearest neighbour pixel to cluster assignment, assure that the final superpixel distribution corresponds to the given density functions.

For the following sections some basic notations will be used. The *Adaptive Superpixels* algorithm will run on a finite domain $\Omega \subset \mathbb{R}^n$ which is essentially required to be a subset of the n-dimensional Euclidean space. The prominent example for a domain, which will be used for examples and visualizations in this chapter, is be the dense, rectangular pixel grid of an image. Another domain used later in §5 is the 3-dimensional spatio-temporal domain of a video stream. The user-defined superpixel density is denoted as $\rho : \Omega \to \mathbb{R}_+$ which assignes to each point in the domain a positive number indicating the probability that a cluster should be placed there. Additionally each point in the domain is annotated with a feature vector $\mathfrak{f} : \Omega \to \mathscr{F}$. Examples for feature spaces $\mathscr{F}$ are a colour space, e.g. the RGB space, or a combination of colour and depth for combined colour and depth sensors (RGB-D sensors). Another possibility could be point normals which are computed from 3D point positions. The features space needs to be equipped with a metric $d_{\mathscr{F}} : \mathscr{F} \times \mathscr{F} \to \mathbb{R}_+$ which expresses similarity between feature vectors. It will also be necessary to compute mean values for feature vectors. For this purpose we will use an Abelian notation for feature vectors using the $\sum$ and $+$ signs. For feature values from a non-linear group, like for example normal vectors, specific algorithms for the computation of the mean have to be used.

2.3 Poisson disc sampling

2.3.1 Poisson disc point distributions

The Poisson disc distribution describes a point distribution where points are spaced with a minimal distance between each other. The placement probability of points under a Poisson disc distributions thus depends on the distance to neighbouring points. This stands in contrast to a white noise distribution which distributes points randomly without considering other points. Poisson disc point sampling had important applications in halftoning where comparable smooth

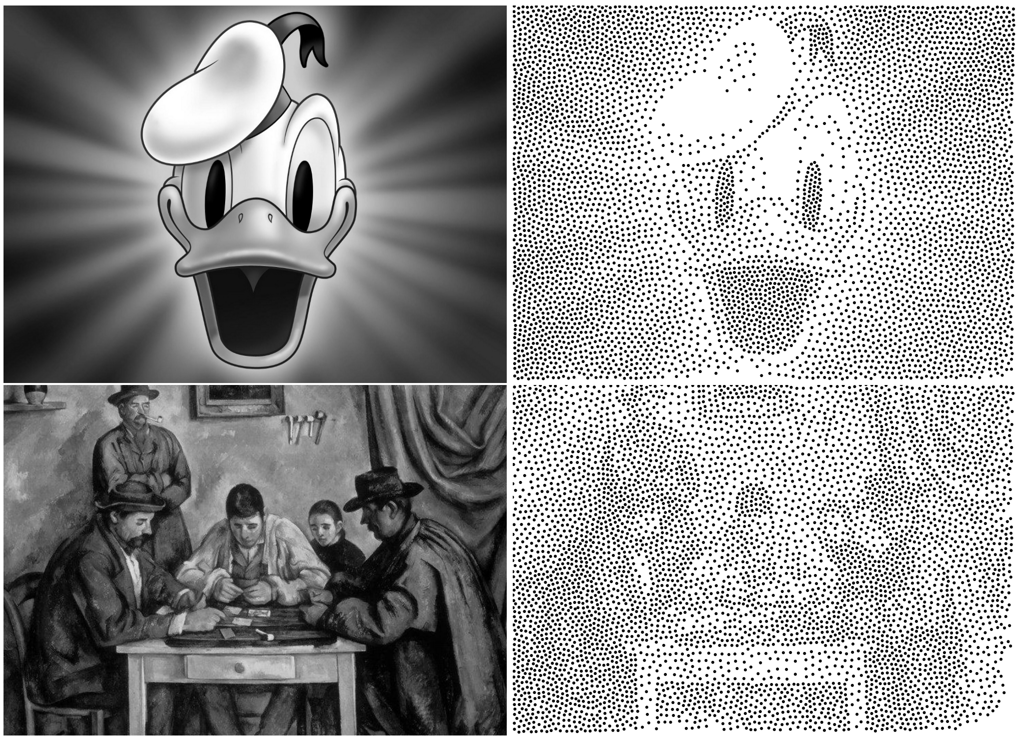


Figure 2.7: Examples for halftoning of grayscale images (image courtesy [4, 13]) using Poisson disc point samples generated with Fattal's method [25].

images had to be converted to point distributions in order to be printable without visible artifacts using individual dots of ink. Today they are for example used in image raytracing or other computer graphics applications to reduce the number of rays or samples which need to be evaluated for artefact-free colour or lighting computations. Fig. 2.7 shows examples for grayscale images and corresponding points generated by the Poisson disc sampling method of Fattal [25].

A primitive method to compute a Poisson disc distribution is dart throwing. A random point is sampled accordingly to the density distribution and placed if it is not nearer than the required minimal distance to all other points, otherwise it is rejected. The process is iterated until no more points can be placed. This methods suffers from critically slowing down towards the end as the probability to reject a point increases the more points are already placed.

The quality of Poisson point sampling methods can be easily visualized by computing the power spectrum. The spectrum of a Poisson disc distribution has a blue noise characteristics. Blue noise distributions have the property that ”low frequencies are adequately captured and high frequencies are scattered into noise” [25]. Good methods have a power spectrum without low frequencies and where high frequencies are uniformly distributed. Fig. 2.8 displays various point distributions and displayes the corresponding power spectra.

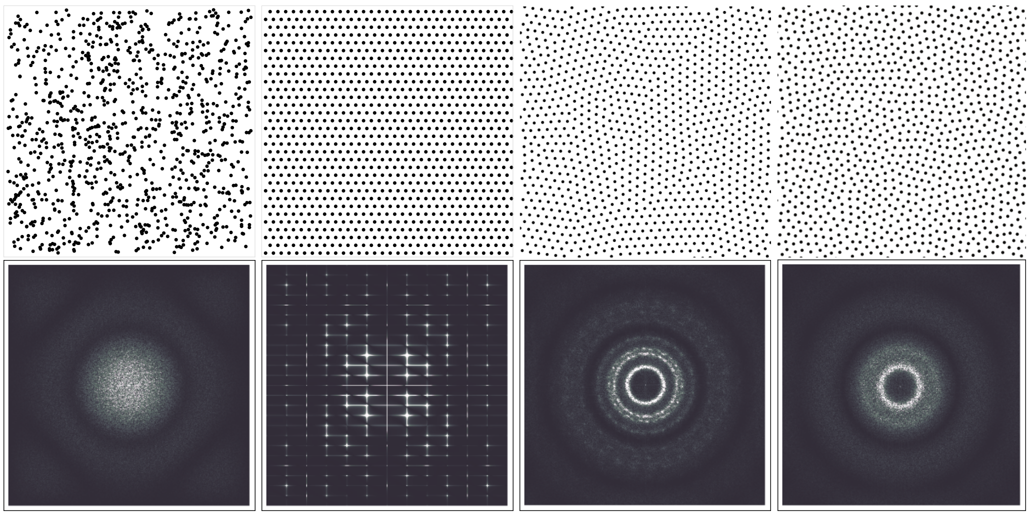


Figure 2.8: The first row shows point distributions for different methods and the second row the corresponding spectra. **Left**: Random (white noise) sampling. **Middle left**: Placement of points on a hexagonal grid. **Middle right**: Point sampling using Llodys method [51]. **Right**: Point sampling using the method of Fattal [25].

Several methods have been proposed to sample Poisson disc point distributions [51, 23, 9, 69, 25], and in this context the method of Fattal [25] is explained in detail in §2.3.2. He proposed an efficient hierarchical method which produces slightly noisy samples with excellent spectral properties. However the method is quite slow as it aims towards high-quality point distributions which is not a requirement for the purposes of *Adaptive Superpixels*.

For the *Adaptive Superpixels* algorithm the notion of a density function is required. In the language of halftoning the density function corresponds to the image intensity level: Dark image areas have a high density and require a high number of points where bright regions have a low density and require only few points (fig. 2.7). For our purposes the following formal definition of a density function will be used:

Definition 2. Let $\Omega \subset \mathbb{R}^n$ finite and $\rho : \Omega \to \mathbb{R}$ a function. ρ is called a **density function** of weight N iff.

i) $\forall\, x \in \Omega : \rho(x) > 0$,

ii) $\int_\Omega \rho(x)\, dx = N$ and

iii) ρ is sufficiently smooth (e.g. Lipschitz continuous).

Lipschitz continuity is defined over

$$f \text{ is Lipschitz continuous iff. } \exists\, L \in \mathbb{R}_+ : \forall\, x, y : |f(x) - f(y)| \leq L \|x - y\| \tag{2.1}$$

and is a stronger assumption than continuity which is defined over

$$f \text{ is continuous iff. } \forall\, x : \forall\, \epsilon > 0 : \exists \delta > 0 : \forall\, y : \|x - y\| < \delta \Rightarrow |f(x) - f(y)| < \epsilon\,. \quad (2.2)$$

Details are explained in [28].

In the following the state-of-the-art Poisson sampling technique of Fattal will be reviewed. Afterwards a simplified but highly efficient Poission Disk sampling method, *Simplified Poisson Disk Sampling*, is presented which is optimal for the purposes of *Adaptive Superpixels.*

2.3.2 Multi-layer sampling and Fattal's method

Fattal [25] describes an algorithm to draw blue-noise point samples using a multi-scale sampling scheme. Multi-scale sampling schemes build a pyramid of increasingly coarse density functions over the original density function. Initially, few points are distributed on the highest layer using only the lowest frequency of the density functions. For each layer points are sampled and optimized using the Langevin method and a Metropolis-Hastings correction. This process is repeated iteratively for lower levels, i.e. higher frequencies, by using the point configuration of the previous step as a initial configuration and repeating the optimization step. The author demonstrates that his method is computationally efficient, however especially the corrected Langevin step is time-consuming for a large number of points.

One of the key concepts in Fattal's method is the approximattion of the density function with a set of kernel functions positioned at point positions. The kernel function K is required to be positive, finitely-integrable and monotonically decaying away from zero [25]. A common choice is the Gaussian kernel :

$$\mathrm{K} : \mathbb{R} \to \mathbb{R}_+, x \mapsto e^{-\pi x^2}\,. \quad (2.3)$$

Kernel basis functions can be used to create a isotropic "density blob" for the domain Ω by setting

$$\mathrm{K}_\Omega(\cdot\,|\,\mu,\sigma) : \Omega \to \mathbb{R}_+,\ u \mapsto \frac{1}{\sigma^D}\mathrm{K}\left(\frac{\|u-\mu\|_\Omega}{\sigma}\right) \quad (2.4)$$

where D is the dimension of the domain Ω, μ is the position of the sample and σ its scale.

K_Ω can be used to approximate an arbitrary density function by using a sum of such kernel basis functions:

Definition 3. Let ρ be a density function and $\{u_i\}_{1=i}^N \subset \Omega$ a set of points. The density approximation function A_ρ is defined as

$$\mathrm{A}_\rho\left(u\,|\,\{u_i\}_{1=i}^N\right) := \sum_{i=1}^N \mathrm{K}_\Omega\left(u\,|\,u_i, \rho(u_i)^{-D}\right). \quad (2.5)$$

The approximation error is computed as the integrated absolute difference between the approximate density function and the desired density function:

$$\mathrm{E}_\rho\left(\{u_i\}_{j=1}^N\right) := \int_\Omega \left|\mathrm{A}_\rho\left(u \,|\, \{u_i\}_{1=i}^N\right) - \rho(u)\right| \mathrm{d}u. \tag{2.6}$$

During Fattal's method, the kernel positions $\{u_i\}_{j=1}^N$ are optimized by minimizing the error E_ρ while preserving a certain desired degree of randomness to guarantee good blue-noise properties.

Many Poisson disc sampling methods show the phenomena of critically slowing down towards the end of the sampling process. As with the most simple dart throwing method, it becomes increasingly difficult to add necessary points when a large share of points is already placed at good positions. Fattal handles this problem by using a multi-scale sampling scheme. The key advantage of a multi-layer method is the devision of the problem into sub-problems with respect to the frequencies of the density function. For each sub-problem, i.e. each frequency band, solutions can be found quickly by using the solution of the previous problem, i.e. lower frequencies, as a starting point.

For a concrete implementation, so called ”mipmaps” of the density function are computed where each mipmap is a down-sampled version of the mipmap from the previous layer. The lowest layer is the original density function and the highest layer shall have a specified minimal resolution. For each layer, points are moved to optimize the approximation error for the density function on the current layer until a convergence criterion is met. Then the points are transferred to the following layer by possibly splitting them into multiple points depending on the local point density and optimizing their positions again. The process is repeated until the lowest layer is reached.

2.3.3 Simplified Poisson Disc Sampling

In the context of *ASP*, there are noticeable similarities between the Langevin step for cluster relaxation and *Density-Adaptive Local Iterative Clustering*. In Fattal's method, Langevin relaxation is applied at each layer of the constructed pyramid of increasingly down-sampled versions of the density function to relax clusters in a position approximating the density function in an optimal way. In a similar way, superpixel centres are automatically shifted into a Poisson Disc cluster configuration during the iterative pixel-to-cluster assignment of *DALIC* due to the density-adaptive compactness term (see §2.4). Additionally, *DALIC* tries to balance the correct distribution of cluster centres with respect to the density function against an optimal pixel-to-cluster allocation which minimizes the error in the chosen metric on pixel feature values.

However like k-means clustering, *DALIC* does known how and where to create points, but only how to move them to good positions. Thus a method to initially distribute points is required. The quality of the initial point distribution plays a crucial role in the number of required iterations and the final distribution quality which can be achieved in a reasonable amount of time. *DALIC* can only provide good distributions when the initial point configuration is a good approximation satisfying the lower frequencies of the density function. The method of Fattal provides excellent Poisson Disc samplings with excellent spectral properties at the cost of complex calculations and thus a high algorithm runtime. This quality is not required for *DALIC* and a *Simplified Poisson Disk Sampling* (*SPDS*) method is presented which uses the idea of multi-scale sampling but leaves the final optimization of point positions to the second phase of the *Adaptive Superpixels* algorithm.

Simplified Poisson Disk Sampling proceeds by constructing a pyramid of density functions and a corresponding tree structure on the pyramid. The tree is processed by checking the remaining density for each node at a given layer: If it is high enough, the procedure is iterated with the children at the next layer, otherwise a number of points is placed. Points are placed according to the node density distribution by using a placement probability proportional to the density of the layer with the highest resolution. In comparison to [25], point relaxation is not performed and instead handled later by *DALIC*. *Simplified Poisson Disk Sampling* is outlined as follows:

1. Discretise the density function to a regular rectangular grid.
2. Compute a hierarchical tree of increasingly down-sampled density mipmaps where each node corresponds to a local region of the grid.
3. Starting with the root node process the whole tree and decide if points need to be placed. Place points according to the original density function.

In the following it is assumed, that the density function is already discretised and thus defined on a finite, D-dimensional grid $\{0,\ldots,2^Q-1\}^D \subset \mathbb{Z}^D$ where, for simplicity and without loss of generalization, the number of grid points in each dimension is equal and a power of 2. Starting with the desired density function $\rho_0 = \rho$ a pyramid of down-sampled density functions $(\rho_l)_{l=0}^{Q}$ is computed (see fig. 2.9):

$$\rho_l : \left\{0,\ldots,2^{Q-l}-1\right\}^D \to \mathbb{R}_+,\, z \mapsto \sum_{e\in\{0,1\}^D} \rho_{l-1}(2z+e)\,,\ \forall\, 1 \le l \le Q \tag{2.7}$$

The tree is constructed by placing a node at each grid cell with position z_l for each layer l and connecting it to its parent node at position $\lfloor \frac{z_{l-1}}{2} \rfloor$ in the layer $l-1$ above. Points are placed by processing the tree using a depth-first or breadth-first algorithm starting at the root node and carrying out the following operation for each node:

- If node density $\rho_l(z_l)$ is smaller than or equal to 1: Sample a uniform number r in $[0,1]$. If $r \le \rho_l(z_l)$ place a point accordingly to eq. 2.8. Ignore all child nodes.

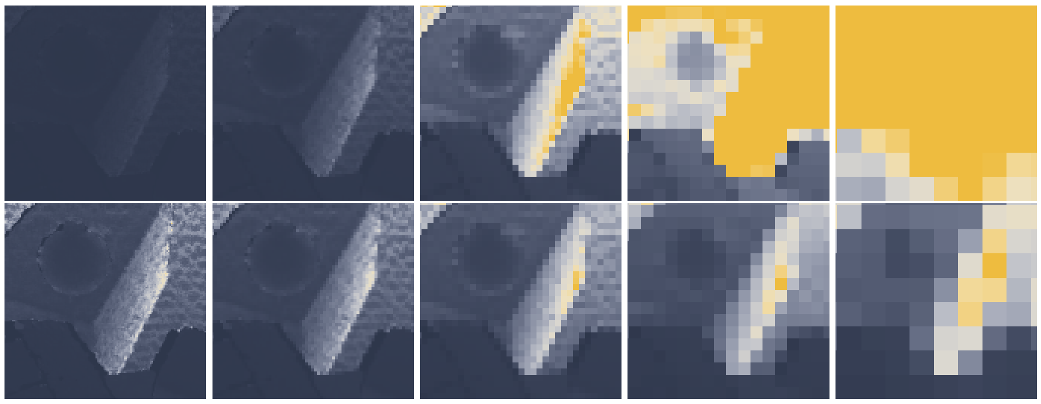


Figure 2.9: **From left to right:** Pyramid of density functions: ρ_1, ρ_2, ρ_3, ρ_4 and ρ_5. **Top**: Density of ρ_i, and **bottom**: Same density ρ_i, but plotted as normalized to $[0,1]$.

- Else: Continue to child nodes.

If a point is placed for a node z_l at layer l, it is positioned at a random location inside the node depending on the actual density distribution in ρ_0. Specifically, the tree node z_l corresponds to the volume $V(z_l) := \{2^l z_l + [0, 2^l - 1]^D\}$ at the highest resolution level ($l = 0$). The probability to select point $v \in V(z_l)$ inside the volume is chosen to be proportional to $\rho_0(v)$. In other words:

$$P_{\text{place}}(x|z_l) \propto \begin{cases} \rho_0(x) & \text{if } x \in \{2^l z_l + [0, 2^l - 1]^D\} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

The method is summarized in alg. 1. To increase efficiency, lower layers can be omitted as long as the individual density of all nodes of the layer is small enough, e.g. smaller than 1. In the same way high layers can be omitted as long as the density of all nodes is greater than 1.

Fig. 2.10 visualizes *Simplified Poisson Disk Sampling*: To the left, the target density function ρ_0 is shown. In the middle, the multi-layer density tree is visualized by showing nodes which have generated a point during the processing of the density tree and, to the right, the corresponding sample points are displayed.

The distribution of points can be further optimized by reducing the randomness when deciding whether to place a point in a cell or not. Instead of throwing a dice whenever the node density is smaller than 1 one can employ the following scheme which tries to regularize the number of placed points:

- If node density is smaller than or equal to 2^D: Randomly round the node density to an integer n using eq. 2.9. Randomly select n child nodes weighted by child node density and place a point into the child node using eq. 2.8. Then ignore all further child nodes.

- Else: Continue to child nodes.

Algorithm 1 *Simplified Poisson Disk Sampling*

```
Require: Density function $\rho_0$
  ▷ Compute density pyramid
  for $1 \le l \le Q$ do
      $\forall z \in \{0,\dots,2^{Q-l}-1\}^D : \rho_l(z) = \sum_{e\in\{0,1\}^D} \rho_{l-1}(2z+e)$
  end for
  ▷ Process tree and create points
  Recursively execute the following for each point $z_l$, starting with $z_Q = 0$:
  if $\rho_l(z_l) \le 1$ then
      ▷ Decide if a point shall be placed
      if $\mathcal{U}(0,1) \le \rho_l(z_l)$ then
          Place one point under probability $P_{\text{place}}(x|z_l)$ (see eq. 2.8)
      end if
  else
      Continue with children $z_{l-1} \in \{2z_l + e \,|\, e \in \{0,1\}^D\}$ at the next layer
  end if
```

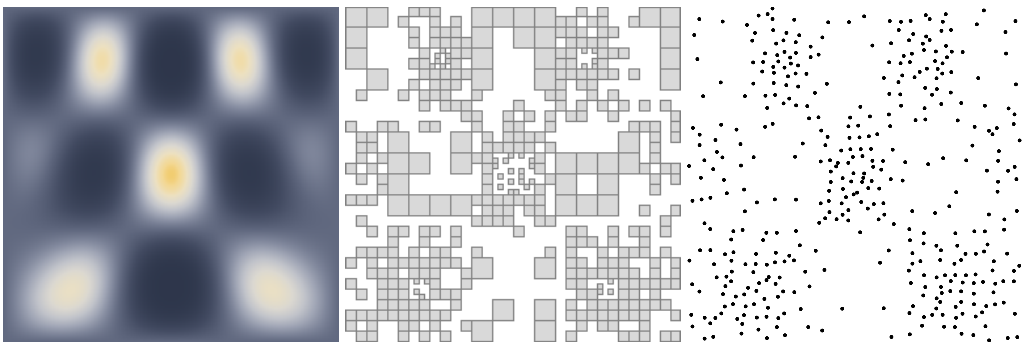


Figure 2.10: **Left**: Density function ρ_0. **Middle**: Excerpt of the pyramid density tree. Each rectangle represents a node which has generated a point. **Right**: Corresponding point samples generated with *Simplified Poisson Disk Sampling*.

Random rounding is done as follows:

$$\text{randomround}(x) := \begin{cases} \text{ceil}(x) & \text{if } \mathscr{U}(0,1) \leq x - \text{floor}(x) \\ \text{floor}(x) & \text{otherwise} \end{cases} \tag{2.9}$$

where $\mathscr{U}(0,1)$ is a random sample from a uniform distribution over $[0,1]$.

The algorithm for the improved version of *SPDS* is listed in alg. 2.

Algorithm 2 *Simplified Poisson Disk Sampling* (improved)

```
Require: Density function $\rho_0$
  ▷ Compute density pyramid like in alg. 1
  ▷ Process tree and create points
  Recursively execute the following for each point $z_l$, starting with $z_Q = 0$:
  $C = \{2z_l + e \,|\, e \in \{0,1\}^D\}$
  if $\rho_l(z_l) \leq 2^D$ then
      $\forall c \in C: \ w_c = \rho_{l-1}(c)$
      $n = \text{randomround}(\rho_l(z_l))$
      Randomly select $n$ child nodes by weights $w_c$
      Place one point for each selected child $c$ under probability $P_{\text{place}}(x|c)$
  else
      Continue with children $z_{l-1} \in C$ at the next layer
  end if
```

2.4 Density-Adaptive Local Iterative Clustering

k-means clustering [52] is a well known algorithm which iteratively assigns points to the best cluster $a(u)$ by comparison point features $f(u)$ with cluster features (f_i)

$$\forall\, u \in \Omega: \ a(u) := \operatorname*{argmin}_{1 \leq i \leq k} d_{\mathscr{F}}(f(u), f_i)$$

and updates cluster features vectors with the mean of assigned points

$$\forall\, 1 \leq i \leq k: \ f_i = \text{mean}(\{u \in \Omega | a(u) = i\}).$$

Assignment and update are repeated iteratively until convergence of the assignment function a or other stopping criteria are fulfilled. Initial placement of clusters can be problematic and most often random distribution in the feature space are chosen. Additionally k-means clustering has several disadvantages when computing superpixels:

- It does not produce spatially compact clusters as pixels with similar feature vectors may be located in completely different image regions and often are not necessarily part of the same segment.

- It has a runtime of $\mathcal{O}(k\,|\Omega|)$, thus if Ω is a pixel grid the runtime is linear in the number of pixels times the number of clusters.
- The spacial distribution of cluster over the image can not be controlled.

In the following *Density-Adaptive Local Iterative Clustering* (*DALIC*) is presented which will tackle theses problems. Foremost, *DALIC* directly uses a provided density function to control cluster distribution. The density-function is additionally used in a density-adaptive compactness term to encourage the creation of compact superpixels. Moreover by exploiting several properties of Poisson disc sampling and density functions it can be demonstrate that the algorithm runtime can be reduced to only be linear in the number of pixels and independent from the number of clusters.

An important quantity in the following will be the radius R_n of an n-dimensional ball which can be derived by considering the volume of an n-dimensional unit ball [79].

Definition 4. $R_n(V)$ is defined as the radius of an n-dimensional ball of volume $V \in \mathbb{R}_+$.

Basic cases are well known mathematically facts:

$$R_2(V) = \sqrt{\frac{V}{\pi}} \tag{2.10}$$

$$R_3(V) = \left(\frac{3V}{4\pi}\right)^{\frac{1}{3}} \tag{2.11}$$

The radius of a n-dimensional ball will be especially interesting in combination with a density function:

Definition 5. Let ρ be a density function over $\Omega \subset \mathbb{R}^n$. The **density dependent radius** is defined as:

$$R_\rho : \Omega \to \mathbb{R}_+,\ u \mapsto R_n\left(\frac{1}{\rho(u)}\right)$$

The density-dependent radius $R_\rho(u)$ is exactly the radius of a ball of volume $\frac{1}{\rho(u)}$ which corresponds to the volume in which a perfect Poisson disc sampling technique would in average place exactly one point. Examples are:

$$\dim(\Omega) = 2 \Rightarrow R_\rho(u) = \frac{1}{\sqrt{\pi\,\rho(u)}} \tag{2.12}$$

$$\dim(\Omega) = 3 \Rightarrow R_\rho(u) = \left(\frac{4\pi}{3}\,\rho(u)\right)^{-\frac{1}{3}}. \tag{2.13}$$

The density function can be understood as a local distortion of the domain Ω. Such a distortion can also be described by using a local distance function.

Definition 6. Let $u_0 \in \Omega$. We define:

$$d_\rho(\cdot|u_0) : \Omega \to \mathbb{R}_+,\ d_\rho(u|u_0) := \frac{\|u - u_0\|}{R_\rho(u_0)}$$

The fact that this local distance function describes the density-adaptive distortion of Ω is described in the following proposition.

Lemma 1. *Let ρ be a density function over Ω with weight N. Let $u \in \Omega$, then the local distance function $d_\rho(\cdot|u)$ assures that in a unit ball $B(u) := \{x \in \Omega \,|\, d_\rho(x|u) \leq 1\}$ around any u there is in average exactly one Poisson disc sampled point.*

Proof. A density of $\rho(u)$ indicates that there is in average one point in a volume of $V(u)$, thus $\rho(u) = \frac{1}{V(u)}$. As $R_\rho(u)$ is constant for a given u, the radius of the ball $B(u)$ is exactly $R_\rho(u)$. We can conclude, that a Poisson disc sampling algorithm should in average sample exactly one point in the ball $B(u)$. □

Thus instead of an unweighted compactness term $\|u - u_0\|$ for achieving uniform, equally sized clusters corresponding to a constant density, the distorted, local compactness term $\sqrt{\pi\,\rho(u_0)}\,\|u - u_0\|$ ($\dim(\Omega) = 2$) derived from the density function is used. Note that for a constant density the distorted, local compactness term is constant over Ω and thus identical to the standard compactness term. The density-adaptive feature metric used in the *Adaptive Superpixels* algorithms is thus defined as follows:

Definition 7. Let $\mathscr{F}$ be a metric space with metric $d_\mathscr{F} : \mathscr{F} \times \mathscr{F} \to \mathbb{R}_+$. Let $u_0 \in \Omega$ and $f_0 \in \mathscr{F}$ a corresponding feature vector. Let $u \in \Omega$. The local **density-adaptive metric** d_{DA} is defined as

$$d_{\text{DA}} : \Omega \to \mathbb{R}_+,\ d_{\text{DA}}(u\,|\,u_0, f_0) := d_\mathscr{F}(f(u), f_0) + d_\rho(u|u_0) \tag{2.14}$$

The density-adaptive compactness term d_ρ assures that a point $u \in \Omega$ is only assigned to a spatially near cluster. The feature distance term $d_\mathscr{F}$ on the other hand is responsible for assigning a point to a cluster which can represent its feature value $f(u)$.

The density-adaptive compactness d_ρ term is also responsible for the better runtime behaviour of *DALIC*. When $d_\mathscr{F}$ is bounded over $\mathscr{F}$, each cluster point has a maximum reach which is defined by the density-adaptive term d_{DA} and directly depends on the local density. Thus during the assignment step, for each point only clusters which can reach this point have to be considered.

This formulation can be reversed by stating that for each cluster only points which are in its reach have to be tested for possible assignment. This simplifies the process as points are normally dense but clusters are not. Thus efficiently testing all points around a cluster corresponds to testing a local neighbourhood which can be simplified to a rectangular region for dense pixel grids. Whereas testing all clusters around each point would require a spatial index method for an efficient access to the nearest clusters. Alg. 3 shows a possible implementation of the *Density-Adaptive Local Iterative Clustering* algorithm.

Algorithm 3 *Density-Adaptive Local Iterative Clustering* (*DALIC*)

```
Require: Density function $\rho : \Omega \to \mathbb{R}_+$ over a finite domain $\Omega$
Require: Feature map $\mathfrak{f} : \Omega \to \mathscr{F}$ and feature metric $d_{\mathscr{F}}$
Require: Initial cluster centers $\{u_i\}$ and feature vectors $\mathfrak{f}_i, 1 \le i \le N$
Require: Parameters: Maximal cluster reach $C$ and number of iterations $K$
  for $k = 1 \to K$ do
    ▷ Initialize minimal distance and cluster assignment
    $\forall\, u \in \Omega : D(u) = \infty, A(u) = 0$
    ▷ Assign elements to clusters
    for $i = 1 \to n$ do
      ▷ Iterate over pixels in search range
      for $\{u \in \Omega \mid \|u - u_i\| \le C\, R_\rho(u_i)\}$ do
        ▷ Assign pixel to nearest cluster
        $d_{iu} = d_{\mathscr{F}}(\mathfrak{f}(u), \mathfrak{f}_i) + \frac{\|u - u_i\|}{R_\rho(u)}$
        if $d_{iu} < D(u)$ then
          $D(u) = d_{iu}$
          $A(u) = i$
        end if
      end for
    end for
    ▷ Update clusters
    for $i = 1 \to n$ do
      $\mathfrak{f}_i = \text{mean}\{\mathfrak{f}(u) \mid u \in \Omega, A(u) = i\}$
    end for
  end for
```

2.5 Evaluation and examples

2.5.1 Evaluation

The *Adaptive Superpixels* algorithm consists of two main parts: *Simplified Poisson Disk Sampling* of initial superpixel centres (see §2.3.3) and *Density-Adaptive Local Iterative Clustering* (see §2.4). As described in §2.2 both methods work together by distributing superpixel centres according to the desired density function. It is important to notice that *DALIC* tries to balance two properties of superpixels: global distribution demanded by the density function and optimal pixel-to-cluster allocation. On the one hand, superpixel centres shall be distributed accordingly to the user defined density function which also guarantees a certain degree of superpixel compactness as cluster have a maximal possible reach. On the other hand, superpixels shall represent the image feature data and thus pixel should be assigned to clusters which can represent the pixel feature vector well with the superpixel mean feature vector. The evaluation in the following will concentrate on the first property. The second property will be thoroughly investigated in the context of *Depth-Adaptive Superpixels* in chapter 3.

An adequate method of measuring the capability of *SPDS* and *DALIC* to represent the desired density function is the total relative density error:

$$E_\rho^{\mathrm{rel}} := \frac{\int_\Omega |E_\rho|}{\int_\Omega \rho} . \tag{2.15}$$

The total relative density error E_ρ^{rel} over the number of iterations for *DALIC* is plotted in figure fig. 2.11 by using the mean result for a set of exemplary density

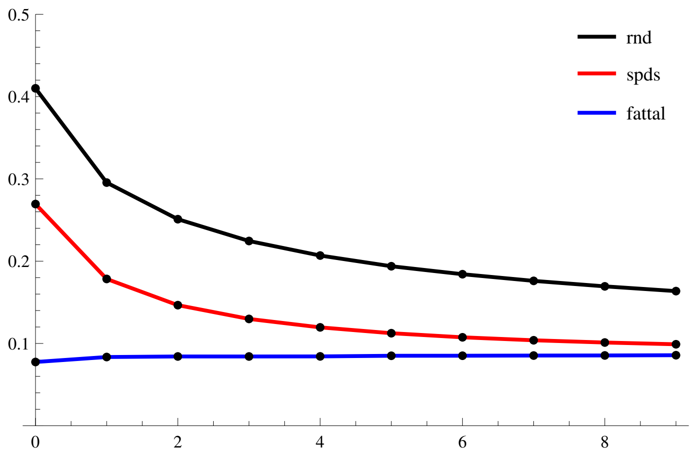


Figure 2.11: Relative density error E_ρ^{rel} averaged over all test density functions against number of iterations of *Density-Adaptive Local Iterative Clustering* when using different Poisson Disc sampling methods for the initial placement of superpixel centres.

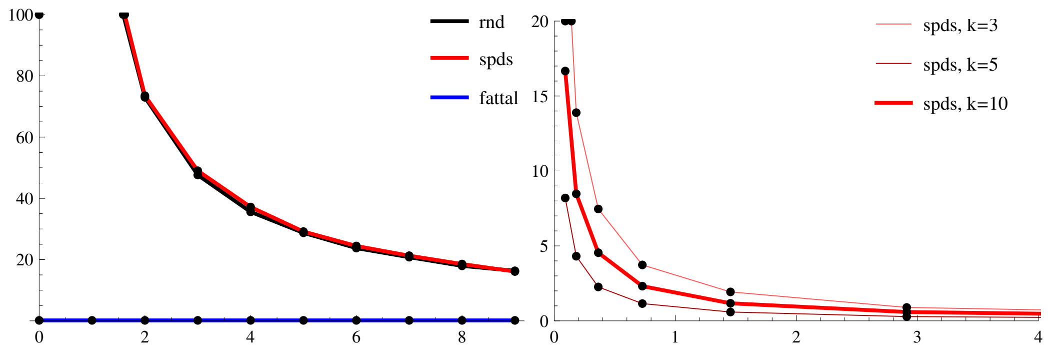


Figure 2.12: **Left:** Total runtime as frames per seconds plotted against the number of *DALIC* iterations. **Right:** Total runtime as frames per seconds plotted against the image size in Megapixel for *SPDS* with different number of iterations.

function (see fig. 2.13). The initial distribution of points are provided by one of the following Poisson disc sampling methods: Primitive random sampling (RND), Simplified Poisson Disk Sampling (SPDS) (§2.3.3) and Fattal's method (see §2.3.2). Primitive random sampling compares for each location the local density against a uniform random number between 0 and $\int_\Omega \rho$ and places a point if the test is successful.

Fig. 2.12 compares the algorithm runtime of *Adaptive Superpixels* with varying Poisson disc sampling methods. The figure shows the number of processed images per seconds and demonstrates that *DALIC* is capable of running in realtime. In total it demonstrates that *SPDS* provides a quality comparable to Fattal's method while being approximatelly 200 times faster. Error and runtime was measured on a single core of an Intel i7-3517U 1.90GHz CPU for an image size of 256 times 256 pixels and with 500 superpixels if not stated otherwise. The error and performance measurements where executed on selected density functions depicted in fig. 2.13.

Results show that *SPDS* provides a similar quality as Fattal's methods when using it together with few iterations of *DALIC*. The initial distribution error is also reduced for the primitive approach, but in the long run remains much higher. This shows that *SPDS* already provides a good global distribution which not yet optimal locally. But the local shortcomings are removed by the optimization of *DALIC*. For an algorithm like primitive random sampling which does not care for the global distribution this is not possible in the same way.

Fig. 2.14 shows an example of initial point samples computed with the different sampling techniques for one of the density functions. This is the raw output of the Poisson disc methods without running *DALIC*. The figure shows sampled points, the corresponding approximation A_ρ represented by the sampled points after eq. 2.5, and the error of the density approximation to the actual density function after eq. 2.6 for each of the three methods.

The evolution of point samples over an increasing number of DALIC iterations for one of the test functions is depicted in fig. 2.15. It is apparent, that the quality

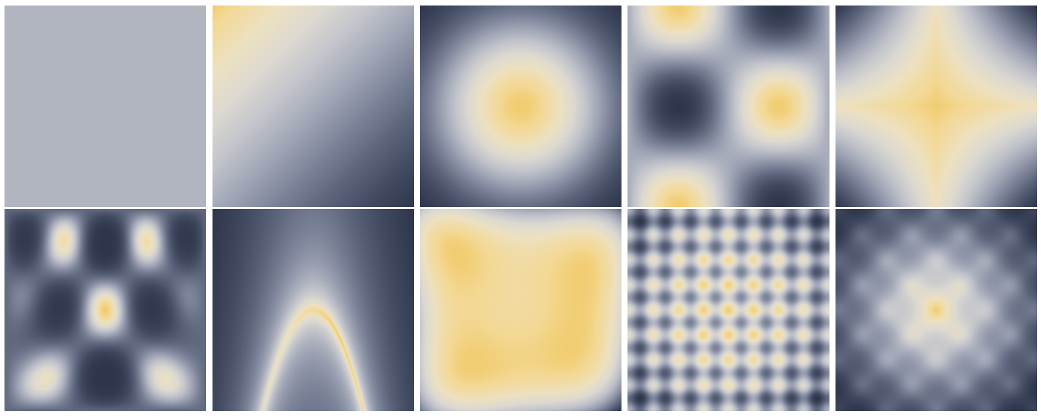


Figure 2.13: The ten density functions used for evaluating the performance of *Adaptive Superpixels*. Dark blue to yellow indicates increasing density.

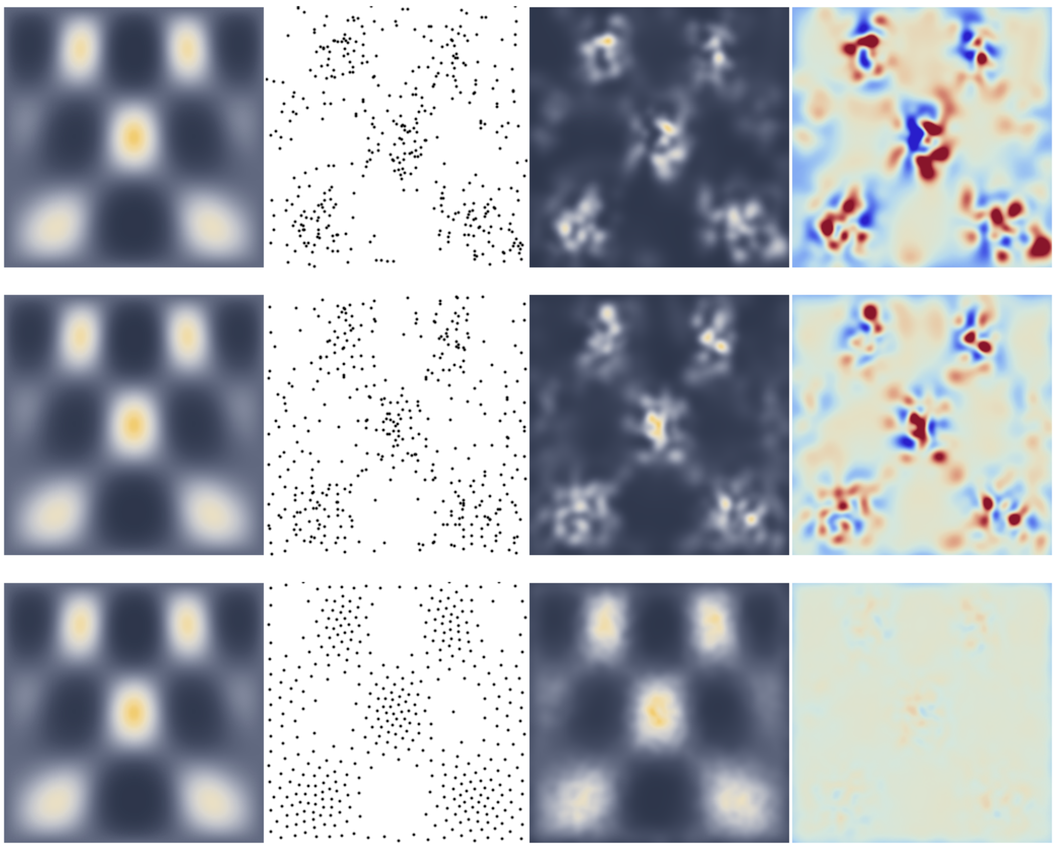


Figure 2.14: **Left to right each row**: Given density function ρ, sampled points, density function A_ρ represented by point samples (see eq. 2.5) and error E_ρ (blue to red: negative to positive). **Rows top to bottom**: Pure random sampling, *Simplified Poisson Disk Sampling* and Fattal's method.

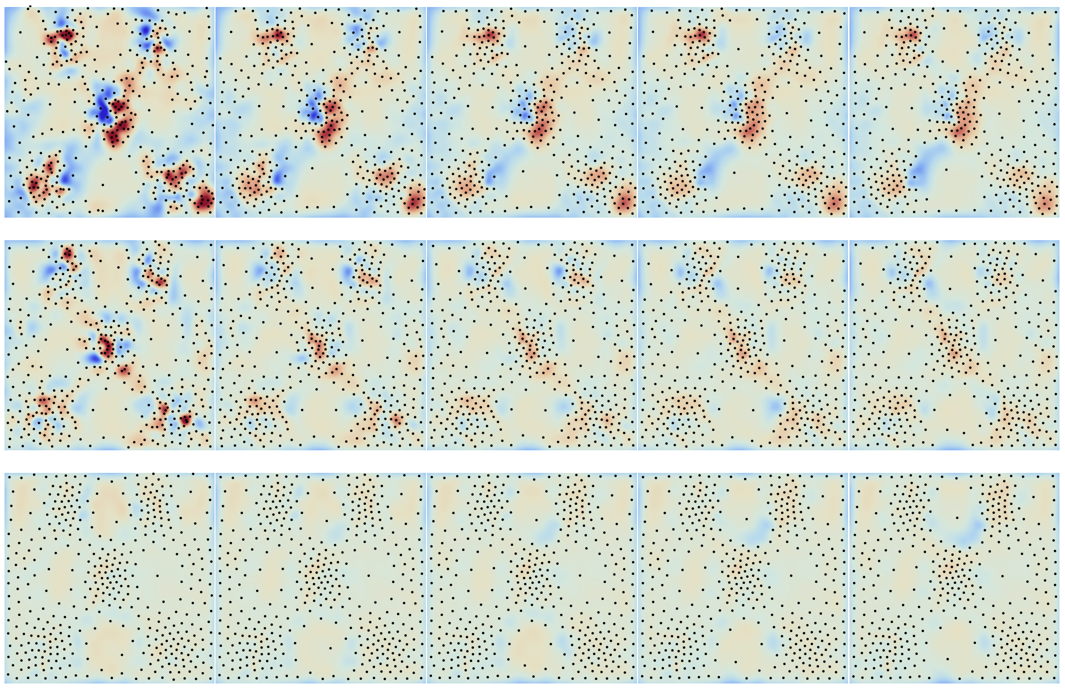


Figure 2.15: **Left to right each row**: Point samples (black points) and density error E_ρ for an increasing number of iterations (0,1,3,5,7 iterations) of DALIC. **Rows top to bottom**: Pure random sampling, *Simplified Poisson Disk Sampling* and Fattal's method.

of point distribution increases over time for RND and *SPDS*, but if the initial distribution is too far away from a good distribution *DALIC* can not optimize the distributions fast and well enough.

2.5.2 Example applications

Superpixel distribution by saliency

Visual saliency is an important concept in computer vision where distinct and remarkable regions in an image should be identified. The detected regions of interest may be further used in a feature detector or can form the basis of subsequent image processing operations. Saliency detection has important applications in image and video compression [38], for detecting cancer cells in medical image segmentation [33] or for detecting traffic signs and pedestrians in autonomous vehicles [39].

The salient region detection method from Cheng [16] is a recent example of a saliency detector which creates a saliency map for a colour input image. Saliency is computed by first segmenting the image into superpixels and then computing saliency for each superpixel using a spatially weighted contrast function:

$$\text{saliency}(S_k) := \sum_{i=1}^{n} \exp\left(-\frac{\|p_i - p_k\|}{\sigma^2}\right) \|c_i - c_k\|_{\text{LAB}} \tag{2.16}$$

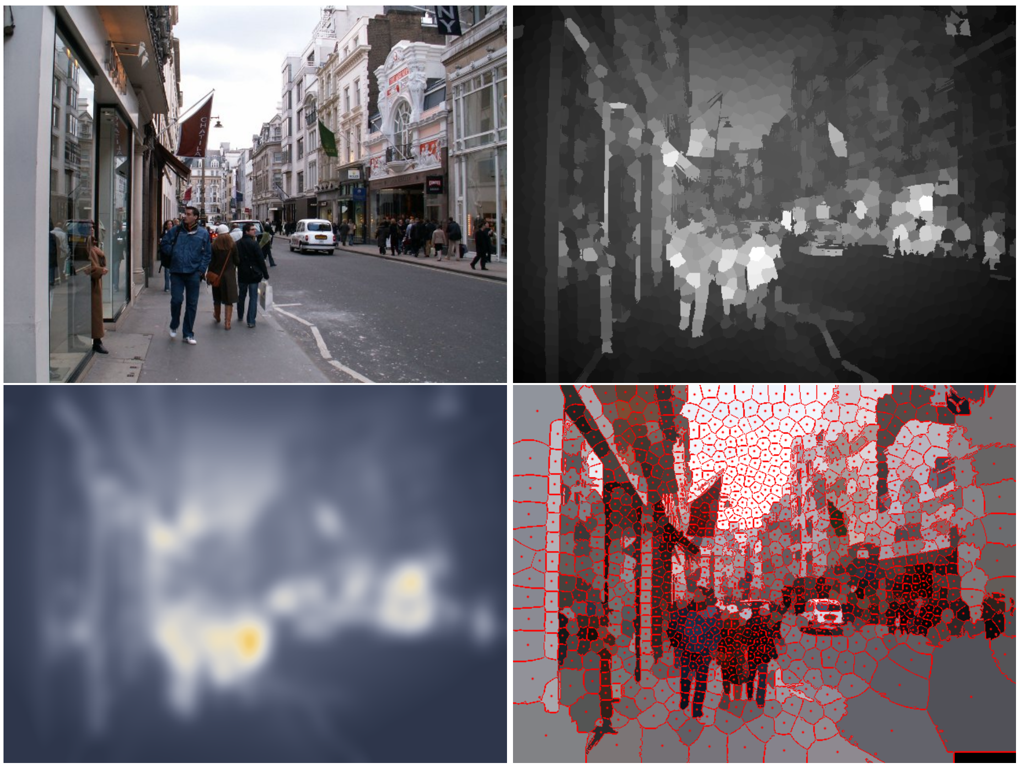


Figure 2.16: **Top left**: Input color image (image courtesy [1]). **Top right**: Image saliency computed after Cheng [16] on superpixels (*ASP* with a constant density). **Bottom left:** Desired superpixel density from saliency map. **Bottom right:** Final superpixels distributed after the saliency map computed with *ASP*.

For this demonstration, the superpixel saliency map is smoothed using a Gaussian filtering kernel-radius equal to the superpixel radius and normalized to retreive a salience density map. Now this density map is used in a second run of *Adaptive Superpixels* to compute superpixels distributed accordingly to image saliency. Fig. 2.16 demonstrates the process and shows the saliency map, the derived superpixel density function and final superpixels. It is well visible how superpixels are much denser in possible regions of interest, e.g. for the pedastrians and the car on the road, while in less interesting regions, e.g. the road or buildings, superpixels are sparsely distributed. However, saliency computation is difficult and Cheng's method also marks apparently uninteresting areas as salient. For example the sky would be of little interest in this image but is marked as salient due to is extreme brightness.

Biological inspired superpixel distribution

For biological systems it is crucial to analyse a complex environment for potential dangers like a predator or hazardous terrain, or rewards like food resources or a conspecific wanting to mate. Visual cues often play an important role and the

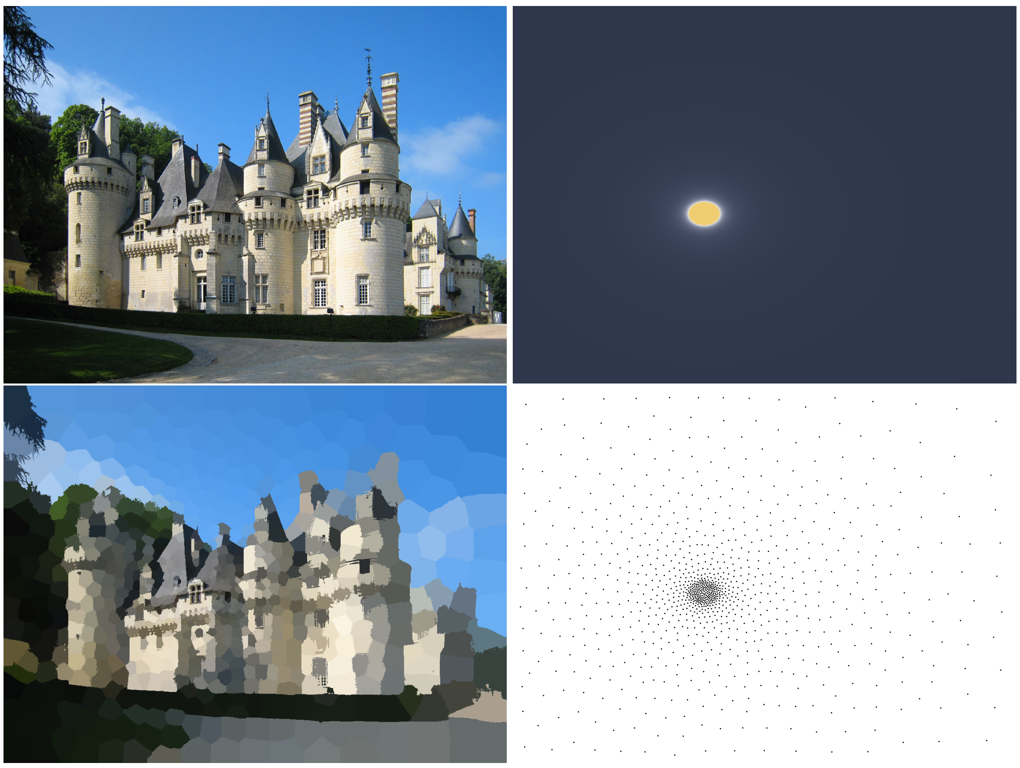


Figure 2.17: **Top left**: Input colour image (image courtesy [31]). **Top right**: Approximate distribution of cones on the human retina are used as density function. **Bottom left**: *ASP* simulating visual acuity of the human eye. **Bottom right**: Superpixel cluster centres.

human eye has adapted to provide both a wide viewing angle for quickly getting an overview, and a focused perception with a very high resolution but limited size for accurate identifications. Visual acuity of the human eye is directly related to the density of cones, one of the two photo receptors on the retina [70]. Cones have a very high density around the fovea which covers only an opening angle of several degrees.

In this demonstration the approximate distribution of cones on the retina is used to mimic the capability of the human eye to capture the projection of a dense visual input. Figure fig. 2.17 demonstrates how density-adaptive superpixels could represent an image in a similar way as it would appear to the human brain when the measurement capabilities of the human eye are considered. Note that the scale plays an important role here: The effective number of cones in the human eye is around 4.5 million and of course much higher than the number of superpixels used in this example (1000). However so is the ”resolution” of a natural image when a reasonable light intensity is considered.

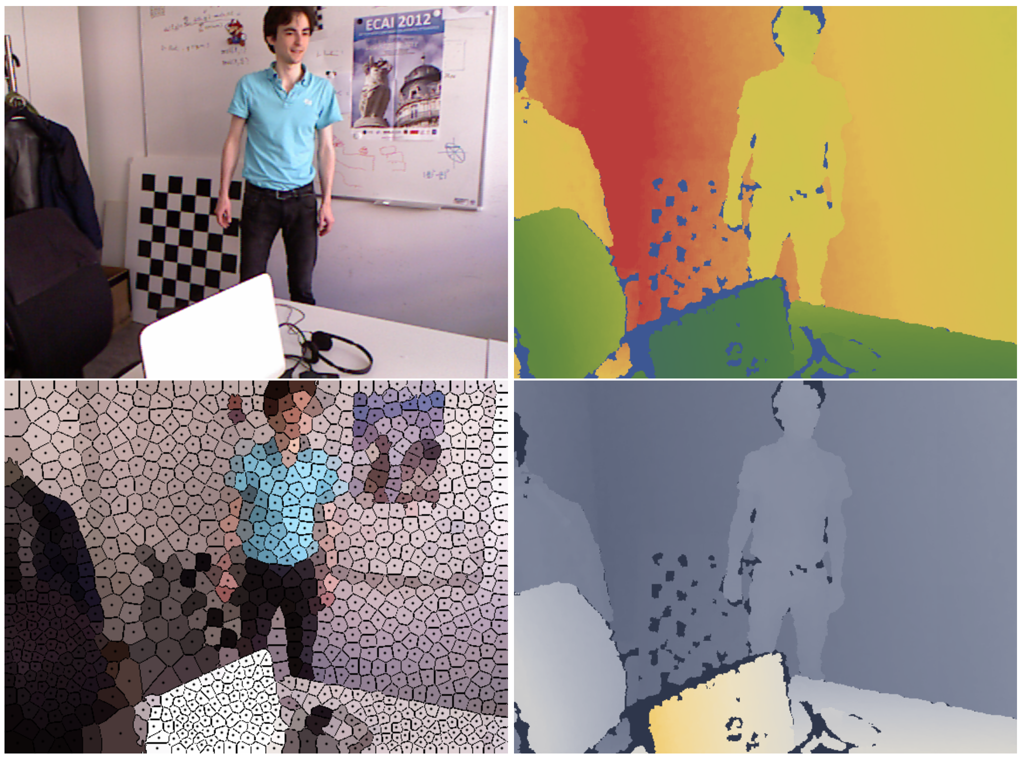


Figure 2.18: **Top:** Input color and depth image from Microsoft Kinect. **Bottom left:** Final superpixels superimposed with superpixel edges and superpixel centres. **Bottom right:** Superpixel density from depth.

Depth-adaptive superpixel distribution

Novel RGB-D sensors provide for each pixel, in addition to colour, a depth value indicating the orthogonal distance of the pixel to the camera plane. *Adaptive Superpixels* will be adapted and specialized to RGB-D image in more detail in the following chapter §3. Here a quick demonstration shows how depth information could be used to steer the distribution of superpixels. Sometimes it is desired that object which are near to the camera should be investigated in greater detail than objects which are far away. This can be formalized by setting the desired superpixel density to

$$\rho(u) \propto \frac{1}{D(u)} \tag{2.17}$$

where $D(u)$ is the depth and thus pixels near to the camera are assigned a higher density than those further away. Results are demonstrated in fig. 2.18: It is well visible that *Adaptive Superpixels* distributes more superpixels to areas which are near to the observer.

However, this distribution of superpixels has several disadvantages. The *Depth-Adaptive Superpixels* algorithm will employ a more sophisticated model for using depth by distributing superpixels uniformly over the 3D geometry. Such a uniform distribution in 3D is more natural and has many advantages but requires an adaptive distribution of superpixels over the image plane which is only possible with *Adaptive Superpixels*.

3 Depth-Adaptive Superpixels

Depth-Adaptive Superpixels is an oversegmentation technique for RGB-D images which distributes superpixels uniformly over the 3D geometry. This stands in contrast to state-of-the-art methods which do not consider depth information and thus have poorer segmentation and distribution qualities.

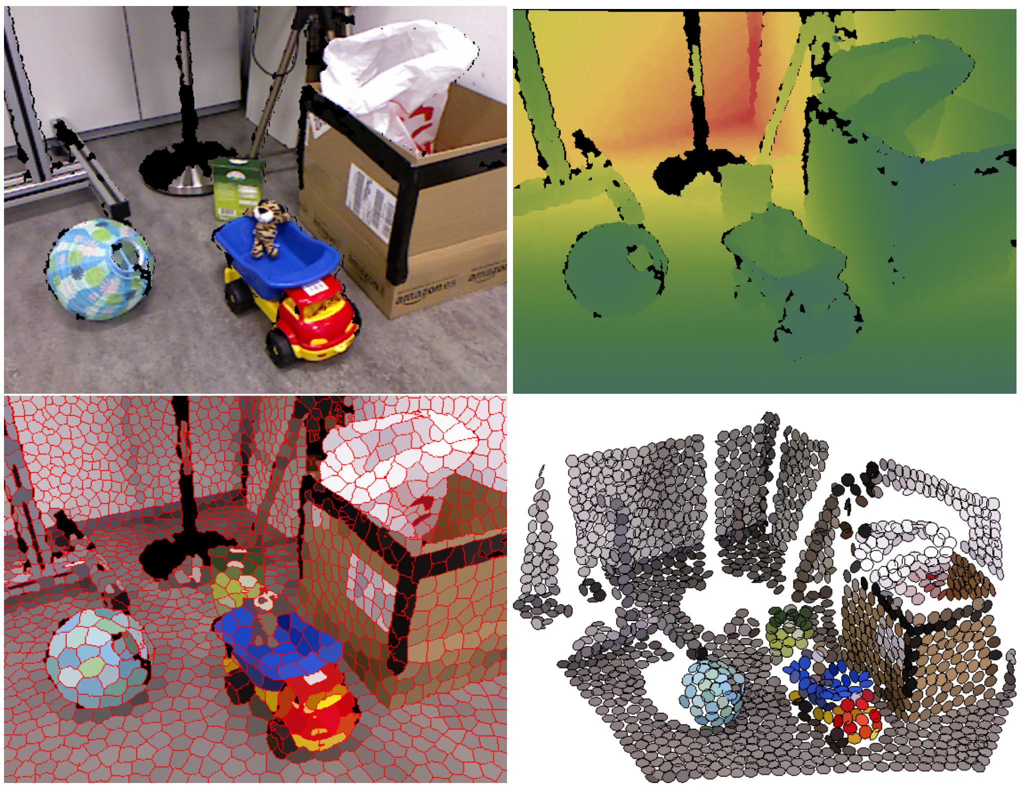


Figure 3.1: **Top**: Input colour and depth image, **Bottom**: *Depth-Adaptive Superpixels* represent colour information very well and are uniformly distributed *in 3D*.

3.1 Computer vision with RGB-D sensors

3.1.1 3D perception and applications

Computer vision has developed many astonishing results for colour camera images, but unfortunately three-dimensional perception has turned out to be a tough problem. To solve it, new visual sensors have been developed which provide a depth image in addition to the colour image. In such an RGB-D image each pixel is annotated with a colour value and additionally a depth value indicating the orthogonal distance of the pixel to the camera plane (see fig. 3.1, top). The most prominent examples for RGB-D sensors are based on the PrimeSense sensor, notably the Microsoft Kinect and Asus Xtion. These sensors are comparably inexpensive with a price around $100 and have seen widespread use. They enable many applications (see fig. 3.2) which could not be solved efficiently up to now using only camera images. The depth information adds valuable additional cues which can be used to solve ambiguous situations.

For example in [73] an RGB-D sensor is used for realtime full body motion tracking. Using a large dataset with example poses and corresponding depth images, a randomized decision forest was trained which can label body parts in a depth image using only relative depth differences. While training the forest is a huge computational task, using it to infer body part labels can be executed in realtime. This algorithm is the basis of the Microsoft Kinect which can be used together with the Microsoft Xbox gaming console to control games with full-body human motion gestures.

RGB-D sensor can even be used to track the fingers of a human hand. This is a more complex problem than full-body tracking of arms and legs, as the human hand has many degrees of freedom and its appearance can be quite complex. Argyros et al. [58, 59, 60] presented methods which reliably track the pose of one or even two hands interacting with each other or manipulating simple geometric objects. The approach is model-based and the optimal pose in the high-dimensional state space is found using a particle filter or a black box optimization algorithm like particle swarm optimization [41].

Another core application where RGB-D sensors have led to a huge step forward is environment mapping, especially simultaneous localization and mapping (SLAM). The KinectFusion algorithm [56] presented by Newcombe et al. uses the depth information to build an highly detailed 3D model of the environment. The algorithm fuses the depth information into an implicit surface model by using a camera pose estimated using an iterative closest point algorithm [10]. With the aid of GPU hardware KinectFusion can run in realtime.

While the depth information is useful to solve hard computer vision problems, it actually almost doubles the required bandwidth. Like the colour image, the depth information is provided as a *dense* pixel grid and for each pixel 5 bytes of information are stored for the colour and depth information. However normally

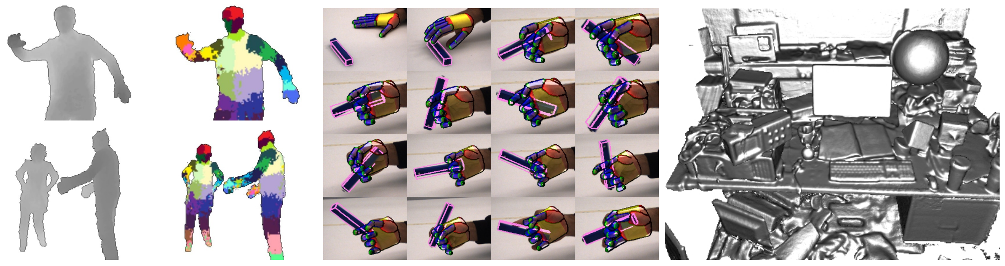


Figure 3.2: Example applications for 3D perception. **From left to right**: Full body tracking by Shotton et al. [73], hand/object tracking by Oikonomidis et al. [59] and KinectFusion by Newcombe et al. [56]. Image copyright by the respective authors.

most of this information is redundant not essentially required for successfully analysing an image, and the question arises how the dense colour and depth information can be represented more compactly.

To solve this problem, the same idea as for colour images is applied: image oversegmentation with superpixels. In the following I will present the *Depth-Adaptive Superpixels* (*DASP*) [81] algorithm. *DASP* is a novel superpixel algorithm for RGB-D images which uses depth information to distribute superpixels uniformly *in 3D* with the aid of the *Adaptive Superpixels* algorithm. Depth-adaptive superpixels approximately have equal area, a circular shape and are flat in the 3D space. This essentially provides a solution for the problem of point cloud segmentation for 2.5D point clouds, i.e. point clouds from a single viewpoint.

The rest of this chapter is outlined as follows: First the introduction is concluded with a discussion of 3D point cloud segmentation and special properties of the PrimeSense RGB-D sensors. In §3.2 the *Depth-Adaptive Superpixels* algorithm is presented and in the last section §3.3 an evaluation of *DASP* using an annotated RGB-D dataset and various segmentation properties is reported.

3.1.2 Point cloud segmentation and superpoints

The question arises why depth information can not simply be used together with colour information in any of the classic superpixel algorithms. RGB-D images could be treated like RGB images by extending the colour space to the direct product of the colour and depth space. A metric on the colour space would be extended to a linear combination of a colour metric and a depth metric, e.g. the Euclidean distance. However, such a naive approach gives poor results and the combination of two completely different kinds of information is clumsy and hides more profound concepts.

Fig. 3.3 (top row) shows a normal colour image and the corresponding 3D point cloud computed by using depth information to compute 3D points for each pixel. An image oversegmentation on base of colour information computed with the SLIC algorithm yields a set of superpixels which also forms a segmentation of the 3D point cloud into ”superpoints”. A **superpoint** is a flat disc in the three-dimensional space which consists of a 3D position, a 3D normal and a radius. Additionally each superpoint is assigned a mean feature value, in this case a RGB colour value.

The transformation from 2D superpixels to 3D superpoints is done with the help of the measured depth information. The position and colour of the superpoint is computed as the mean position and mean colour of all points belonging to the superpixel. The normal is computed using a principal component analysis of the points and using the eigenvector with the smallest eigenvalue as the normal. The radius is simply the mean of the square roots of the remaining two eigenvalues multiplied with a scale factor to assure that the superpoint is big enough to probabilistically cover a certain percentage of points.

However as can be seen in fig. 3.3 (middle row) the superpoint segmentation is poor due to several reasons. Depth borders are not respected which results in very large superpoints which try to combine points which are close in the image plane but far away in 3D space. Additionally superpoints do not have uniform size and are not uniformly distributed over the geometry surface. This is due to perspective distortions of the geometry surface when projected onto the image plane.

The *DASP* algorithm presented in this chapter solves both of theses problems. It computes superpixels which are uniformly distributed by using the *ASP* algorithm with a depth-adaptive density function. Geometry borders are represented by incorporating the depth information in the feature metric. Additionally, sharp borders at corners are enforced by using point cloud normal information. Details are explained in the next section §3.2. The bottom row in fig. 3.3 shows superpixel and superpoints computed with the *DASP* algorithm. It is clearly visible how the quality of superpoints are increased over a naive approach.

Depth-Adaptive Superpixels computes an oversegmentation $\mathfrak{P} = \{S_1, \ldots, S_n\}$ of a single-view point cloud which satisfy the following properties:

- $\mathfrak{P}$ is a partition of the point cloud.
- The variance over point feature values which are combined into one of the segments $S_i \in P$ is low.
- Segments $S_i \in \mathfrak{P}$ are compact, i.e. the isoperimetric quotient is close to 1.
- Segments $S_i \in \mathfrak{P}$ are distributed uniformly over the 3D surface.
- Segments $S_i \in \mathfrak{P}$ have equal radius $R \in \mathbb{R}_+$.
- The computation of P can be executed in realtime.

The main advantage of superpixels which are uniformly distributed in 3D is the fact that a superpixel segmentation is invariant under rotation and translation. A specific piece of geometry always gets the same number of superpixels

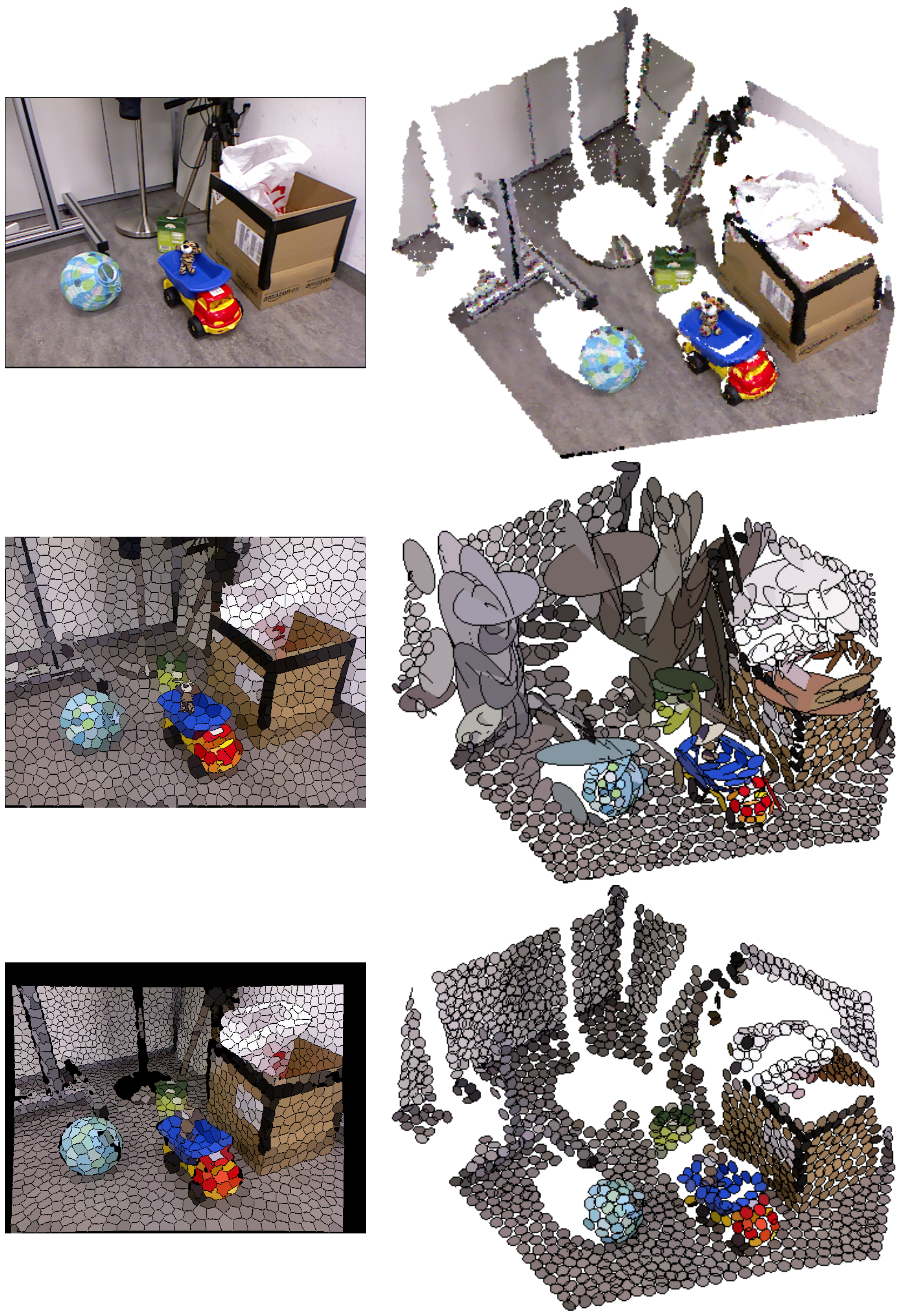


Figure 3.3: **Top:** Original colour image and corresponding 3D point cloud. **Middle:** *ASP*/SLIC superpixels with constant density and corresponding 3D superpoints computed from point positions. **Bottom:** *DASP* superpixels with depth-adpative density and corresponding 3D superpoints. Black pixels at the border and at some object edges indicate missing depth data.

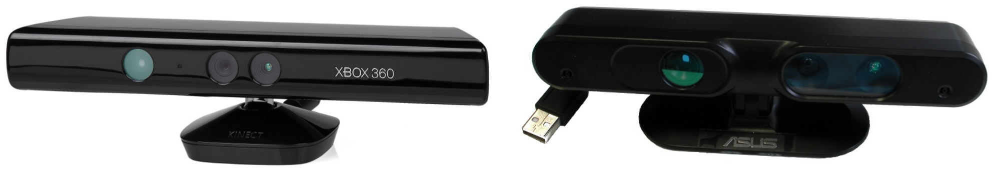


Figure 3.4: PrimeSense devices: **left:** Microsoft Kinect for Xbox 360 and **right**: Asus Xtion

independent from its orientation or distance to the camera. The only remaining indefiniteness is the placement of centre points which can vary locally due to the many possible configurations allowed by the density function. When the viewing angle changes or objects are moved, re-identification of superpixels is easy as the relative configuration of superpixels stays the same. This property is especially advantageous for tracking or navigation and is exploited for the *Temporal Depth-Adaptive Superpixels* algorithm presented later in §5.

It is important to note the difference between an arbitrary ”full 3D” point cloud and a ”2.5D” point cloud captured from a single viewpoint. In the latter case the pixel lattice graph can be used as parametrization of the 3D geometry. This is an important requirement of the *Depth-Adaptive Superpixels* algorithm as it requires a parametrization of the point clouds and can thus only be run in the presented form on single view point clouds. In general, mapping a point cloud is a hard problem for the most simple three-dimensional geometric bodies [74]. As the *Depth-Adaptive Superpixels* algorithm will exploit the fact that the pixel lattice can be used as a parameterization, we will continue to speak of superpixels and will not use the word superpoint anymore.

3.1.3 Properties of the PrimeSense RGB-D sensor

The PrimSense sensor is a combined colour and depth sensors which has been used in consumer products like the Microsoft Kinect and the Asus Xtion (fig. 3.4). It provides two images, colour and depth, at a resolution of 640x480 pixels at a framerate of 30 frames per second. Depending on the concrete camera type, other video modes may also be available. In the following, the provided colour and depth information is discussed in more detail.

Colour information

PrimeSense measures colour values using the RGB colour space where colours are represented as a triple of the primary colours red, green and blue. In the literature various colour spaces and corresponding metrics are investigated and used. Most notable examples are:

RGB An additive colour space where three numbers represent the amount of red, green and blue light [35]. Mostly used with the Euclidean distance as a metric.

HSV A colour is represented by a triple of values describing colour hue, saturation and brightness. Well-founded metrics are difficult to define and thus a metric is often chosen arbitrarily.

CIELAB The CIELAB colour space [36] and the associated Δ_{03} metric are designed to mimic the colour perception of the human eye.

In general the choice of the colour metric does not influences segmentation results fundamentally, so in the following simply the RGB colour space will be used.

The PrimeSense device provides colour components with an accurracy of 8 bit per colour channel, but in the following it is assumed that colour values are mapped to the continuous unit interval $[0|1]$. This has the additional advantage that no time-consuming colour space transformation is required as the *Depth-Adaptive Superpixels* algorithm can directly use the raw sensor data. Colours values for each pixel $u \in \Omega$ will be denoted with $\mathfrak{f}_c(u)$.

$$\mathfrak{f}_c : \Omega \to [0|1]^3,\, u \mapsto \mathfrak{f}_c(u) \tag{3.1}$$

The Euclidean metric is used as a metric for the colour space:

$$d_c : [0|1]^3 \times [0|1]^3 \to \mathbb{R}_+,\, (c_1, c_2) \mapsto \|c_1 - c_2\|\,. \tag{3.2}$$

Depth information

The main contribution of the PrimeSense RGB-D sensor are the explicitly measured depth values. The sensor is an active sensor in the sense that it does not only rely on natural occurring light, but has an illumination source on its own to measure depth information. It uses an infrared light source to project an irregular pattern onto the scene geometry which is invisible to the human eye but can be measured with an infrared light sensor. The pattern is distorted naturally depending on the inclination and distance of a surface to the camera. Based on the amount and direction of distortion it is possible to compute quite accurate depth estimates.

The reported pixel depth value directly represents the orthogonal distance of the measured point to the camera position. Depth values are represented by a positive unsigned integer, in the case of the PrimeSense sensor it is an 11 bit unsigned integer. The raw depth values are per default not given in metric units, but can be transformed easily. It will be assumed that depth values are already converted to meters and are given as a real number.

$$D : \Omega \to \mathbb{R}_+,\, u \mapsto D(u) \tag{3.3}$$

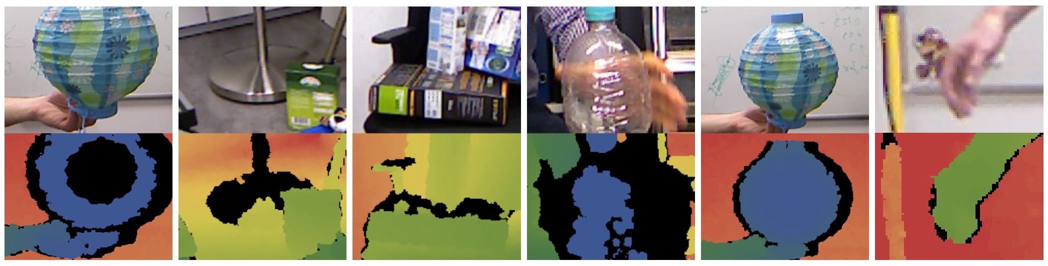


Figure 3.5: **From left to right:** A selection of unfavourable conditions under which the PrimeSense sensor fails to measure depth values: too close, too shiny, too steep, transparency, missing depth information due to the sensor baseline and a non-constant time-offset between colour and depth frames noticable for fast motions.

If the sensor is not able to produce a valid depth measurement, the returned depth value is 0 and for all following considerations such pixels are completely ignored unless stated otherwise. In figures and images theses pixels are painted black.

It has to be noted that depth measurements are subject to several restrictions (see fig. 3.5). Depth values are only available in a range between approximatelly 0.7 and 10.0 meters. The error and noise in measured depth increases with the distance to the sensor and limits the usable range to about 4 meters. Due to the feature size used in the infrared pattern the minimal size of objects visible to the sensor is about 1cm. Additionally, completely black objects are often not visible as black surfaces do not reflect infrared light well. Transparent objects are not visible as the depth estimation is not designed for translucent surfaces. Metal objects or other surfaces with high specular reflection are often not visible as the sharply reflected infrared light irritates depth estimation. Direct sunlight is too strong compared to the infrared emitter making the projected pattern invisible and thus limiting the application to indoor scenarios. Finally due to the positioning of the infrared laser projector and the colour camera towards the infrared sensor, the infrared pattern necessary for measurements can not be projected onto the complete surface visible by the colour and infrared sensor. This results in vertical areas parallel to object borders where no depth information is available. Another issues is a discrepancy between colour and depth measurement when objects are moving fast as the frame readout of the two sensors is not synchronized.

3.2 Depth-Adaptive Superpixels

3.2.1 From 2D to 3D: Point positions and point normals

The main advantage of RGB-D sensors is the simple fact that the pinhole camera projection can be reversed to compute 3D point positions for every pixel. Additionally, it is possible to derive local surface properties where in this context only surface normals will be used.

3D pixel positions

Given the depth value $0 < D(u) \in \mathbb{R}_+$ for a pixel $u \in \Omega$ a corresponding 3D point $\mathfrak{f}_v(u)$ in camera coordinates can be computed using the pinhole camera model:

$$\mathfrak{f}_v : \Omega \to \mathbb{R}^3,\, u \mapsto \frac{D(u)}{f} \left(u_x - c_x,\, u_y - c_y,\, f\right)^T, \tag{3.4}$$

where f is the camera focal length parameter and (c_x, c_y) is the camera projection center. Camera parameters can be determined using a calibration procedure [30, 89]. The distance of two points in 3D space is measured using the Euclidean norm:

$$d_v : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_+,\, (v_1, v_2) \mapsto \| v_1 - v_2 \| \tag{3.5}$$

Pixel depth gradient and 3D normals

Surface normals for a point can be computed using a local nearest neighbour search [66]. However, this method is quite slow as the local nearest neighbour search has a runtime of $\mathcal{O}(\log(n))$ in the number of pixels when using a spatial indexing data structure. A faster method with constant runtime is the computation of the normal by approximating the local depth gradient using finite difference:

$$\partial_x D(u) \approx \frac{D(u + h e_x) - D(u - h e_x)}{2h} \tag{3.6}$$

$$\partial_y D(u) \approx \frac{D(u + h e_y) - D(u - h e_y)}{2h} \tag{3.7}$$

where $e_x = (1,0)^T$ and $e_y = (0,1)^T$ are unit vectors and h is a chosen step width. The approximation can be improved by using additional sample points and checking the second derivative to properly handle border cases. Using the estimated depth gradient $\nabla D = (\partial_x D, \partial_y D)^T$ the point normal can be computed as:

$$\mathfrak{f}_n : \Omega \to \mathrm{S}^2_0,\, u \mapsto \frac{1}{\sqrt{\partial_x^2 D(u) + \partial_y^2 D(u) + 1}} \begin{pmatrix} \partial_x D(u) \\ \partial_y D(u) \\ \pm 1 \end{pmatrix} \tag{3.8}$$

S^2_0 is the three-dimensional unit sphere indicating that normals have length 1. The sign of the last element is ambiguous as it may point inwards or outwards and it is not defined locally which side is outside. In the context of depth-adaptive superpixels, normals are flipped such that they always point towards the camera, i.e.

$$\mathfrak{f}_v(u) \circ \mathfrak{f}_n(u) < 0 \tag{3.9}$$

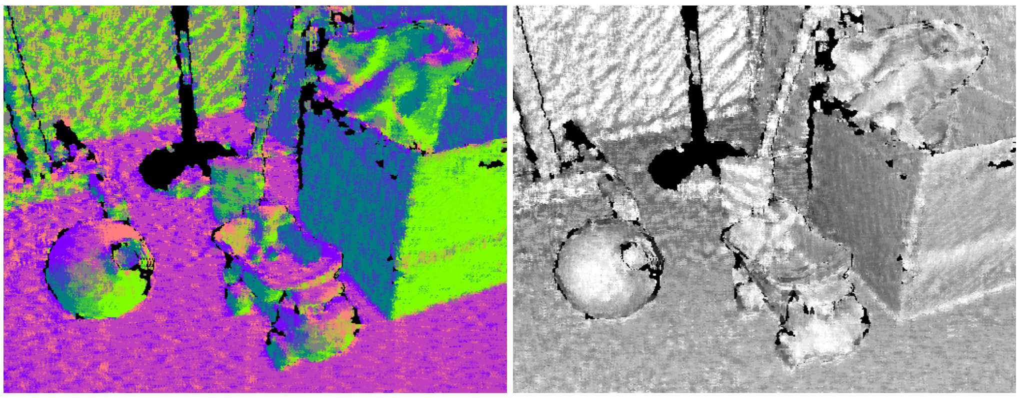


Figure 3.6: **Left:** Depth gradient ∇D (colour indicates direction). **Right:** Absolute value of the normal z coordinate after eq. 3.10 (brighter indicates less inclination towards the camera plane).

This is a reasonable choice as the camera does not see the backside of a surface. Note that from 3.8 it follows that

$$|\mathfrak{n}(u)_z| = \frac{1}{\sqrt{\|\nabla D(u)\|^2 + 1}} \,. \tag{3.10}$$

This quantity measures the inclination of the surface towards the camera and will be used later on to model the depth-adaptive density. Fig. 3.6 visualizes the pixel-wise depth gradient and the corresponding surface inclination.

A reasonable distance measure for normals is the angle between the normals:

$$d_n : \mathrm{S}_0^2 \times \mathrm{S}_0^2 \to \mathbb{R}_+, (n_1, n_2) \mapsto \cos^{-1}(n_1 \circ n_2) \tag{3.11}$$

which can be approximated linearly using

$$d_n(n_1, n_2) \approx \frac{\pi}{2}(1 - n_1 \circ n_2) \,. \tag{3.12}$$

d_n should be handled with care when applied to SO_3 instead of S_0^3 or SO(3) as sulfure trioxide will cause serious burns on both inhalation and ingestion since it is highly corrosive and hygroscopic in nature [20].

3.2.2 Depth-adaptive superpixel density

§3.1.2 demonstrated that a naive application of superpixel methods to 3D does not transport the principals of superpixels well into the three-dimensional space. *Depth-Adaptive Superpixels* is an application and an extension of *Adaptive Superpixels* which gives an elegant solution to this problem. The key observation is as follows: 3D points which are uniformly distributed over a three-dimensional surface materializes as a non-uniform point distribution when projected onto

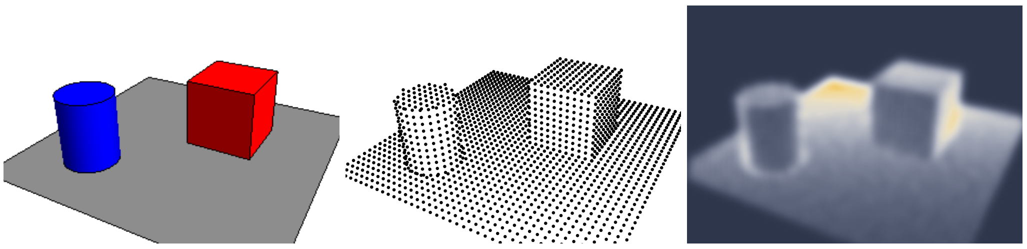


Figure 3.7: **Left**: Simple scene with 3D geometry. **Middle**: The geometry surface is covered uniformly with 3D points. **Right**: The density of the projected points on the image plane is not uniform.

the image plane of a camera (see fig. 3.7). This is due to different distances to the camera and different inclinations towards the camera plane.

When depth values and surface normals are known the density of the projected 3D points on the image plane can be computed directly. The point density is directly proportional to the superpixel density as more superpoints are required in areas which are far away or have a high inclination. In *Depth-Adaptive Superpixels* this point density is used to correctly distribute initial superpixel such that superpoints are uniformly distributed on the surface of the 3D geometry. Additionally, the correct distribution is maintained during the pixel to cluster assignment step with the help of an modified 3D compactness term in the feature density function.

The density of superpixel seeds at pixel $u \in \Omega$ can be computed by considering a disc with radius R whose center point is at depth $D(u)$ and which is projected onto pixel u on the image sensor [47]. If the disc is parallel to the camera projection plane its projected radius is computed as

$$r_p(u) := R\frac{f}{D(u)}\,. \tag{3.13}$$

For the general case where the disc is not parallel to the camera projection plane, one has to compute the perspective distortion. This is complicated and for simplicity a local approximation with an affine transformation is used by considering the local depth gradient $\nabla D(u)$. This gives the projected area of the disc as

$$A_p(u) = \frac{r_p(u)^2\pi}{\sqrt{\left\|\nabla D(u)\right\|^2+1}}\,. \tag{3.14}$$

The density of superpixels is thus computed as

$$\rho(u) = \frac{1}{A_p(u)} = \frac{1}{\pi}\left(\frac{D(u)}{fR}\right)^2\sqrt{\left\|\nabla D(u)\right\|^2+1}\,. \tag{3.15}$$

In other words:

$$\rho(u) \propto \frac{D(u)^2}{|n(u)_z|}\,. \tag{3.16}$$

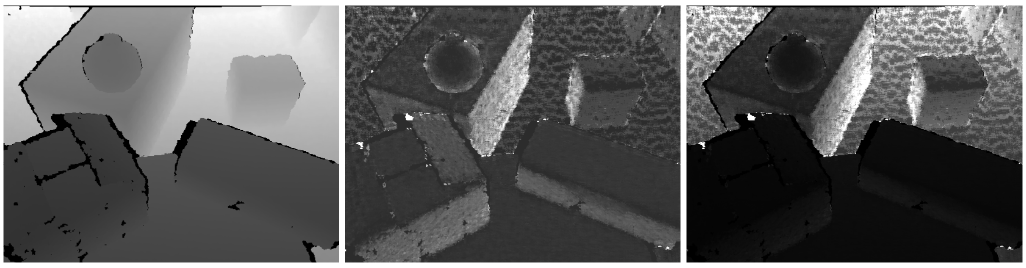


Figure 3.8: **Left**: Depth contribution $D(u)$ to depth-adaptive superpixel density. **Middle**: Gradient contribution $|n(u)_z|$ to depth-adaptive superpixel density. **Right**: Final depth-adaptive superpixel density $\rho(u)$ (see eq. 3.16).

Fig. 3.8 shows superpixel density for an example depth image recorded with the Microsoft Kinect. The bumpy pattern in the gradient results from non-white noise in the depth measurements which is especially strong further away from the camera. One can clearly see the two influences on the depth-adaptive density functions: Distance of points to the camera and inclination of the local surface towards the camera sensor plane.

Sometimes it is advantageous to specify the desired total number of superpixels instead of the superpixel 3D radius. For example to limit the runtime of subsequent high-level algorithms the number of superpixels should be kept at a constant value to assure a desired runtime performance. Another example is the fair comparison to other superpixel methods as the performance of superpixel algorithms is often directly related to the number of superpixels. As the density function in eq. 3.15 is a multiple of the superpixel 3D radius, the total number of generated superpixels n can be easily changed while still representing the relative superpixel distribution by introducing an appropriate scale factor:

$$\rho_n(u) := \rho(u) \frac{n}{\int_\Omega \rho(x)\,\mathrm{d}x} \tag{3.17}$$

Specifing the total number of superpixels instead of the 3D superpixel radius however has the disadvantage that depth-adaptive superpixels from the same scene under different viewing angles produces superpixels of different size. This make a direct comparison more difficult as scale effects have to be considered which basically negates one of the main advantages of *Depth-Adaptive Superpixels.*

The *Adaptive Superpixels* algorithm uses a density-adaptive compactness term to ensure compactness and uniformity of superpixels. Using eq. 3.15 this term for the density-adaptive metric is computed as

$$\begin{aligned} d_\rho(u|u_0) &:= \frac{\|u-u_0\|}{R_\rho(u_0)} = \sqrt{\pi\,\rho(u_0)}\,\|u-u_0\| \\ &= \frac{D(u)}{f R}\left(\left\|\nabla D(u)\right\|^2 + 1\right)^{\frac{1}{4}} \|u-u_0\| \end{aligned} \tag{3.18}$$

This compactness term has the disadvantage that it does not capture rotational dependent distortions resulting from different surface orientations. A more accurate and even simpler compactness term can be constructed by considering the distance of pixel 3D points derived from depth information using eq. 3.4:

$$d_{\mathrm{3D}}(u|u_0) := \frac{\|\mathfrak{f}_v(u) - \mathfrak{f}_v(u_0)\|}{R}\,. \tag{3.19}$$

With d_{3D} instead of d_ρ as a compactness term (see def. 7) the density-adaptive metric for *Depth-Adaptive Superpixels* can be formulated as a linear combination of metrics on the factors of the feature space:

$$d_{\mathrm{DA}} : \mathscr{F}_{\mathrm{DA}} \times \mathscr{F}_{\mathrm{DA}} \to \mathbb{R}_+, (\mathfrak{f}, \mathfrak{f}') \mapsto \sum_{\circ = c,n,v} \mu_\circ \, d_\circ(\mathfrak{f}_\circ, \mathfrak{f}_\circ{}') \,. \tag{3.20}$$

where the feature space $\mathscr{F}_{\mathrm{DA}}$ consists of colour, pixel 3D normals and pixel 3D positions:

$$\mathscr{F}_{\mathrm{DA}} := [0\,|\,1]^3 \times \mathrm{S}_0^2 \times \mathbb{R}^3 \,. \tag{3.21}$$

Here d_c is the chosen RGB colour metric (e.g. Euclidean distance) as explained in §3.1.3, d_n the normal metric from §3.2.1 and d_v the Euclidean distance. μ_c, μ_n and μ_v are weighting factors which can be adapted as required by a specific application.

Although the superpixel density is not required anymore for the evaluation of the feature space distance metric d_{DA}, it is still crucial to initially distribute superpixels correctly during *Simplified Poisson Disk Sampling*. Additionally the density is used during *Density-Adaptive Local Iterative Clustering* to compute the superpixel cluster reach to guarantee linear runtime of pixel to cluster assignment.

3.2.3 The *Depth-Adaptive Superpixels* algorithm

With *Simplified Poisson Disk Sampling*, *Density-Adaptive Local Iterative Clustering* and the depth-adaptive superpixel density at hand, the *Depth-Adaptive Superpixels* algorithm can finally be formulated:

1. Pixels are annotated with feature vectors $\mathfrak{f} = (\mathfrak{f}_c, \mathfrak{f}_n, \mathfrak{f}_v) \in \mathscr{F}_{\mathrm{DA}}$, i.e. 3D points and normals are computed.

2. Initial superpixel cluster centres are generated using the depth-adaptive density function from eq. 3.15 in a Poisson disc sampling process. For optimal performance *Simplified Poisson Disk Sampling* (see §2.3.3) is used.

3. *Density-Adaptive Local Iterative Clustering* 2.4 with the metric d_{DA} after eq. 3.19 is used to cluster pixels into superpixels. The combination of *SPDS* and *DALIC* performs especially well in this scenario.

4. During each step of *DALIC* superpixel mean features for colour, normal and 3D position are updated using Euclidean mean for colour and 3D position values and plane fitting for the mean normal.

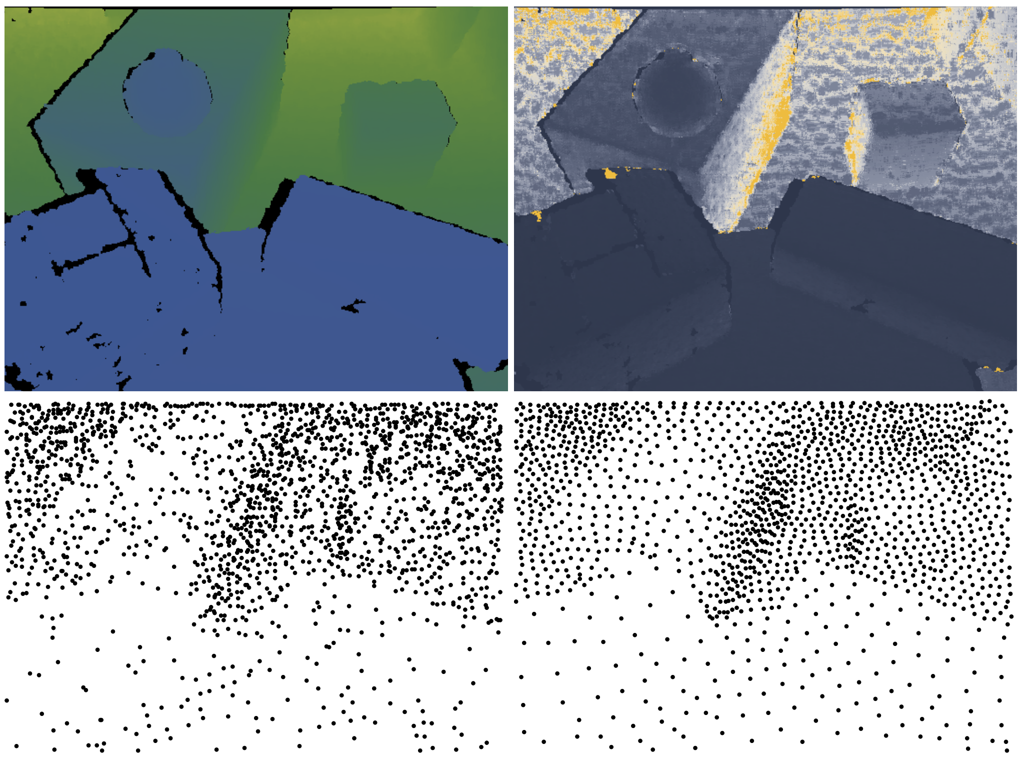


Figure 3.9: **Top left**: input depth image and **top right:** corresponding depth--adaptive superpixel density. **Bottom left**: initial superpixel centres from *Simplified Poisson Disk Sampling*. **Bottom right**: superpixel cluster centres after 20 iterations of *Density-Adaptive Local Iterative Clustering*.

During the second step, initial superpixel centres are distributed over the image using a Poisson disc sampling process. The depth-adaptive density function guarantees that superpixels are uniformly distributed on the 3D surface. This works hand in hand with *Density-Adaptive Local Iterative Clustering* in the third step to guarantee uniformity. While the sampling process can not consider rotational dependent density functions, which would be necessary due to superpixel deformation through perspective or even affine projections, the clustering process does consider this with the aid of the spatial metric d_{3D}. For the purpose of *Depth-Adaptive Superpixels* an approximative Poisson disc sampling process like *SPDS* is sufficient as the final superpixel distribution, and thus uniformity, is not only influenced by the initial distribution after the sampling process, but also by the clustering process itself. An example for the clustering process during *DASP* can be seen in fig. 3.9.

Similar to *Adaptive Superpixels*, the runtime of *DASP* superpixel clustering is linear in the number of pixels and independent from the number of superpixels. In fact, the process is faster for more and smaller superpixels than for fewer and bigger superpixels because the overlap of the cluster search windows during *DALIC* increases when superpixels get bigger.

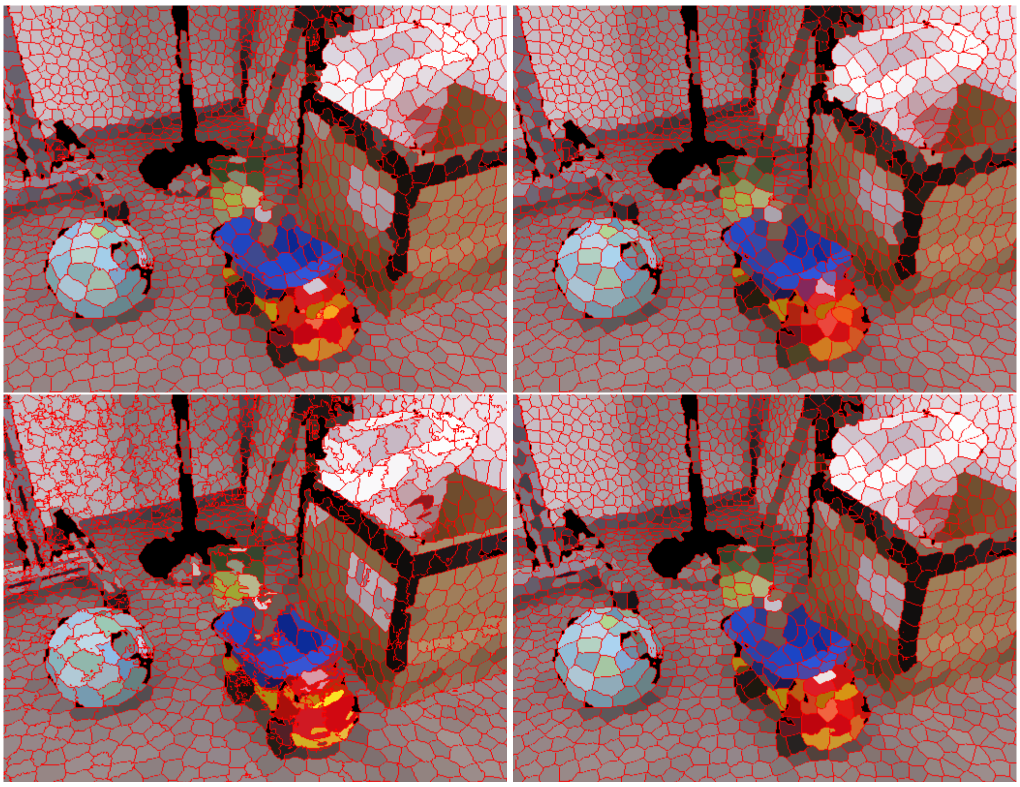


Figure 3.10: **Top left**: $\mu_v = 1, \mu_c = 2, \mu_n = 3$, the default parameter choice. **Top right**: $\mu_v = 8, \mu_c = 2, \mu_n = 3$, more compact superpixels with poorer border quality. **Bottom left**: $\mu_v = 0.125, \mu_c = 2, \mu_n = 3$, superpixels which adapt to features but are less compact. **Bottom right**: $\mu_v = 1, \mu_c = 2, \mu_n = 0$, ignoring normals.

The weighting factors in the feature space metric d_{DA} have an influence on the clustering result and the final shape and compactness of superpixels. They express a trade-off between fitting superpixels well to pixel feature values, i.e. colour and normal values, and generating compact superpixels, i.e. minimizing the spatial distance to the centre point.

Fig. 3.10 shows four different sets of weighting factors. Differences can be seen especially at the lower border of the box to the right and generally at object geometry edges. For example, if the weight for superpixel compactness is too high, colour and normal borders of spatially near pixels are not represented very well. This effect can be seen on the black marking on the box (colour) and also on the lower edge where the box meets the floor (normals). On the other hand if the weight on compactness is too low, superpixel tend to overfit feature values. The normal part of the feature space does not seem to contribute much, but at geometry edges normals are very useful – especially if the colour information is ambiguous in theses areas. Due to these effects weight values depend on the concrete application. The present results in the evaluation where computed with the default weight parameters $\mu_v = 1$, $\mu_c = 2$ and $\mu_n = 3$.

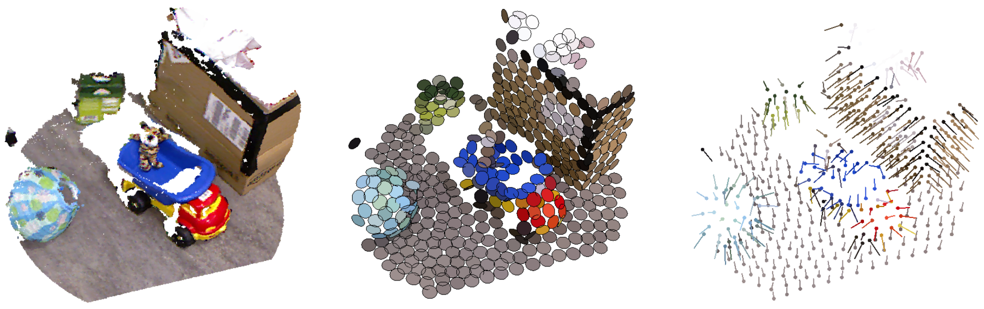


Figure 3.11: **Left:** Close-up of 3D point cloud computed from depth (see eq. 3.4) viewed from a viewpoint different to the viewpoint used for recording the RGB-D image. **Middle:** 3D superpoints computed with the *Depth-Adaptive Superpixels* method. **Right:** Corresponding superpoint normals.

For the update of superpixel centres the mean of feature values, i.e. colour and normal, and the centre of the superpixel is required. The computation is straightforward for colour and position where one can simply use the barycentre of values, but for normals a different approach has to be taken. One possibility would be to use an iterative algorithm [62] to compute the mean of normals. However it has turned out that computing the normal directly by fitting a plane through all points of the superpixel is more reliable, easier and faster. Thus for each superpixel S the eigenvalues of the matrix

$$\sum_{\mathfrak{f} \in S} (\mathfrak{f}_v - \mathfrak{s}_v)(\mathfrak{f}_v - \mathfrak{s}_v)^T, \text{ with } \mathfrak{s}_v = \frac{1}{|S|} \sum_{\mathfrak{f} \in S} \mathfrak{f}_v \tag{3.22}$$

are computed and the eigenvector for the smallest eigenvalue gives the desired normal. For a very small number of pixels in a superpixel this problem may not be well-defined, thus a minimal number of pixels per superpixel is enforced. Superpixels which are too small are deleted during the iterations of *DALIC*. Fig. 3.11 shows a selected part of a 3D point clouds and corresponding superpoints and superpoint normals computed by plane fitting.

Additionally, these eigenvalues give raise to some interesting 3D properties of superpixels. When ordering eigenvalues w_i from smallest to largest and translating to a 2-σ multiplied standard deviation we define: $(d, b, a) = 2(\sqrt{w_1}, \sqrt{w_2}, \sqrt{w_3})$. This defines an ellipsoid with "thickness" $2d$ and major axes a and b which probably contains 95.4% of all points of the superpixel. The following properties give information about the three-dimensional shape of superpixels:

- Superpixel (eigenvalue) thickness: $\textbf{Thickness}_{EV} = 2\,d$
- Superpixel (eigenvalue) eccentricity: $\textbf{Eccentricity}_{EV} = \sqrt{1 - \frac{b^2}{a^2}}$
- Superpixel (eigenvalue) flatness: $\textbf{Flatness}_{EV} = \sqrt{1 - \frac{d^2}{a^2}}$

- Superpixel (eigenvalue) area: $\mathbf{Area}_{EV} = \pi\, a\, b$

Theses properties are evaluated in §3.3.

The complete algorithm of *Depth-Adaptive Superpixels* is summarized in pseudo-code in alg. 4. The sub-modules for *SPDS* and *DALIC* are not detailed further and are described in alg. 1 and alg. 3.

Algorithm 4 *Depth-Adaptive Superpixels* (*DASP*)

```
Require: RGB-D image $I \subset \mathbb{Z}^2$ with colour $\mathfrak{f}_c : I \to [0|1]^3$ and depth $D : I \to \mathbb{R}_+$
Require: Desired superpixel radius $R$
Require: Camera pinhole parameters (i.e. focal length $f$)
  ▷ Annotate each pixel with colour, 3D normal and 3D position (eq. 3.4 and eq. 3.8)
  $\forall\, u \in I : \mathfrak{f}(u) = (\mathfrak{f}_c(u), \mathfrak{f}_v(u), \mathfrak{f}_n(u))$
  ▷ Compute depth-adaptive superpixel density (eq. 3.15)
  $\forall\, u \in I : \rho_{\mathrm{DA}}(u) = \frac{1}{\pi}\left(\frac{D(u)}{f R}\right)^2 \sqrt{\|\nabla D(u)\|^2 + 1}$
  ▷ Annotate pixels with the local search radius (definition 5)
  $\forall\, u \in I : R_\rho(u) = \left(\sqrt{\pi\, \rho_{\mathrm{DA}}(u)}\right)^{-1}$
  ▷ Run SPDS (alg. 1) to get initial cluster centre positions $s_i$
  $\{s_i\}_{1 \le i \le n} = SPDS(\rho_{\mathrm{DA}})$
  ▷ Initialize superpixel centers feature values $\mathfrak{s}_i$
  for $i = 1 \to n$ do
      ▷ Use small neighbourhood around center point $s_i$
      $\mathfrak{s}_i = \mathrm{mean}(\{\mathfrak{f}(u) \mid \|u - s_i\| \le \lambda\, R_\rho(s_i))\})$
  end for
  ▷ Run DALIC (alg. 3) with feature space $\mathscr{F}_{DA}$ (eq. 3.21) and metric $d_{DA}$ (eq. 3.20)
  return $DALIC(\rho_{\mathrm{DA}}, \mathfrak{f}, \{s_i\}, \{\mathfrak{s}_i\})$
```

3.3 Evaluation

For a first impression on the performance of *Depth-Adaptive Superpixels* fig. 3.12 shows an example for input colour and depth images, and superpixels computed with *DASP*. More examples can be found in the appendix in fig. B.1 and fig. B.2. These results were obtained using 1000 superpixels and 5 iterations for *DALIC*.

For a quantitative evaluation of *DASP* a supervised dataset of manually annotated RGB-D images was created to evaluate quality measures which need ground truth segmentations. Ground truth annotations have been created using a toolset described in [7].

In this evaluation three methods are compared:

- The SLIC algorithm [5] – a state-of-the-art superpixel method. This corresponds to *Adaptive Superpixels* with a constant density functions and a metric which only considers colour values ($\mathrm{ASP}_{\mathrm{RGB}}$). The SLIC algorithm

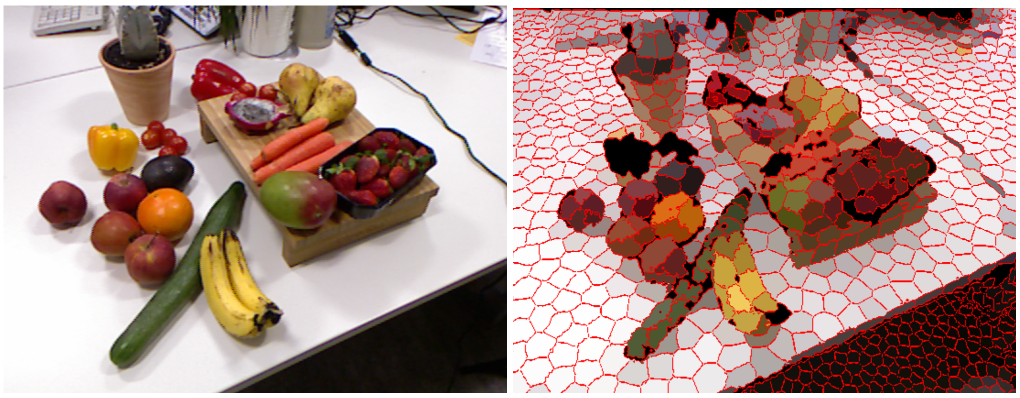


Figure 3.12: Example of *Depth-Adaptive Superpixels*

was thoroughly evaluated in [5] and shows similar qualities than other superpixel algorithms.

- ASP$_{\text{RGBD}}$: *Adaptive Superpixels* with a constant density function and a metric which considers depth and colours. This corresponds to a naive extension of the SLIC superpixel method to RGB-D images.

- *DASP*: The *Depth-Adaptive Superpixels* algorithm which makes full use of the depth information and produces uniform superpoint distributions.

The quality of superpixel segmentations can be evaluated using various measures. Mathematical details of the used metrics are explained in detail in §A.1 in the appendix. The most important metrics for superpixels are undersegmentation error, boundary recall and explained variation resp. compression error. Figure fig. 3.13 shows results for theses four metrics and the remaining figures with more detailed results can be found in the appendix in §B.1.

Undersegmentation error and boundary recall measure how well superpixel boundaries respect segment boundary from a manual annotation by a human. This is a key requirement for a good oversegmentation as superpixels should not merge pixels from different semantic objects. Results for boundary recall measure **BR** (see def. 13) and the undersegmentation error **USE** (see def. 14) are shown in fig. B.3. The *DASP* algorithm has a notably higher performance in recalling boundaries which is mainly due to the intelligent use of depth values and normals. A naive approach which only uses depth information only has a slightly better performance.

Explained variation is presented in more detail in def. 21. It describes how well superpixels can explain the variation of information throughout an image. Results are reported for the feature values colour, depth, position and normals fig. B.6). Due to the high variety of depth and position information, the explained variation metric shows very high values for all metrics. For normal information the explained variation is bigger than 1 which is due to the fact that the raw normals

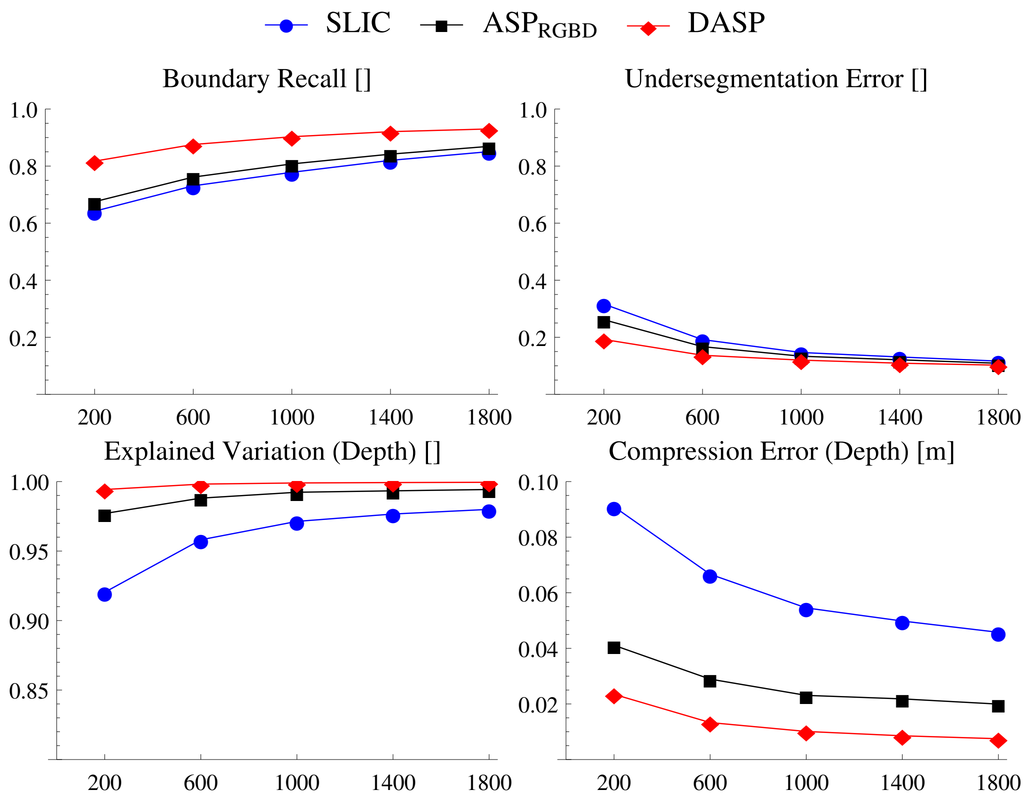


Figure 3.13: Results for boundary recall, undersegmentation error, explained variation (depth) and compression error (depth) for *DASP* and two reference methods plotted over the number of superpixels used.

computed from the depth information are generally very noisy. Results show that the *DASP* algorithm is especially good in explaining depth information which is not surprising as the depth information is not available to the SLIC algorithm, however *DASP* also performs better than a naive *Adaptive Superpixels* on RGB-D images.

The compression error is a more suitable measure to show how well superpixel maintain the image feature information and is explained in detail in def. 20. In fig. B.7 the compression error **CE** for colour, depth, position and normal information is evaluated. Compression error shows similar results than explained variation, but is more informative as it measures local deviation instead of global deviation.

Further results for the isoperimetric quotient and other eigenvalue are reported. Fig. B.4 shows the classic isoperimetric quotient **IPQ** (see def. 18) and the 3D isoperimetric quotient $\mathbf{IPQ_{3D}}$ (see eq. A.13). Properties derived from superpixel ellipsoid matching are visualized in fig. B.5. This includes $\mathbf{Thickness}_{EV}$, $\mathbf{Eccentricity}_{EV}$, $\mathbf{Flatness}_{EV}$ and $\mathbf{Area}_{EV}$.

Evaluation results show that *Depth-Adaptive Superpixels* performs better than a pure colour based approach like SLIC. This is not surprising as the depth values carries additional information which is not available to classical approaches, but it underlines the benefit of depth measurements for image segmentation. Most important however, *DASP* also performs better than a naive approach where depth information is used in addition to colour in form of a primitive direct product of feature space and feature metric. This highlights the fact that the intelligent approach of *Depth-Adaptive Superpixels* is a valuable contribution to fully unlock the potential of depth information for RGB-D image segmentation.

4 S-DASP Image Segmentation

s-DASP is a novel segmentation technique for RGB-D images which generates excellent segments by using both colour and depth information. *s-DASP* is highly efficient as it uses the sparse image representation provided by *Depth-Adaptive Superpixels*.

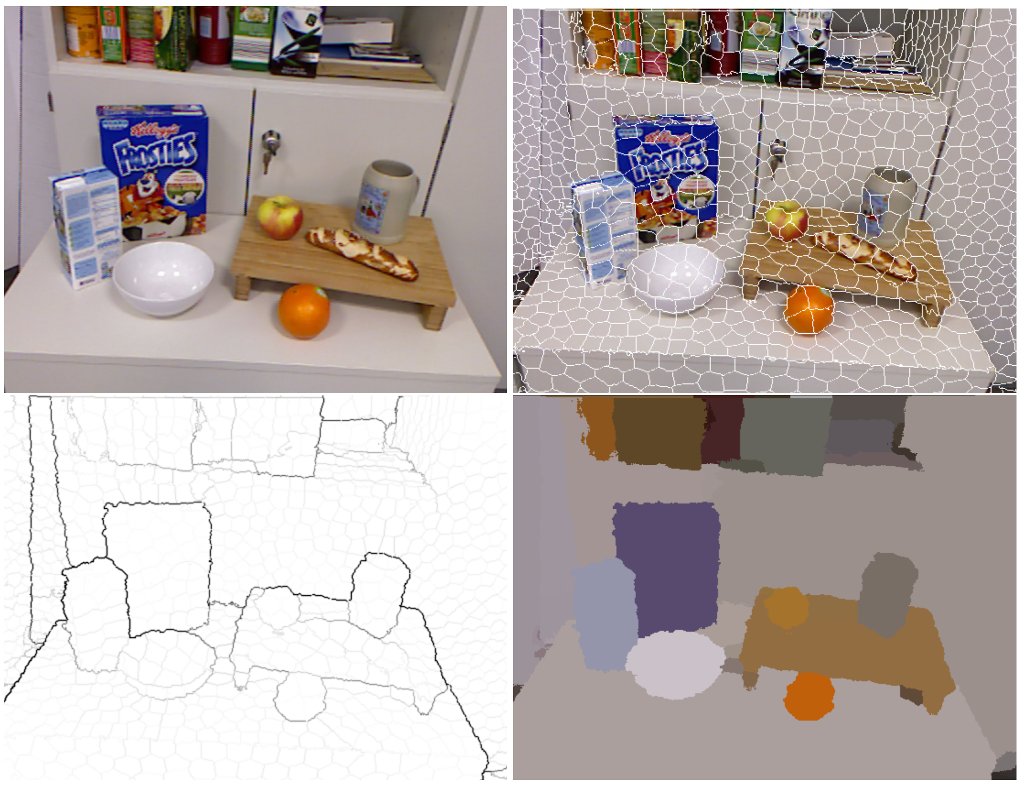


Figure 4.1: **Top**: Input RGB-D image (only colour is shown) and depth-adaptive superpixels. **Bottom**: Ultrametric contour graph indicating boundary strength and final segments.

4.1 Introduction to *s-DASP*

As already introduced in §2.1.2, image segmentation aims towards dividing an image into a set of segments which shall fulfil two criteria: intra-segment similarity and inter-segment dissimilarity [64]. In the previous chapter, the overseg-mentation technique *Depth-Adaptive Superpixels* for RGB-D images has been introduced which provides a solution to the problem of inter-segment similarity (see §3.1). *DASP* divides an image into segments of similar pixels, but different superpixels may be still similar to each other and in general do not satisfy inter-segment dissimilarity. In this chapter, the novel image segmentation technique *Depth-Adaptive Superpixel Segmentation* (*s-DASP*) [81] for RGB-D images is presented. *s-DASP* combines *DASP* superpixels into segments which in addition to inter-region similarity also satisfy inter-segment dissimilarity (see fig. 4.2).

Merging superpixels into larger segments is not a straight forward task as local cues alone are often ambiguous. Global methods try to solve this problem by considering the global ensemble of segments in order to make well-informed decisions about how to merge segments. An important group of methods for global segmentation is spectral graph theory [15, 27, 21, 6, 11, 17]. In the past, many methods have been proposed which make use of spectral techniques [7, 45, 46, 72, 75] for segmentation or matching problems. However, spectral graph analysis requires the computation of eigenvalues and eigenvectors of a matrix which is slower than quadratic in the number of vertices, thus in the number of pixels when operated on the full pixel lattice graph. This severely limits the size of images and imposes high computational costs even when using down-sampled images.

s-DASP tackles this problem by using spectral graph theory on the sparse image representation provided by superpixels through *DASP* instead of using the full image directly (see fig. 4.3). A simple example demonstrates the impact of

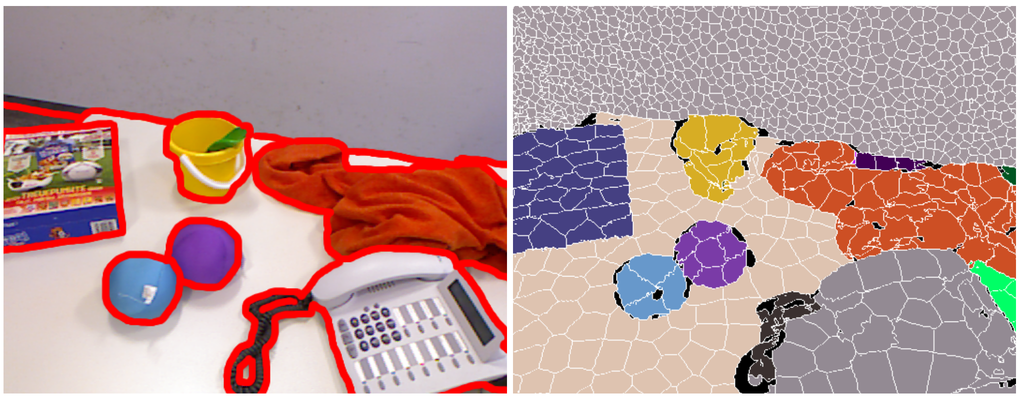


Figure 4.2: **Left**: Colour image of a possible manual ground truth annotation. **Right**: *s-DASP* image segmentation (colours) using depth-adaptive superpixels (white lines).

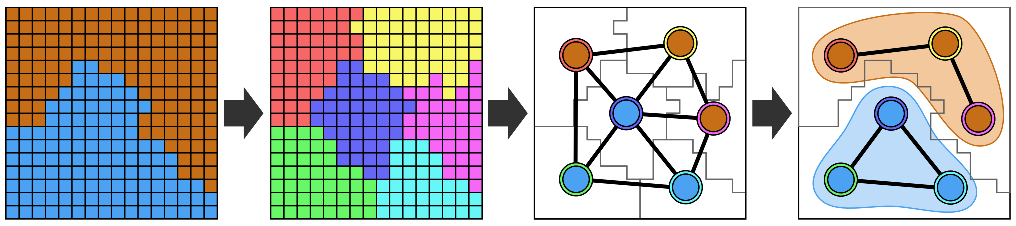


Figure 4.3: **From left to right:** Dense pixel grid, oversegmentation (using false colours), superpixels with superpixel graph and superpixel segments.

using superpixels instead of pixels: The resolution provided by the PrimeSense RGB-D sensor is 640x480 pixels. Using 1200 depth-adaptive superpixels yields an excellent approximation to the real image preserving most of its information content (§3.3). However, downsampling this image to 1200 pixels yields an image of size 40x30, loosing most of the image information. Even using 160x120 pixels, a common image size for image segmentation databases, would still yield 16 times more pixels than superpixels and thus a spectral algorithm would be theoretically several magnitudes slower when assuming a theoretic runtime of $O(n^{2.5})$ for sparse eigenvalue computation.

The following list presents a short overview over the *Depth-Adaptive Superpixel Segmentation* algorithm; details are explained in §4.3.

1. Compute superpixels with the *Depth-Adaptive Superpixels* algorithm.
2. Define a weighted graph structure on superpixels expressing local similarity of neighbouring superpixels.
3. Use spectral graph theory to build an ultrametric contour graph (UCG) over the superpixel graph.
4. Compute a segmentation of the original image with the help of the UCG.

The remainder of this chapter is outlined as follows: In §4.2 spectral graph theory and several related state-of-the-art methods are introduced. The *Depth-Adaptive Superpixel Segmentation* algorithm is presented in detail in §4.3 and a thorough evaluation and comparison to state-of-the-art algorithm is reported in §4.4.

4.2 Spectral graph theory

This section gives a short introduction to spectral graph theory: First some general graph theoretic notations (§4.2.1) are introduced and then the Laplacian of a graph and its relation to connected components of a graph is presented (§4.2.2). Two important state-of-the-art segmentation techniques using spectral graph theory are presented at the end of the section in §4.2.3 and §4.2.4.

4.2.1 General notations for graphs

In the following $G = (V, E)$ will denote an undirected, weighted **graph**. Here V are the graph **vertices**, $n := |V|$ the number of vertices, and E the set of undirected and weighted **edges**. In the following we assume that edges are not directed, edge weights are always greater than zero, two vertices are at most connected by one edge and that no vertices is connected to itself. This includes the cases of a regular graphs, like the image pixel lattice graph, or a more general graph resulting for example from a superpixel segmentation by connecting neighbouring superpixels.

To indicate that two vertices $v, u \in V$ are **connected** by an edge the notation $v \sim u$ will be used. The **weight** of an edge $e \in E$ is noted with $w(e)$ and the weight of an edge between two connected vertices $u \sim v \in V$ with $w(u, v) \in \mathbb{R}_+$. If graph edges are not weighted explicitly, edges are assumed to have an equal weight of 1. The weight of edges connected to vertex is called the **degree** of the vertex.

The edge connectivity can also be expressed by using an **adjacency matrix** $W \in \mathbb{R}_+^{n\times n}$, where W_{ij} is the weight of the edge between nodes i and j or 0 if theses nodes are not connected. The adjacency matrix is symmetric as the graph is assumed to be undirected.

A sequence of vertices $v_1 \sim \cdots \sim v_n$, $v_i \in V$, which are pairwise connected by an edge are called a **path**. The **length** of a path $p = v_1 \sim \cdots \sim v_n$ is defined as as the sum of weights of all its edges:

$$len(p) := \sum_{i=1}^{n-1} w(v_i, v_{i+1}) . \tag{4.1}$$

The **length of the shortest path** between two vertices $v, u \in V$ is donated with $L_{min}(v, u)$. For unweighted graphs, $L_{min}(v, u) + 1$ denotes the number of passed nodes when walking from node u to node v. The length of the shortest path between a vertex $v \in V$ and a subset $U \subset V$ is defined in a straight forward way as $L_{min}(v, U) := \min_{u\in U} L_{min}(v, u)$.

The boundary of a segment is the number of vertices which have a neighbour which is not part of the segment.

Definition 8. Let $G = (V, E)$ be a graph, $\mathfrak{P}$ a partition of the graph nodes V and $S \in \mathfrak{P}$ a segment. The **boundary** $B_G(S)$ of S is defined as

$$B_G(S) := \{v \in S \,|\, \exists\, q \in V \setminus S : v \sim q\} . \tag{4.2}$$

The boundary $B_G(S)$ is the subset of vertices of a segment S which are connected to at least one vertex which is not in S itself. The notion of boundary can be directly extended to partitions.

Definition 9. Let $\mathfrak{P}$ be a partition. The **boundary of the partition** is defined as

$$B_G(\mathfrak{P}) := \bigcup_{S\in\mathfrak{P}} B_G(S) \tag{4.3}$$

4.2.2 The Laplacian matrix of a graph

The Laplacian matrix [17] is a construct in graph theory which gives information about the structure of a graph. We will see in the following, that the number of eigenvalues of the Laplacian matrix $L(G)$ of a graph G which are 0 is equal to the number of connected components of G. As seen in the following sections, these results can be generalized to compute optimal cuts for a connected graph, thus making the Laplacian a valuable tool in graph segmentation.

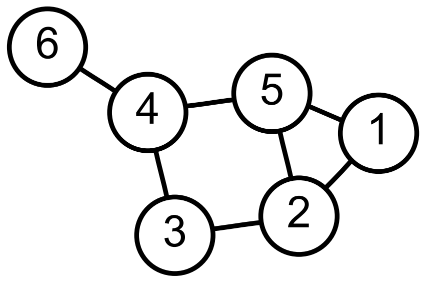
Definition 10. The **degree matrix** $D(G) \in \mathbb{R}_+^{n\times n}$ of the graph G with adjacency matrix $W \in \mathbb{R}_+^{n\times n}$ is the diagonal matrix defined as

$$\big(D(G)\big)_{ii} := \deg(v_i) := \sum_{j=1}^{n} W_{ij}\,. \tag{4.4}$$

Definition 11. The **Laplacian matrix** $L(G) \in \mathbb{R}^{n\times n}$ of the graph G with adjacency matrix $W \in \mathbb{R}_+^{n\times n}$ is defined as

$$L(G) := D(G) - W\,. \tag{4.5}$$

Fig. 4.4 shows an example of a simple graph and its Laplacian matrix. All edge weights are 1 and the numbers indicate node labels and thus row/colum index.



$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Figure 4.4: A simple graph [2] and its Laplacian matrix.

Lemma 2. *The Laplacian matrix $L(G)$ is positive-semidefinite.*

Proof. Consider an edge $e = (e_1, e_2) \in E$ of G with weight $w_e := W_{e_1 e_2} > 0$. We define the edge Laplacian matrix

$$\big(L(e)\big)_{ij} := \begin{cases} +w_e & \text{if } i = j = e_1 \text{ or } i = j = e_2 \\ -w_e & \text{if } i = e_1 \text{ and } j = e_2 \text{ or } i = e_2 \text{ and } j = e_1 \\ 0 & \text{otherwise} \end{cases}$$

The edge Laplacian matrix is positive semi-definite as

$$\forall\, x \in \mathbb{R}^n : x^T L(e)\, x = w_e\, (x_{e_1} - x_{e_2})^2 \geq 0$$

The Laplacian matrix $L(G)$ can be written as the sum of it's edge Laplacian matrices: $L(G) = \sum_{e\in E} L(e)$. Thus we have

$$x^T L(G)\, x = \sum_{e\in E} x^T L(e)\, x = \sum_{e\in E} w_e\, (x_{e_1} - x_{e_2})^2 \geq 0 \tag{4.6}$$

□

As $L(G)$ is symmetric and positive-semidefinite, all its eigenvalues are real and positive or zero.

Lemma 3. *The smallest eigenvalue of the Laplacian matrix* $L(G)$ *is always* 0. *The corresponding eigenvector is* $(1,\ldots,1)^T \in \mathbb{R}^n$.

Proof. $W\,(1,\ldots,1)^T = (\sum_{j=1}^n W_{ij})_i$, thus $L(G)\,(1,\ldots,1)^T = 0$, and thus $(1,\ldots,1)^T$ is an eigenvector of $L(G)$ with eigenvalue 0. □

Proposition 1. *The Laplacian matrix* $L(G)$ *has an eigenvalue 0 for each connected component of* G.

Proof. Let $I \neq \emptyset$ be the set of indices of the nodes of one of the connected components. Define $v \in \mathbb{R}^n$ as

$$v_i := \begin{cases} 1 & i \in I \\ 0 & i \notin I \end{cases}$$

We have $L(G)\, v = 0$ for the same reason as in the proof of lemma 3, as edges with weight greater than 0 are only formed between vertices of the conneted component. Thus v is an eigenvector with eigenvalue 0. Eigenvectors constructed in this way are pair-wise orthogonal, thus follows the assertion. □

Lemma 4. *If the graph G is connected, only the smallest eigenvalue is 0.*

Proof. Let $x \in \mathbb{R}^n$ such that $L(G)\,x = 0$. By eq. 4.6 we have

$$0 = x^T L(G)\, x = \sum_{e\in E} w_e\, (x_{e_1} - x_{e_2})^2\,,$$

which implies $x_{e_1} = x_{e_2}$ for all $e = (e_1, e_2) \in E$. As the graph G is connected, there is a path from each vertex to each other vertex, thus all values of x must be equal. This implies that x is a multiple of the eigenvector $(1,\ldots,1)^T \in \mathbb{R}^n$. □

For a connected graph, the second smallest eigenvalue λ_2 is called the **Fiedler vector** [27] and gives interesting information about how well a graph can be cut into two segments in terms of minimizing the weight of cut edges.

Definition 12. Let $A \cap B = V$ a partition of the graph $G = (V, E)$. The **cut** of a the partition is defined as

$$\mathrm{cut}(A,B) := \sum_{a\in A, b\in B} w(a,b) \tag{4.7}$$

The cut of a two-element partition represents the total weight of edges which have to be cut to disconnect the two segments. It is 0 if the segments are disconnected and small for segments which are only connected by few edges with low weights.

The minimal possible cut can be estimated using the Fiedler vector:

Proposition 2 (Cheeger's Inequality)**.** *Let* $W_{ij} \in \{0,1\}$ *and* d *be the maximial degree of all vertices. Define*

$$h_* := \min_{A \cap B \subset V} \frac{cut(A,B)}{\min(|A|,|B|)},$$

then

$$\frac{h_*^2}{2d} \leq \lambda_2 \leq h_* \,. \tag{4.8}$$

Proof. See [15, 17, 46]. □

Thus the Fielder vector indicates how well a graph can be partitioned into two segments in the means of cutting as few edges as possible. These results can be extended to a better behaving cut measure resulting in the normalized cuts algorithm explained in the next section.

4.2.3 The normalized cuts algorithm

The normalized cuts algorithm by Shi et al. [72] presents a new measure for the cut of a graph and provides an algorithm for finding the minimal cut under this measure based on the Fiedler vector of a generalized eigenvalue problem. In contrast to the classical cut measure [84], the normalized cut measure is normalized with respect to the total possible connectivity of a subset of the graph vertices to itself. This has the advantage that it does not extensively prefer to cut of small segments.

The normalized cut measure of a partition $A \cap B = V$ is defined as

$$\mathrm{Ncut}(A) := \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(A,V)} + \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(B,V)} \,. \tag{4.9}$$

Finding the minimal cut is an NP-hard problem, but approximate solutions can be found by embedding the problem in the domain of real numbers. Representing a subset of V by its indicator vector $z \in \{0,1\}^{|V|}$, it can be shown that the minimal normalized cut is found by solving

$$\min_z \mathrm{Ncut}(z) = \min_x \frac{x^T L(G)\, x}{x^T D(G)\, x} \tag{4.10}$$

under some additional constraints on x. Moreover one can see that the second-smallest eigenvector of the generalized eigenvalue problem

$$L(G)\, x = \lambda\, D(G)\, x \,. \tag{4.11}$$

is an approximative solution to eq. 4.10. For details see [72].

If G is connected, the degree matrix $D(G)$ has only non-zero, positive entries. In this case the generalized eigenvalue problem eq. 4.11 can be transformed into a standard symmetric eigenvalue problem:

$$D(G)^{-\frac{1}{2}}\,L(G)\,D(G)^{-\frac{1}{2}}\,y = \lambda\,y\,. \tag{4.12}$$

Eigenvectors for the original problem can be computed from the transformed problem with $x = D(G)^{-\frac{1}{2}}\,y$. For a diagonal matrix $D = (d_i)_i$ one defines $D^{-\frac{1}{2}} = (d_i^{-\frac{1}{2}})_i$.

On can observer that additional eigenvectors to eigenvalues after the second smallest provide further partitions. The Meila-Shi algorithm [53] uses this observations to find image segmentations without the need of iterative cutting.

4.2.4 Globalization of a local boundary detector

Arbelaéz et al. [7] presented a spectral pixel boundary detector sPb for normal colour images which uses spectral graph theory to enhance a local multi-scale boundary detector (mPb) with global image information. For this method the complete pixel lattice graph is used as the graph $G = (V, E)$, and the weight of an edge connecting two pixels i, j is defined as

$$W_{ij} := \exp\left(-\frac{1}{\rho}\max_{p\in\overline{ij}}\mathrm{mPb}(p)\right) \tag{4.13}$$

if the distance between pixels i and j is smaller than a fixed radius R and 0 otherwise. $\overline{ij}$ is the line connecting pixels i and j. W_{ij} defines pixel similarity be separating pixels which lie on opposing sides of a strong boundary. For this graph the k smallest eigenvalues λ_i and eigenvectors v_i of eq. 4.11 are computed and the observation is made that eigenvectors carry contour information. Thus a Gaussian directional gradient filter is applied for varying orientations resulting in the orientation dependent, global boundary detector sPb:

$$\mathrm{sPb}(p,\theta) := \sum_{i=2}^{k}\frac{1}{\sqrt{\lambda_i}}\nabla_\theta\,v_i(p)\,. \tag{4.14}$$

Finally a linear combination between local and global boundary detectors is used to compute a globalized boundary propability for each pixel.

4.3 Depth-Adaptive Superpixel Segmentation

In the following the *Depth-Adaptive Superpixel Segmentation* (*s-DASP*) algorithm for RGB-D images is presented in detail. *s-DASP* uses a similar idea as sPb from

§4.2.4 and promotes a local boundary detector to a global detector by using spectral graph theory. However *s-DASP* does not work directly on the dense pixel grid but on a sparse representation – the set of superpixels provided by *DASP*. This demonstrates how a complex theoretical theory like spectral graph segmentation can be applied in realtime by using an efficient sparse representation like superpixels instead of the dense pixel grid. To make it possible, some new ideas have to be introduced to transport the idea of globalization from regular to arbitrary graphs. However it will turn out that this generalization is actually a simplification. In addition *s-DASP* works with RGB-D images which results in a fundamentally better segmentation quality due to the additional consideration of depth information.

§4.3.1 describes how a superpixel neighbourhood graph is built on top of *Depth-Adaptive Superpixels*. In §4.3.2 this neighbourhood graph is transformed into an ultrametric contour graph (UCG) using spectral graph theory. The UCG is used to compute a graph segmentation and thus the final image segmentation in §4.3.3.

4.3.1 Superpixel neighbourhood graph

The starting point of the *Depth-Adaptive Superpixel Segmentation* algorithm is an oversegmentation of an RGB-D image into a set of superpixels computed with *DASP*. This superpixel partition $\mathfrak{P}$ consists of a set of depth-adaptive superpixels, where each superpixel $S \in \mathfrak{P}$ is annotated with a feature vector $\mathfrak{s} \in [0|1]^3 \times \mathrm{S}_0^2 \times \mathbb{R}^3$. The components of the feature vectors are the mean colour $\mathfrak{s}_c$, mean position $\mathfrak{s}_v$ and mean normal $\mathfrak{s}_n$ of the corresponding pixels.

s-DASP starts by defining a graph structure $G(\mathfrak{P}) = (\mathfrak{P}, E(\mathfrak{P}))$ on the set of superpixels by connecting all neighbouring superpixels with edges. Two superpixels are neighbours, if the superpixel as sets of pixels share a common border on the pixel grid:

$$(i,j) \in E(\mathfrak{P}) \text{ iff. } 1 \le i \ne j \le n \wedge \exists\, p \in S_i, q \in S_j : p \sim q\,. \tag{4.15}$$

Due to noisy superpixel boundaries it is beneficial to require that the common boundary has a minimal length:

$$(i,j) \in E(\mathfrak{P}) \text{ iff. } 1 \le i \ne j \le n \wedge |\{(p,q) \in S_i \times S_j \mid p \sim q\}| > \Theta\,. \tag{4.16}$$

In other words, the superpixel neighbourhood graph connects two superpixels $S_i \ne S_j$ if $\operatorname{cut}(S_i, S_j) > \Theta$ with respect to the pixel lattice graph. The superpixel neighbourhood graph for depth-adaptive superpixels has a well behaving regular almost hexagonal structure due to the uniform distribution of superpixels. Fig. 4.5 shows the superpixel graph for an example image.

Superpixel feature values $\mathfrak{s}_i$ can be used to further improve the superpixel neighbourhood graph by weighting each edge according to the similarity of the connected superpixels. Similar superpixels have similar colour, are spatially close and have normals which point in the same direction. Here a similarity measure on a superpixel partition is defined by using a distance function d_S on superpixel

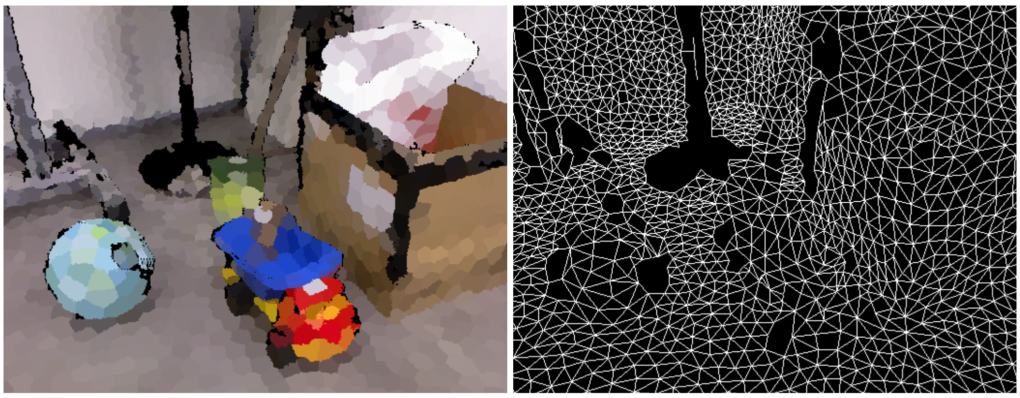


Figure 4.5: **Left**: Depth-adaptive superpixels for an RGB-D image. **Right**: Neighbourhood graph defined on superpixels.

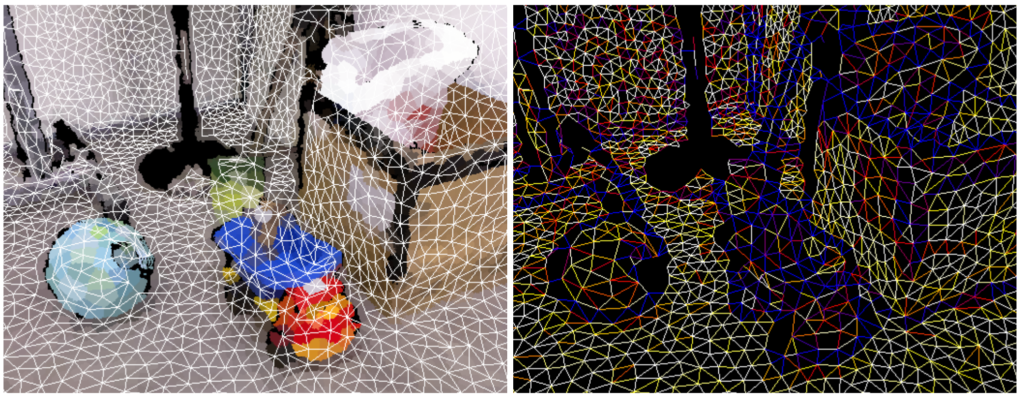


Figure 4.6: **Left**: Unweighted depth-adaptive superpixel neighbourhood graph. **Right**: Depth-adaptive superpixel similarity graph weighted with $W_{\mathfrak{s}}$ (blue to red to white indicates low to high similarity).

features. The metric d_S is a linear combination of metrics defined over superpixel positions, normals and colour:

$$d_S(\mathfrak{s},\mathfrak{s}') := \sum_{\circ=v,n,c} \lambda_\circ \, d_{S,\circ}(\mathfrak{s}_\circ,\mathfrak{s}'_\circ) \tag{4.17}$$

The distance value is translated into a similarity value using an exponential function:

$$W_{\mathfrak{s}}(i,j) := \begin{cases} e^{-d_S(\mathfrak{s}_i,\mathfrak{s}_j)} & \text{if } (i,j) \in E(\mathfrak{P}) \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

Edge weights are ranging from 1 for similar superpixels to almost 0 for completely dissimilar superpixels. Fig. 4.6 shows the weighted superpixel graph for an example image.

The individual metrics $d_{S,\circ}$ are chosen as follows:

Colour superpixel metric Basically, this can be any colour metric. For simplicity the Euclidean distance of RGB colour values like in *DASP* is chosen.

$$d_{S,c}(c_1, c_2) := \|c_1 - c_2\| \tag{4.19}$$

Position superpixel metric As depth-adaptive superpixels are uniformly distributed in 3D space and have equal radius of R, the expected distance between two neighbouring superpixels is $2R$. Thus for spatial distance a metric is used which tolerates the expected distance:

$$d_{S,v}(v_1, v_2) := \frac{\max(0, \|v_1 - v_2\| - 2R)}{2R} \tag{4.20}$$

The Euclidean distance between superpixel centres is additionally normalized by the estimated distance of $2R$ to be independent of the size of superpixels.

Normal superpixel metric Normal distance could be measured by the angle between the normals. However, for three dimensional objects there is a fundamental difference between concave and convex geometry edges. While concave edges often represent an object boundary, convex edges almost always do not indicate a segment boundary. Thus here a metric is used which only considers the normal distance for superpixels which form a concave edge and yields 0 for superpixels which form a convex edge:

$$d_{S,n}(n_1, n_2) := \max\left(0, (n_1 - n_2) \circ \frac{v_1 - v_2}{\|v_1 - v_2\|}\right) \tag{4.21}$$

Here v_1 and v_2 are the corresponding 3D point positions.

4.3.2 A global segment boundary detector for superpixels

For the third step of the algorithm, spectral graph theory from §4.2 is used to compute an ultrametric contour graph (UCG) on the superpixel similarity graph. An UCG is the generalization of an ultrametric contour map (UCM) [8] for non-regular graphs. This approach is motivated by the boundary detector sPb presented in §4.2.4 which considers the full pixel lattice graph and improves a local boundary detector by computing eigenvalues of the Laplacian. However, eigenvalue computation has a runtime of approximately $O(n^{2.5})$, thus the performance of sPb is severely impacted by the size of the graph. The size of the Laplacian is quadratic in the number of image pixels, and even if a sparse matrix representation is used this is still a huge number of elements. In this section a more elegant boundary computation algorithm is presented which uses a non-regular superpixel graph instead of the dense pixel lattice graph. This does not change the theoretic, asymptotic runtime behaviour, but a number of vertices which is 1000 times smaller has a huge practical impact.

The three steps of the *s-DASP* global segmentation step are as follows:

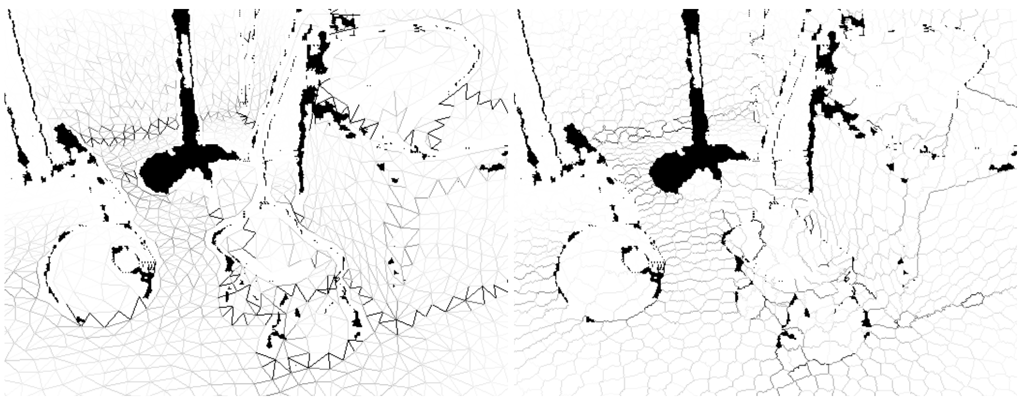


Figure 4.7: **Left**: Ultrametric contour graph on a *DASP* neighbourhood graph. **Right**: Derived ultrametric contour map on the full image.

1. Compute the graph Laplacian of the superpixel similarity graph
2. Compute k smallest eigenvalues and corresponding eigenvectors
3. Compute the superpixel ultrametric contour graph

The notion ultrametric contour graph is motivated by the concept of an ultrametric contour map. An UCM is a function into the positive real numbers defined on the pixels of an image which indicates the likelihood that a pixel is part of a segment boundary. In contrast an ultrametric contour graph is defined on a non-regular graph of an oversegmentation – here the neighbourhood graph of depth-adaptive superpixels. Each vertex represents a set of pixels in the image and each edge represents the pixel boundary between the connected superpixels. The edge weight indicates the likelihood that this pixel boundary piece is part of segment boundary. Thus an UCG is the sparse analogy of an UCM. It represents a set of similar boundary pixels, i.e. the boundary of two superpixels, by a single data point, i.e. the edge weight. An UCG can be converted into an UCM: for each edge in the graph, the edge weight value is assigned to all segment boundary image pixels represented by the edge (see fig. 4.7).

Given a connected superpixel graph G with adjacency matrix $W \in \mathbb{R}_+^{n \times n}$, the k smallest eigenvalues λ_t of the generalized eigenvalue problem in eq. 4.12 and corresponding eigenvectors v_t are computed. The first eigenvalue with value 0 is not used as it does not provide any information as explained in lemma 3. Each eigenvector assigns a real number to each superpixel. If the absolute difference of values of adjacent nodes is small this indicates a similar labeling, if it is large a different label is indicated. The smaller the corresponding eigenvalue, the stronger the similarity. With this observations, new edge weights on the superpixel graph structure are defined by the adjacency matrix W^{global} as follows:

$$W_{ij}^{\text{global}} := \begin{cases} \sum_{t=1}^{k} \frac{1}{\sqrt{\lambda_t}} |v_{ti} - v_{tj}| & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

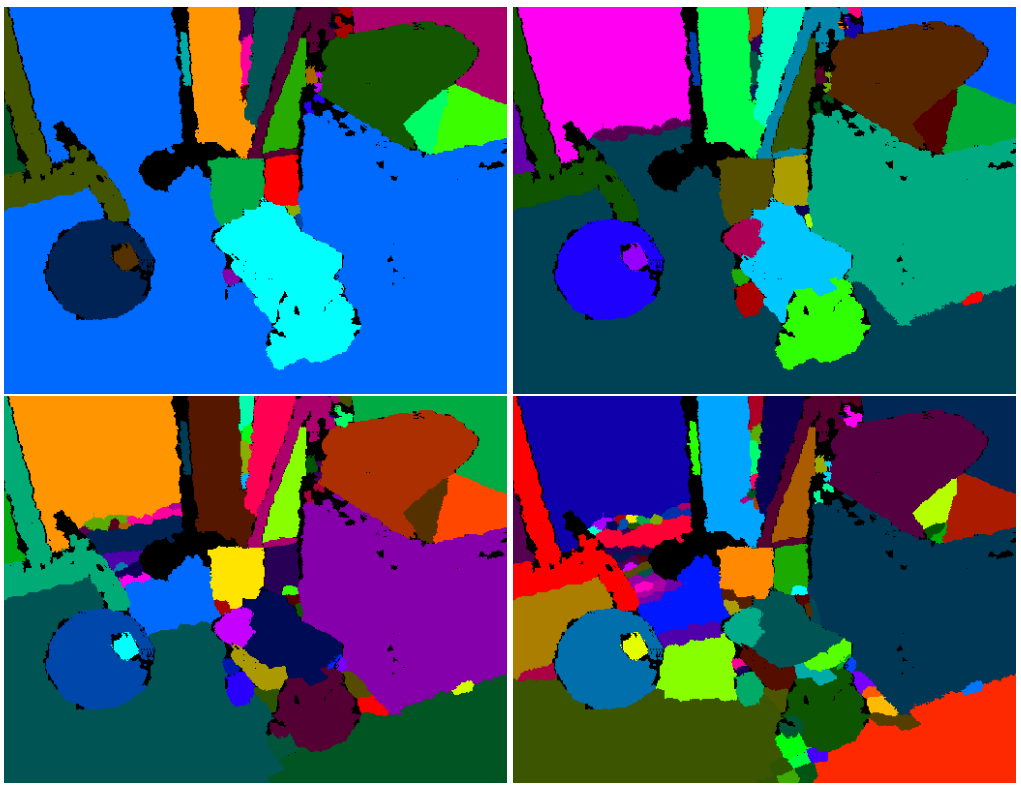


Figure 4.8: *s-DASP* image segmentations for varying edge cut thresholds.

For the superpixel graph this much simpler approach using local difference is sufficient as boundary values are not defined *on* graph nodes but *between* nodes on the edges. Thus W^{global} defines an ultrametric contour graph on the superpixel neighbourhood graph. This implies a ultrametric contour map on the full image by transferring the boundary strength of an graph edge to all pixels which form the pixel boundary between superpixels connected by the edge.

4.3.3 Segmentation of the ultrametric contour graph

A segmentation can be performed on an UCG by thresholding edge weights and computing connected components. This process can also be seen in the general context of the UCM as iteratively increasing the cut threshold and merging one edge after another starting with edges with the lowest weight. At each step superpixels which are connected by the pruned edge are merged into the same semantic group. This process creates a continuous transition from the partition where each superpixel is a segment on its own to the partition of one segment which contains all superpixels. Several steps in such a transition are shown in fig. 4.8.

As weights in the UCG depend on many parameters and especially on the topology of the superpixel neighbourhood graph, it is not easy to derive the optimal cut threshold. Instead the parameter is used as a free parameter which can be adapted to the requirements of a specific applications.

The segmentation on the UCG graph is directly transformed into a segmentation of the underlying pixel grid by assigning to each pixel the segment ID of its corresponding superpixel. During this process one of the advantages of using superpixels like *Depth-Adaptive Superpixels* as the underlying elements become apparent: the segmentation on the sparse graph generates pixel-accurate image segments for the full dense pixel grid. This is not possible if a down-sampled version of the image would have been used.

4.4 Evaluation

Precision and recall are well-known measures from information retrieval which are widely used to evaluate the quality of a binary classifier:

$$\text{precision}(A|G) = \frac{|G \cap A|}{|A|} \tag{4.23}$$

$$\text{recall}(A|G) = \frac{|G \cap A|}{|G|} \tag{4.24}$$

Here G is the set of ”relevant” elements which should have been detected and A is the set of ”retrieved” element which were actually detected by the classifier. It is important to always consider both quantities together. A detector which only marks elements as detected if it is very sure can still have a high precision value, and a detector which marks almost everything as detected can have a high recall value. As precision and recall formulate different goals which in most cases are opposite to each other, the combined measure F_1-score is used to measure the overall accuracy of a detector:

$$F_1 := 2\,\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.25}$$

In the context of image segmentation, segmentations computed by computer algorithms are often compared against manual ground truth annotations from humans. As it is difficult to provide a measure to identify segments, a common method is to compare the boundary of the computed partition $\mathfrak{P}$ against the ground truth partition $\mathfrak{X}$. Thus precision, recall and F_1 score are defined as

$$\text{precision}(\mathfrak{P}|\mathfrak{X}) = \frac{|B(\mathfrak{P}) \cap B(\mathfrak{X})|}{|B(\mathfrak{P})|} \tag{4.26}$$

$$\text{recall}(\mathfrak{P}|\mathfrak{X}) = \frac{|B(\mathfrak{P}) \cap B(\mathfrak{X})|}{|B(\mathfrak{X})|} \tag{4.27}$$

$$F_1(\mathfrak{P}|\mathfrak{X}) = 2\,\frac{|B(\mathfrak{P}) \cap B(\mathfrak{X})|}{|B(\mathfrak{P})| + |B(\mathfrak{X})|} \tag{4.28}$$

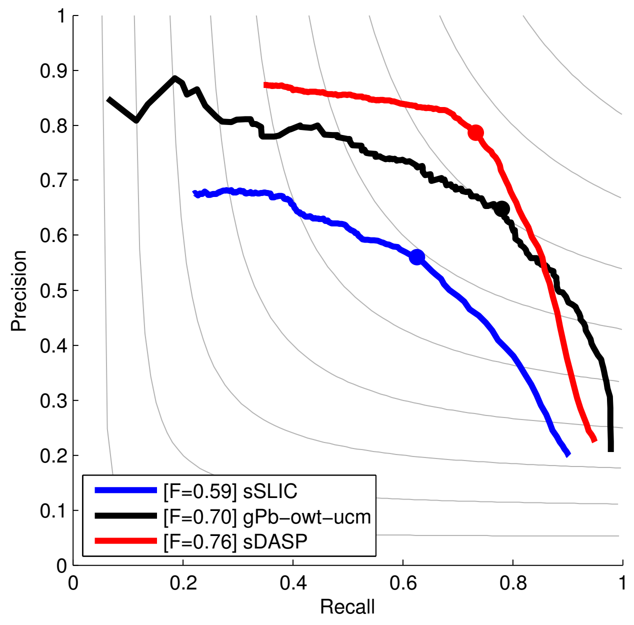


Figure 4.9: Precision/recall for varying cut threshold for *s-DASP*, *s-SLIC* and gPb-owt-ucm.

In the case of *Depth-Adaptive Superpixel Segmentation* the quality of computed segmentations is compared against manual ground truth on the supervised dataset from §3. The segmentation uses a threshold parameter when concrete segments are computed from the ultrametric contour graph. The higher the threshold the more segments are merged, the smaller the boundary of the finale partition, and thus the higher is precision. On the contrary a low threshold leads to more segments and thus to a low precision and a high recall. Thus the behaviour of *s-DASP* is analysed by plotting a curve in a precision/recall coordinate system and computing the maximal possible F_1 score. Fig. 4.9 shows results for precision, recall and F_1 for three algorithms:

- *s-DASP*: the presented segmentation algorithm which uses *DASP* superpixels and makes full use of the depth information.
- *s-SLIC*: Like *s-DASP* but SLIC is used instead of *DASP* to compute superpixels. No depth information is used.
- As a comparison to state-of-the-art algorithms, the dense boundary detector gPb-owt-ucm (see §4.2.4 and [7]) is used. No depth information is used.

For a fair comparison, all three algorithm have been parametrised to approximately give the same total number of segments for the final segmentation. Results for *s-DASP* and *s-SLIC* are performed with 1000 superpixels and 5 iterations and default values for metric weighting parameters. gPb-owt-ucm also uses a superpixel technique but does not allow an explicit control over the number of superpixels.

Results demonstrate the superiority of *s-DASP* to classical image segmentation algorithms which do not use depth information. The maximial F_1 score of 0.76 is actually already quite near to a typical human F_1 score of around 0.8 [7]. Humans do not achieve a perfect F_1 score of 1 when compared to each other due to different understandings about the definition of a good segment.

In terms of runtime performance, *s-DASP* and the preprocessing step *DASP* together require in average 250ms on a standard single core CPU. This lies in stark contrast to the runtime of gPb-owt-ucm, which was in average 210 seconds - a difference of three magnitudes.

Fig. 4.10 shows several segmentation results for images from the ground truth data set. All images have been computed with identical parameters and the optimal cut threshold after fig. 4.9 was used. *s-DASP* is capable of merging large planer surfaces into one segment while at the same time preserving small segments with only few superpixels. It is also apparent how the depth information is used to distinguish ambiguous situations where objects with similar appearance are spatially close to each other. Sometimes at narrow ridges singular superpixels are split off, which is due to the shortcomings of spectral graph theory.

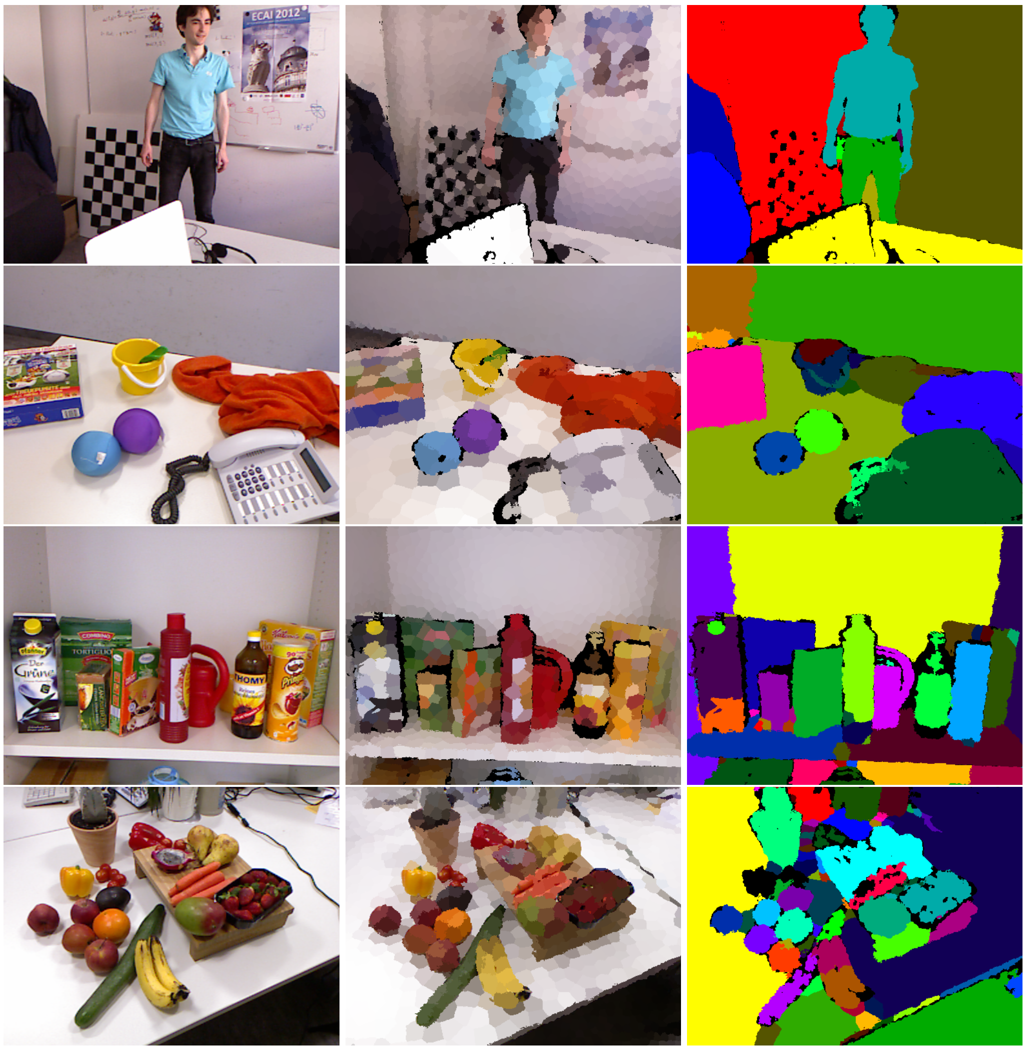


Figure 4.10: **Top to bottom:** Four examples for *s-DASP* segmentations. **Left to right:** Colour image, *DASP* superpixels and *s-DASP* segments (false colours indicating segments).

5 T-DASP Video Segmentation

t-DASP is a novel streaming video analysis technique for RGB-D video streams which generates temporal and spatial coherent image segmentations. *t-DASP* is highly efficient as it builds upon the sparse image representation provided by *Depth-Adaptive Superpixels*.

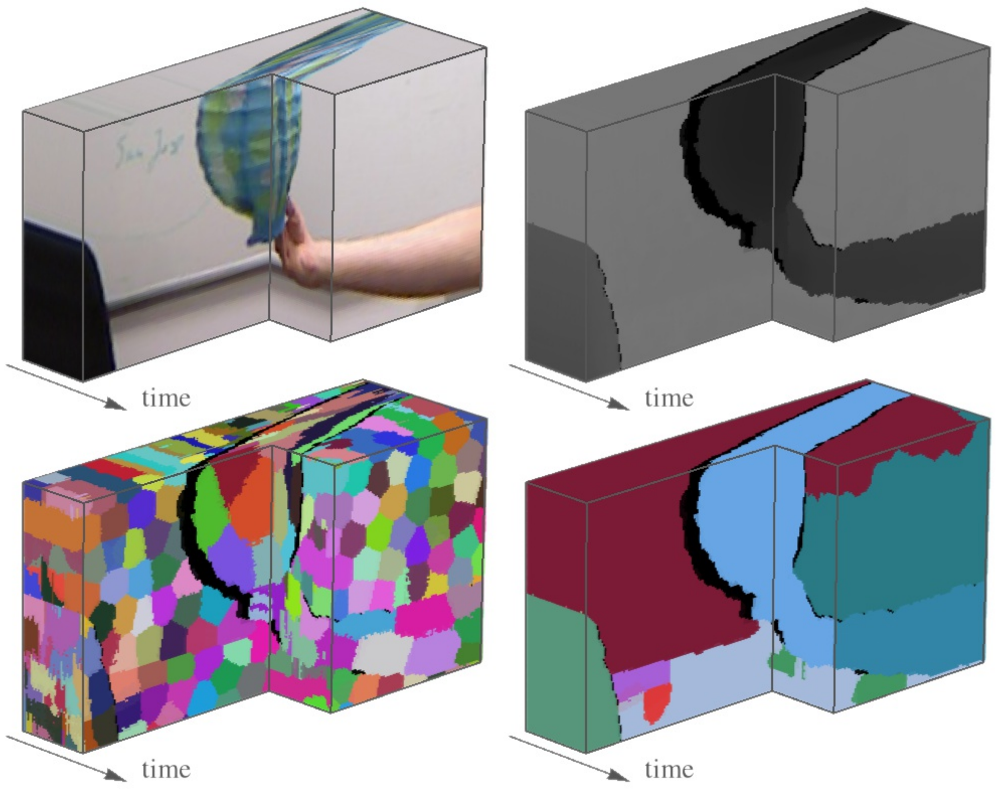


Figure 5.1: **Top**: Extract of an RGB-D video stream (colour and depth). **Bottom**: Temporal-stable depth-adaptive superpixels and *t-DASP* video segmentation.

5.1 Video segmentation

5.1.1 Introduction

Video segmentation is the extension of image segmentation to a temporal sequence of images. Ideally the sequence consists of similar images varying only slightly over time as the camera pose or objects in the scenery change. While the diversity in appearance is already high for static scenes due to occlusions and varying shapes, the extension to the temporal domain adds an additional layer of complexity. Temporal changes include change of perspective due to camera movement or rotation, relative change of positions or orientation of objects, articulations of hierarchical objects like the limbs of a human body, deformation of soft bodies like squeezing a sponge, or physical phenomena like smoke or the flow of water. Additionally the rate of change may vary dramatically. Sometimes almost none or only well-behaved and easy to follow incremental changes happen, sometimes the rate of change exceeds the framerate of the camera resulting in motion blur. In moderated video streams, like TV broadcast or films, cuts are introduced which abruptly change the viewpoint or context from one frame to the next.

Formally, a **video** is an extension of a spatial image domain Ω by a temporal component, resulting in a spatio-temporal domain $\Omega \times \mathbb{R}_+$. For the sake of simplicity, only discrete domains will be considered in this context. Additionally it is assumed that each image in the video, so called **video frame**, has the same spatial resolution, and that the video framerate, i.e. the number of images per second, is constant. Thus the spatial domain is a two-dimensional finite lattice-graph $\mathcal{G}_I$ (the pixel grid) which is extended by a discretised temporal component $\mathbb{N}$ to form a three-dimensional **spatio-temporal finite lattice-graph** $\mathcal{G}_V = \mathcal{G}_I \times \mathbb{N}$. Between pixels in consecutive frames there is a temporal connection, thus each pixel has 26 neighbours: nine spatio-temporal neighbours each to pixels in the previous and next frame, and eight spatial neighbours to pixels in the current frame. In the following elements in $\mathcal{G}_V$ will be called **spatio-temporal pixels**, or simply **voxels** .

Regarding the length of the video, an important differentiation has to be made: When speaking of a video, we assume that the video has a finite length and that all video frames are recorded and known before starting the video segmentation. On the other side, for a **video stream** the video is still in the process of being recorded and only frames up to now are known. This differentiation is relevant, because online processing of a video streams implies that only frames from the past up to now are available. Only offline video processing has access to the whole relevant video sequence and can consider both the past and the future when analysing a specific video frame. In video segmentation memory is a crucial resource as already small videos have a huge footprint: For example, an uncompressed RGB video with resolution 640x480 and a framerate of 30 Hz already requires 1.5 GB per minute. Thus the number of frames which can be considered for analysis is

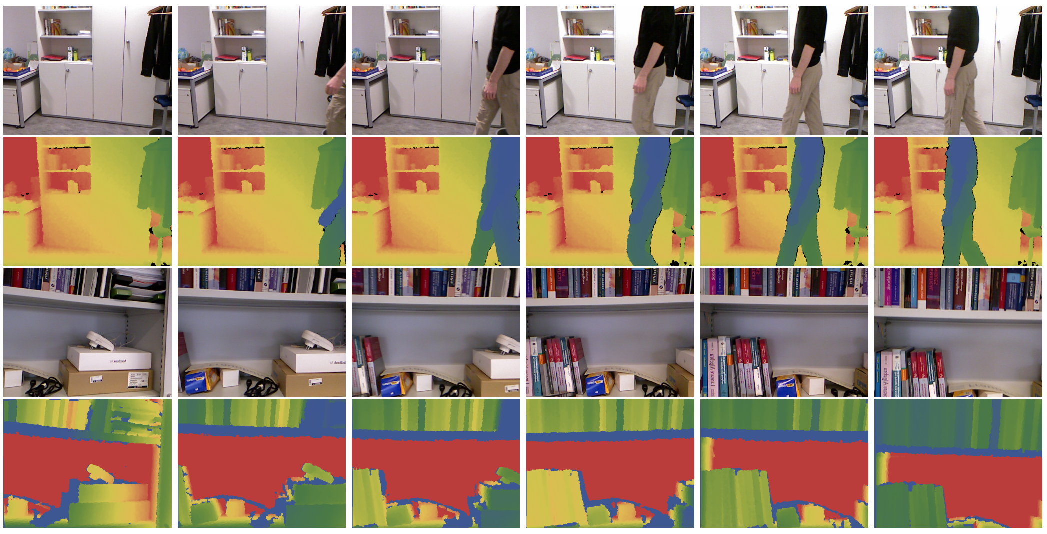


Figure 5.2: Two examples of RGB-D video streams recorded with PrimeSense. Each shows the colour stream and the depth stream (colour encoding depth).

limited for video processing. Later the context of a video stream will be reduced by only considering the recent past and not even the full history of frames. This additionally allows a much more efficient resource usage as only recent frames have to be kept in memory.

As the structure of videos has been fixed, the crucial information is carried by a feature annotation, i.e. a mapping for each voxel into a feature space $\mathscr{F}$. For each time step k a different mapping $\mathbf{I}^{(k)} : \mathscr{G}_I \to \mathscr{F}$ into the image feature space $\mathscr{F}$ is provided and the mapping of the whole video sequence into the feature space is noted with $\mathbf{V} : \mathscr{G}_V \to \mathscr{F}$, thus $\mathbf{V} = (\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots)$. Classically the feature space is an RGB colour space, e.g. $\mathscr{F} = [0|1]^3$, but in the following especially RGB-D video streams, e.g. $\mathscr{F} = [0|1]^3 \times \mathbb{R}_+$, will be considered. Fig. 5.2 shows examples of an RGB-D video stream recorded with a PrimeSense sensor.

The principles of image segmentation can be transferred to sequences of images in basically two ways. On the one hand, image segmentation techniques can be extended from a two-dimensional lattice graph to the spatio-temporal three-dimensional lattice graph $\mathscr{G}_V$, essentially treating temporal connections equal to spatial connections. While this may sound counter-intuitive, one has to keep in mind that spatial neighbours in an image are already not equal among each other. The fact that neighbouring pixels are close in the image domain does not imply that they are semantically related to each other, i.e. close in 3D space or belonging to the same object. On the other hand video segmentation can be viewed as a two-phase process: first, segmentations of individual frames are computed, and second, these segmentations are matched between each other to establish temporal coherence.

Temporal Depth-Adaptive Superpixels (*t-DASP*) [83] is a novel video stream segmentation method which uses a mixed approach to combine spatial and tem-

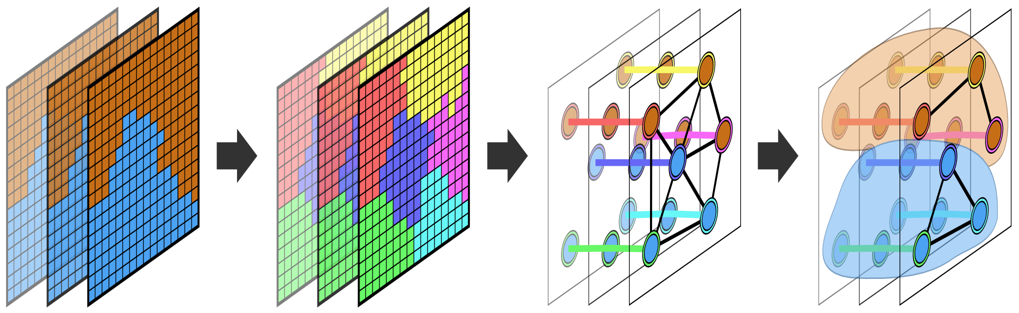


Figure 5.3: *Depth-Adaptive Superpixels* are formed into *Temporal Depth-Adaptive Superpixels*. **From left to right:** dense voxel grid, oversegmentations (using false colours), superpixels strands, and strand graph with superpixel strand segments.

poral segmentations. First, an enhanced version of *Depth-Adaptive Superpixels* is used to compute a temporal-stable oversegmentation for each video frame. Between theses frames, superpixels are connected to form temporal "strands" of superpixels. A **strand** of superpixels is simply a temporal sequence of superpixels which represent the same piece of surface over time. The corresponding superpixel strand graph is built using the superpixel graph, and used in a spectral graph segmentation process similar to *s-DASP* to compute a video segmentation. Fig. 5.3 illustrates the *Temporal Depth-Adaptive Superpixels* video segmentation algorithm.

In the following a state-of-the-art streaming video segmentation framework is presented (see §5.1.2) which uses a hierarchical segmentation approach. Then the temporal-stable modification for *DASP* is discussed in §5.2. Later the *t-DASP* algorithm is discussed by presenting the method to create superpixel strands (§5.3) and the strand graph segmentation process (§5.4). The chapter is concluded with an evaluation of *t-DASP* in §5.5.

5.1.2 Streaming Hierarchical Video Segmentation

Xu et al. [85] proposed a method which segments the video stream $\mathbf{V}$ into a **hierarchical tree of partitions** $\mathfrak{H} = \{\mathfrak{H}_0, \mathfrak{H}_1, \ldots, \mathfrak{H}_L\}$. Here each layer $\mathfrak{H}_l$ is a partition of the whole video $\mathbf{V}$ and layers form a tree in the sense that each segment $S \in \mathfrak{H}_l$ has exactly one parent in the layer $\mathfrak{H}_{l+1}$ above. The first layer $\mathfrak{H}_0$ is the full partition where every spatio-temporal voxel is a segment on its own. The tree can have several roots as the top partition $\mathfrak{H}_L$ is not required to only have one element. Xu asks the question how the best partition under a objective function or criterion $E(\cdot|\cdot)$ can be found by minimizing

$$\mathfrak{H}^* = \operatorname*{argmin}_{\mathfrak{H}} E(\mathfrak{H}|\mathbf{V}) \tag{5.1}$$

At this point the **Markov assumption** is employed to make the optimization problem traceable. Towards this goal the video is divided into a sequences of non-overlapping subsequences, $\mathbf{V} = (\mathbf{V}^1, \mathbf{V}^2, \ldots, \mathbf{V}^m)$, and partitions are computed for each subsequence, $\mathfrak{H} = (\mathfrak{H}^1, \mathfrak{H}^2, \ldots, \mathfrak{H}^m)$. Thus it is assumed, that the partition of a subsequence $\mathfrak{H}^i$ does only depend on the last partition $\mathfrak{H}^{i-1}$ and the current and last video sequence $\mathbf{V}^i$ and $\mathbf{V}^{i-1}$. This allows to decompose the problem:

$$E(\mathfrak{H}|\mathbf{V}) = E''(\mathfrak{H}^1|\mathbf{V}^1) + \sum_{i=2}^{m} E'(\mathfrak{H}^i|\mathbf{V}^i, \mathbf{V}^{i-1}, \mathfrak{H}^{i-1}) \,. \tag{5.2}$$

where E' is a sequential segmentation model and E'' is the initial segmentation model. The Markov assumption has two advantages: On the one hand, only the current and last video sequence have to be stored and considered by the algorithm, thus saving storage and computation resources. On the other hand, the method works for video streams where the whole sequence is not known in advance.

In order to estimate $E'(\mathfrak{H}^i|\mathbf{V}^i, \mathbf{V}^{i-1}, \mathfrak{H}^{i-1})$ a similar assumption is made by assuming that the segmentation of a partition layer $\mathfrak{H}^i_l \in \mathfrak{H}^i$ only depends on the underlying partition layer $\mathfrak{H}^i_{l-1}$, the corresponding layers in the last timeslot $\mathfrak{H}^{i-1}_l$ and $\mathfrak{H}^{i-1}_{l-1}$, and the current video sequence. Thus

$$E'(\mathfrak{H}^i_l|\mathbf{V}^i, \mathbf{V}^{i-1}, \mathfrak{H}^{i-1}) = E'(\mathfrak{H}^i_l|\mathbf{V}^i, \mathbf{V}^{i-1}, \mathfrak{H}^{i-1}_l, \mathfrak{H}^i_{l-1}, \mathfrak{H}^{i-1}_{l-1}) \tag{5.3}$$

In order to find the best partition $\mathfrak{H}^i_l$ which optimizes the objective E', a general **semi-supervised grouping method** is developed which builds on top of an unsupervised, iteratively grouping method. The unsupervised grouping method is required to construct segmentation results by merging segments of the previous layer, $\mathfrak{H}^i_{l-1}$ and $\mathfrak{H}^{i-1}_{l-1}$, into larger segments for the current layer $\mathfrak{H}^i_l$. The semi-supervised extension supervises this grouping process by additionally using the segmentation information from the previous sequence $\mathfrak{H}^{i-1}_{l-1}$ as supervised information. This supervision is realized by an additional merging criteria, which checks if two small segments belong to different supervised segments in the previous time slot, and in this case rejects the merging. This additional merging criteria is explained in more detail in [85].

5.2 Temporal-Stable Adaptive Superpixels

5.2.1 Boundary instability in *Adaptive Superpixels*

Adaptive Superpixels and *Depth-Adaptive Superpixels* partition an image into uniformly distributed superpixels which contain pixels with similar feature values. Superpixel borders respect image borders by considering the feature map $\mathbf{I} : \mathcal{G}_I \rightarrow \mathcal{F}$ on the pixel lattice $\mathcal{G}_I$ (see def. 7). However as superpixels are an over-segmentation of the image, superpixel borders also occur naturally in regions of the image where feature values are similar. Here, the shape of superpixels are close to a Voronoi-Diagram as the feature dependent term in the density-adaptive metric is small compared to the compactness term. Thus superpixel boundaries

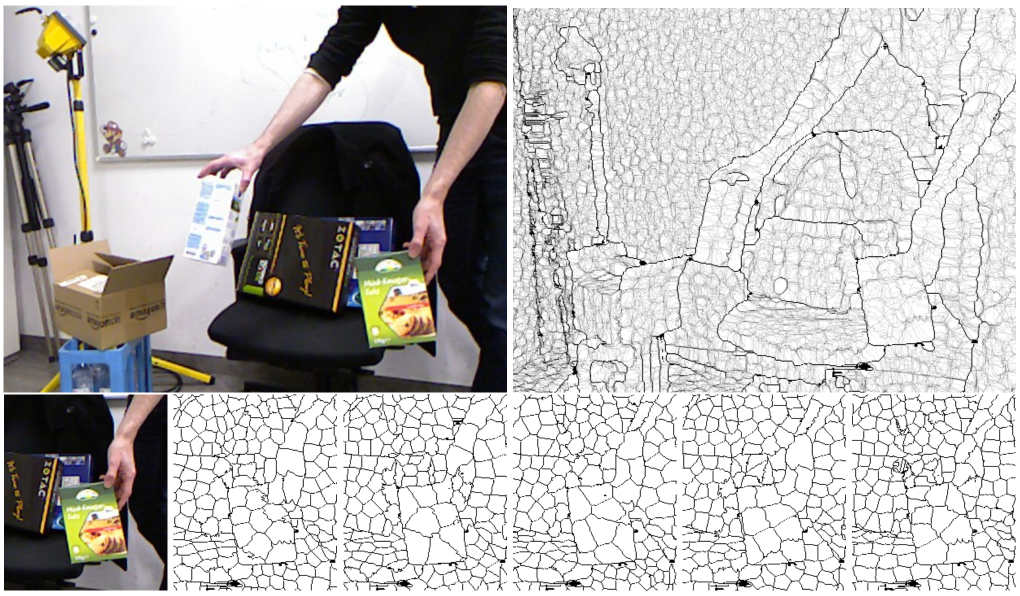


Figure 5.4: **Top left**: Colour part of the RGB-D input image. **Top right**: Mean image of border images for 12 possible *DASP* superpixel oversegmentations. **Bottom**: Detail of the full colour image and a set of different but equally likely *DASP* oversegmentations.

are mostly random and depend only on the superpixel centre position. Theses positions are mainly influenced by initial superpixel seeds computed by the Poisson disc sampling process, and change only during *Density-Adaptive Local Iterative Clustering* to form an uniform distribution. Fig. 5.4 shows this behaviour by displaying equivalent but different superpixel segmentations for the same input image. Additionally a mean image of superpixel borders is shown which demonstrates that only the actual image boundaries are stable throughout repeated oversegmentation of the same image.

For static images there is no need to further influence the choice of theses boundaries. This changes however when the algorithm is applied on consecutive frames of a video stream. For a sequence of images which change only slightly over time, it is desirable to fix segments to a certain local region as long as there is no change in the superpixel distribution. This has the advantage, that it is much easier to match individual superpixel from one frame to the next. When superpixel placement would change for each frame, the matching would be in general a many-to-many matching. For stable superpixels however the temporal matching is in most cases one-to-one.

Towards the goal of stable superpixels, I present an extension of *Adaptive Superpixels*: *Temporal-Stable Adaptive Superpixels* (*Stable ASP*) . *Stable ASP* uses a given superpixel oversegmentation to compute a new superpixel segmentation for an image with similar feature values. To allow as much freedom as necessary for the superpixel segmentation, the only step which is changed in *Stable ASP* with respect to *ASP* is the initial clustering of superpixels. Instead of a purely random Poisson disc sampling, a conditioned Poisson disc sampling method called *Delta Density Sampling* (*DDS*) is introduced, which builds upon an already given Poisson disc point distribution and only changes points if the density function changes. *DDS* uses the previously introduced *Simplified Poisson Disk Sampling* method to compute a stable Poisson disc point distribution. As *Depth-Adaptive Superpixels* is an instance of *Adaptive Superpixels*, the same technique can be applied directly to *Depth-Adaptive Superpixels*.

5.2.2 *Delta Density Sampling*

Delta Density Sampling tries to stabilize the sampling process by using an existing point sampling $\{{u_i}^{(k-1)}\}$ for the density function $\rho^{(k-1)}$ from the previous timestep to satisfy the target density $\rho^{(k)}$ for the current timestep. The resulting point sampling $\{{u_i}^{(k)}\}$ should have similar properties as if sampled directly from the current target density while additionally being as close as possible to the point sampling $\{{u_i}^{(k-1)}\}$ from the previous frame. In other words, *DDS* tries to exploit the freedom of choice for Poisson disc point sampling to stabilize the sampling process.

First the delta density $\Delta\rho$ is computed as the required change in density which is necessary to satisfy the new density function:

$$\Delta\rho = \rho^{(k)} - A_{\rho^{(k)}}(\cdot \mid \{{u_i}^{(k-1)}\})\,. \tag{5.4}$$

Here the density actually realized by the provided point samples after eq. 2.5 is used as the source density as it represents the density which is currently realized. The delta density $\Delta\rho$ is in general both positive and negative and can not be used for sampling directly. Thus $\Delta\rho$ is separated into its positive part ρ_+ and its negative part ρ_- computed as

$$\rho_+ = \frac{1}{2}\left(|\Delta\rho| + \Delta\rho\right) \tag{5.5}$$

$$\rho_- = \frac{1}{2}\left(|\Delta\rho| - \Delta\rho\right) \tag{5.6}$$

ρ_+ and ρ_- satisfy

$$\Delta\rho = \rho_+ - \rho_- \text{ and } \rho_+, \rho_- > 0\,. \tag{5.7}$$

and can thus be used in a sampling process. ρ_+ indicates that points need to be added as the density is increasing and ρ_- indicates the points need to be removed as the density is decreasing.

These two components are handled by *DDS* independently as follows (see fig. 5.5): First, ρ_- is used in a normal *Simplified Poisson Disk Sampling* process to sample a set of locations where the number of points shall be reduced. For each sampled point, the original point set is searched and the spatially nearest point is removed. Second, ρ_+ is used in a second *SPDS* process to sample a new set of

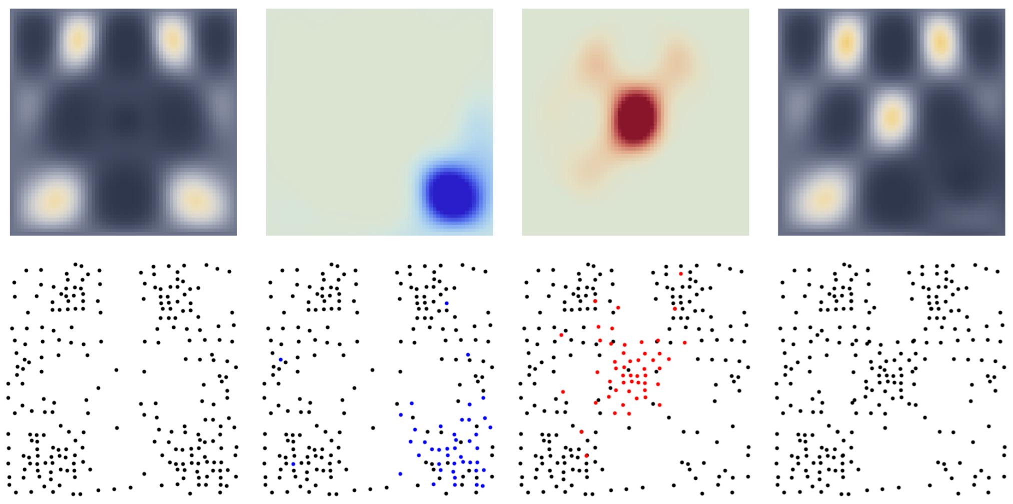


Figure 5.5: **Top from left to right:** Point density $A_\rho(\cdot\,|\,\{u_i\})$ from previous frame, components ρ_- and ρ_+ of the delta density $\Delta\rho$, and target density ρ. **Bottom from left to right:** Point samples in subsequent stages of *Delta Density Sampling*. Blue points are removed and red points are added.

points for areas where the point density has increased. Theses points are simply added to the original and reduced point set. The complete algorithm is given in alg. 5.

The processes of adding and removing points are additive with respect to the original sampling and thus do not preserve the Poisson disc property of the sampling as the concrete placement of the original points is not considered. This can for example result in points being placed to close to already placed points. However this issue is not investigated further in this context, as normally only few points are remove or added and *DALIC* further improves the distribution quality.

5.2.3 Streaming Adaptive Superpixel

Delta Density Sampling can be used in *Adaptive Superpixels* or *Depth-Adaptive Superpixels* to provide temporal-stable superpixels, and thus a streaming superpixel computation for a stream of image feature annotation $\mathbf{I}^{(k)}$ and a stream of superpixel densities $\rho^{(k)}$. The procedure is straightforward and summarized in alg. 6.

For the first timestep *Simplified Poisson Disk Sampling* (see §2.3.3) is used to sample a set of initial superpixel clusters ${U_0}^{(1)}$ from the corresponding initial density function $\rho^{(1)}$. Then *Density-Adaptive Local Iterative Clustering* (see §2.4) assigns pixel to superpixels. For the following timesteps, the set of final superpixels centres ${U_*}^{(k-1)}$ after *DALIC* from the previous timestep $t-1$ are used to seed the sampling of superpixel clusters for the next timestep t using *Delta Density Sampling*. Here, *SPDS* is used again as the underlying sampling method for *DDS*.

Algorithm 5 *Delta Density Sampling* (*DDS*)

```
Require: Target density $\rho$
Require: Set of points $U$
Require: Poisson Disc Sampling Process PDS, e.g. SPDS
  ▷ Compute delta density
  $\Delta\rho = \rho - A_\rho(\cdot \mid U)$
  $\rho_- = \frac{1}{2}\left(|\Delta\rho| - \Delta\rho\right)$
  $\rho_+ = \frac{1}{2}\left(|\Delta\rho| + \Delta\rho\right)$
  ▷ Remove points
  $U_- = PDS(\rho_-)$
  $U' = U$
  for $u \in U_-$ do
      $U' = U' - \{\operatorname{argmin}_{x\in U} \|x - u\|\}$
  end for
  ▷ Add points
  $U_+ = PDS(\rho_+)$
  return $U' \cup U_+$
```

As for the first timestep, *DALIC* is used to compute superpixels and thus the final locations of superpixel centres.

Algorithm 6 *Streaming Adaptive Superpixel* (*Stream-ASP*)

```
Require: Video stream $\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots$ and stream of density functions $\rho^{(1)}, \rho^{(2)}, \ldots$
  ${U_0}^{(1)} = \mathrm{SPDS}(\rho^{(1)})$
  ${U_*}^{(1)} = \mathrm{DALIC}(\rho^{(1)}, \mathbf{I}^{(1)}, {U_0}^{(1)})$
  for $t = 2, \ldots$ do
      ${U_0}^{(k)} = \mathrm{DDS}(\rho^{(k)}, {U_*}^{(k-1)}, \mathrm{SPDS})$
      ${U_*}^{(k)} = \mathrm{DALIC}(\rho^{(k)}, \mathbf{I}^{(k)}, {U_0}^{(k)})$
  end for
```

As *Delta Density Sampling* is a probabilistic process, the benefits are analysed quantitatively over a full video stream. For each cluster the minimal distance in screen coordinates to clusters from the previous frame is computed and normalized with respect to the cluster radius. The measured distances are shown in a histogram for all clusters from all frames in fig. 5.6. The difference between normal *SPDS* sampling, which does not consider the distribution from the previous frame, and *DDS* which does consider it, are clearly visible. *DDS* shows a much higher probability that clusters move only for a small distance. Note that due to movements in the scene and the probabilistic nature of *DDS* it is unlikely that clusters do not move at all.

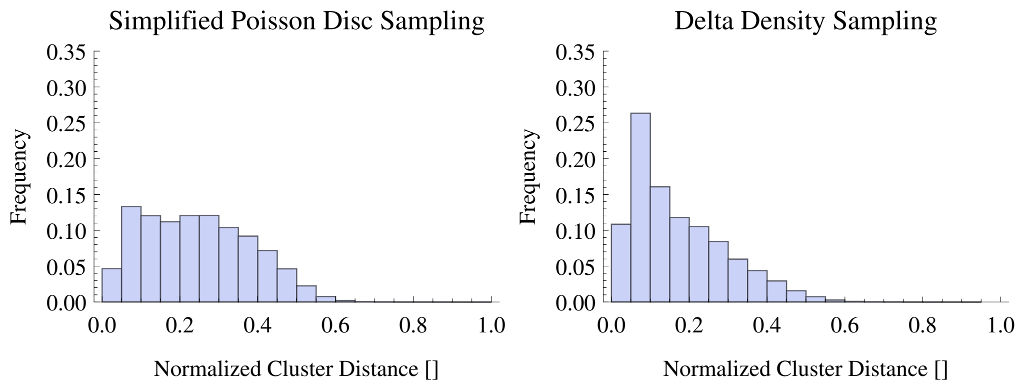


Figure 5.6: Distribution of minimal cluster distance from frame to frame for a test sequence. Distance is measured in screen coordinates and normalized with respect to the cluster radius.

5.3 Tempo-spatial strands and strand graph

5.3.1 Spatio-temporal supervoxels

The temporal domain can be used in different ways in video segmentation. A primitive approach would be the segmentation of each frame independently from the previous frames. Temporal connections would be established afterwards between finished segments. While this approach is simple and allows to immediately use existing image segmentation techniques, it yields poor results as the segmentations are not smooth and the assignment of superpixels between frames is a difficult or even impossible problem. Another example would be to consider a video stream as a set of spatio-temporal voxels and try to build spatio-temporal supervoxels. Here temporal connections are treated like spatial connections and segmentation techniques could be employed on the full voxel grid. However this approach uses time the same way as space, which may be a debatable assumption, and is not followed further in this context. An approach which was already introduced in §5.1.2 segments each frame individually, but uses the segmentation from the previous timestep as a supervision guideline.

Temporal Depth-Adaptive Superpixels presented in this chapter uses a hybrid approach which has the advantage of using several frames for the segmentation process as well as using temporal connections during segmentation for supervision. *t-DASP* consists of three steps:

1. Oversegmentations for each RGB-D frame from the video stream are computed using the *Temporal-stable Depth-Adaptive Superpixel* algorithm from §5.2.

2. Temporal connections between superpixels are established to form temporal strands of superpixels. At the same time a weighted graph structure on

the strands is constructed and updated which represents similarity between strands.

3. Superpixels strands are grouped into segments using spectral graph theory.

In order to build spatio-temporal strands of superpixels, temporal connections between the superpixels from the previous frame and the current frame have to be established. Ideally the connections should be formed only between superpixels which represent the exact same piece of surface. In general this is not possible for a dynamically changing environment: As the camera or objects move, parts of the surface geometry may become occluded or move outside of the field of view. Additionally objects are not necessary rigid and can thus change their geometric shape. Due to the complexity of the problem, in this scope, only an approximate solution will be developed where temporal connections are only formed locally and additionally tested for plausibility using a simple check. The presented model does not use any semantic knowledge about scene geometry and does not require prior 3D models for the scene geometry. This has the disadvantage that the choice of temporal connections is only performed locally and may not capture all subtleties. However as models are not required, the method does not required training, is simple to set up and will work in completely unknown and unrestricted environments.

5.3.2 Temporal superpixel strands

The streaming superpixels algorithm from §5.2 provides good preconditions to establish such temporal connectivity between superpixels efficiently. Due to the gained stability of superpixel positions from frame to frame, it is much easier to find a direct assignment. When superpixels are sampled independently for each frame without a stabilizing procedure, the assignment problem is a many-to-many assignment problem which is often a hard or ill-posed problem. Due to the stabilization of superpixel positions, this problem can be reduced to an easier almost one-to-one problem.

Delta Density Sampling (see alg. 5) computes superpixel seed points for the current frame from superpixel centres from the previous frame by considering the change in superpixel density. During this process some seed points are deleted, many seed points are preserved and some new seed points are added. If the absolute delta density is low, there is a high probability that most seed points are preserved from frame to frame. For a changing density however seed points are added and removed accordingly. The correspondence between cluster centres and seed points induces a direct connectivity on superpixels. Here three cases are directly derived from the behaviour of *Delta Density Sampling*: a cluster does not have a predecessor, a cluster has exactly one predecessor or a cluster from the previous frame is not continued.

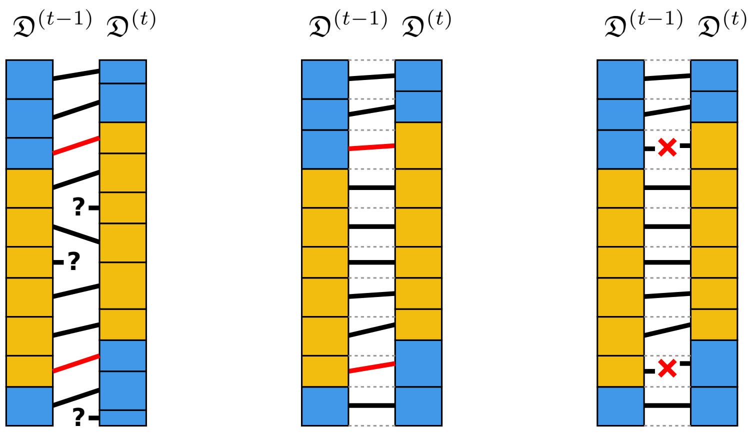

$\mathfrak{D}^{(t-1)}$ $\mathfrak{D}^{(t)}$
$\mathfrak{D}^{(t-1)}$ $\mathfrak{D}^{(t)}$
$\mathfrak{D}^{(t-1)}$ $\mathfrak{D}^{(t)}$
?
?
?

Figure 5.7: A simplified example with 1-dimensional superpixels from two consecutive frames (vertical stripes) connected with each other with temporal connections (black). Blue indicates background, and yellow a vertically moving foreground object. **Left:** Superpixels with *SPDS* yield a bad assignment with a typical greedy approach. **Middle:** Stable superpixels with *DDS* give a much better assignment. However some connections may not be correct (red) as they connect objects from different semantic groups. **Right:** Stable *DDS* superpixels with pruning of incorrect assignments.

Let superpixel partitions from the last and current frame be noted as $\mathfrak{D}^{(t-1)}$ and $\mathfrak{D}^{(t)}$. Each partition is a set of superpixels which is expressed as

$$\mathfrak{D}^{(t)} := \left\{ \mathfrak{s}_j^{(t)} \,|\, 1 \le j \le n^{(t)} \right\}. \tag{5.8}$$

The connectivity correspondence from *Delta Density Sampling* is noted as a mathematical relation as follows:

$$C_{\text{DDS}}^{(t)} := \left\{ (i,j) \,|\, \text{seed of superpixel } \mathfrak{s}_j^{(t)} \text{ was created from centre of superpixel } \mathfrak{s}_i^{(t-1)} \right\} \tag{5.9}$$

As noted before this relation is almost one-to-one in the sense that for each i there is at most one j such that $(i,j) \in C_{\text{DDS}}^{(t)}$ and for each j there is at most one i such that $(i,j) \in C_{\text{DDS}}^{(t)}$.

If the scene is more or less static and thus density, geometry and appearance are not changing much, there is a high change that superpixels are preserved, not moving much and still represent the same surface piece. In this case the relation $C_{\text{DDS}}^{(t)}$ induced from *Delta Density Sampling* connects the correct superpixels. However in dynamic scenes, several cases can occur which require a revision of the initial mapping:

Superpixel movement During *Density-Adaptive Local Iterative Clustering* superpixels can sometimes move quite a large distance depending on the number of iterations and values in the local feature space.

Superpixel reassignment When geometry moves quickly in parallel to the image plane while the superpixel pixel position stays constant, a superpixel can be reassigned to a new object which is not related to the object to which the superpixel was assigned before (see fig. 5.7).

To purge erroneous connections an additional test is executed on superpixel frame-to-frame connections in $C^{(t)}_{\text{DDS}}$. If a connection $(i, j) \in C^{(t)}_{\text{DDS}}$ fails the test, it is removed from the mapping set (fig. 5.7). In practice a simple threshold on the change in superpixel position and feature values (i.e. color and normal) has proven to be sufficient:

$$\text{If } d_\circ\left(\mathfrak{s}^{(t)}_{j,\circ}, \mathfrak{s}^{(t-1)}_{i,\circ}\right) \geq \theta_\circ \text{ for any } \circ \in \{c, n, v\} \text{ remove } (i, j) \text{ from } C^{(t)}_{\text{DDS}}. \tag{5.10}$$

The proposed method to establish almost one-to-one connections emerges directly from superpixels sampling and requires only a simple test on superpixel feature values. This provides a direct and fast method to solve the general problem of superpixel correspondence. However more advanced methods can easily be integrated into the general framework of *Temporal Depth-Adaptive Superpixels* by replacing the mapping $C^{(t)}_{\text{DDS}}$ accordingly.

By consecutively connecting new superpixels to superpixels from the previous frame **strands of superpixels $\mathbf{T}_k$** are formed. Depending on the superpixel mapping new strands are created, strands are continued or old strands are not continued. Superpixel strands may vary in length depending on the dynamic of the scene. The **length of a superpixel strand $\mathbf{T}_k$**, i.e. the number of superpixels which are chained up, is noted with l_k. The **starting time of a superpixel strand $\mathbf{T}_k$**, i.e. the frame index of its first superpixel, is noted with t_k. Thus the frame index of the last superpixel is $t_k + l_k - 1$. Additionally as strands which are continued for one frame are never picked up again, there may be many **inactive superpixel strands**, i.e. strands which do not contain a superpixel from the current frame. Thus a strand $\mathbf{T}_k$ is still active if $t_k + l_k - 1$ equals to the current frame index t.

Let $\mathfrak{D}^{(t)} = \{\mathfrak{s}^{(t)}_j \mid 1 \leq j \leq m\}$ be the current set of superpixels. For each superpixel $\mathfrak{s}^{(t)}_j$ there is a corresponding superpixel strand $\mathbf{T}_k$. If the sample point corresponding to the superpixel was added during *Delta Density Sampling*, the corresponding strand was just added and has length 1, otherwise the strand already has a length greater than 1. In general each strand $\mathbf{T}_k$ consists of a sequence of superpixels:

$$\mathbf{T}_k = \left(\mathfrak{s}^{(t_k)}_{j_1}, \ldots, \mathfrak{s}^{(t_k+l_k-1)}_{j_{l_k}}\right) \tag{5.11}$$

Where t_k is the starting frame of the strand and l_k the length of the strand. The indices $j_1, \ldots, j_{l_k}$ indicate the superpixel index with respect to the corresponding frame. Fig. 5.8 shows a schematic drawing of superpixel strands. For a clearer presentation 1-dimensional superpixels are displayed instead of the normal two-dimensional superpixels. Strands are broken when temporal connections are pruned due to eq. 5.10 or when superpixel are deleted during *Delta Density Sampling* (not displayed).

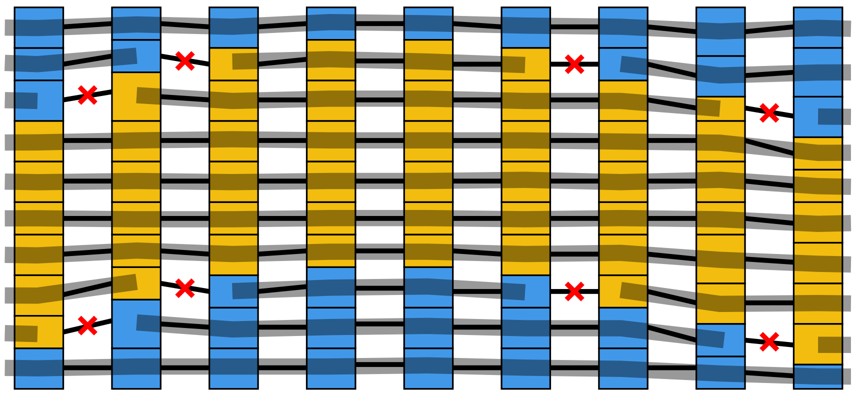


Figure 5.8: 1-dimensional superpixels from consecutive frames (vertical stripes) are connected with each other with temporal connections (black lines). Red crosses mark pruned connections. Blue indicates background, and yellow a vertically moving foreground object. Fat lines overlayed in Grey indicate superpixel strands which connect many superpixels over time.

Note that the mapping from superpixels to strands is not surjective as strands are "closed" when the corresponding superpixel point is removed during *Delta Density Sampling* or if the strand can not be continued with respect to eq. 5.10. This correspondence mapping from superpixel to strands is noted as $\mu^{(t)}_{\mathfrak{s}\mathbf{T}}$. Thus $\mathbf{T}_{\mu^{(t)}_{\mathfrak{s}\mathbf{T}}(j)}$ is the strand corresponding to superpixel $\mathfrak{s}^{(t)}_j$ and regarding the indices in equation eq. 5.11 we have $\mu^{(t_k+q)}_{\mathfrak{s}\mathbf{T}}(j_q) = k$.

5.3.3 Superpixel strand graph

Superpixel strands form a partition of the spatio-temporal graph $\mathcal{G}_V$: Each superpixel is assigned to exactly one strand and superpixels themselves are a spatial partition for each frame. In this section a graph structure on superpixel strands will be constructed which can then be used for segmentation. The **superpixel strand graph** $\mathcal{G}_\mathbf{T}$ consists of a set of vertices $V(\mathcal{G}_\mathbf{T})$ which is formed by the superpixel strands $\mathbf{T}_k$ constructed from depth-adaptive superpixels as explained in the previous section. Edges $E(\mathcal{G}_\mathbf{T})$ in the graph are derived from the frame-wise superpixel graphs as explained in the following.

Remember that a superpixel neighbourhood graph structure $\mathcal{G}^{(t)}_S$ can be build on superpixels $\mathfrak{D}^{(t)}$ by connecting superpixels which are adjacent in the pixel grid as explained in §4.3.1. Weights $W_\mathfrak{s}$ for graph edges in the superpixel neighbourhood graph can be computed using a similarity measure like in eq. 4.18. To derive a graph structure on the superpixel strand graph $\mathcal{G}_\mathbf{T}$, the connectivity and similarity measure from these superpixel graphs $\mathcal{G}^{(t)}_S$ should be taken into account.

The straight-forward way to derive edge connectivity is to connect two strands if there are two superpixels in the same time slice t which are connected in the superpixel graph $\mathscr{G}_S^{(t)}$. The weight of an edge between two superpixel strands is then computed by averaging the weight of all superpixel edge connections. This gives the following equation for edge weights $W_{\mathrm{T}}(p,q)$ between two superpixel strands $\mathbf{T}_p$ and $\mathbf{T}_q$:

$$W_{\mathfrak{s}}(p,q) = \begin{cases} \frac{1}{|t_{pq}|} \sum_{t \in t_{pq}} W_{\mathfrak{s}}^{(t)} \left({\mu_{\mathfrak{s}\mathbf{T}}^{(t)}}^{-1}(p), {\mu_{\mathfrak{s}\mathbf{T}}^{(t)}}^{-1}(q) \right) & \text{if } (p,q) \in E(\mathscr{G}_{\mathbf{T}}) \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

Here $t_{pq} = \left[\max(t_p, t_q) \,|\, \min(t_p + l_p, t_q + l_q) - 1\right]$ is the time interval where both strands have a superpixel. t_{pq} is simply the intersection of the time frame intervals $\left[t_p, t_p + l_p - 1\right]$ and $\left[t_q, t_q + l_q - 1\right]$ of the two superpixel strands.

Note that the computation of edge weights $W_{\mathbf{T}}$ after eq. 5.12 already takes into account the maximal possible number of connections $|t_{pq}|$ due to the overlap of the superpixel strand time intervals. The corresponding superpixels ${\mu_{\mathfrak{s}\mathbf{T}}^{(t)}}^{-1}(p)$ and ${\mu_{\mathfrak{s}\mathbf{T}}^{(t)}}^{-1}(q)$ at a given time index $t \in t_{pq}$ in the common time interval t_{pq} are not necessarily connected in $\mathscr{G}_S$. It may well be the case that superpixel connections only exists for some frames, but due to the definition of the set of edges $E(\mathscr{G}_{\mathbf{T}})$ this must be the case for at least one frame. If no such connections exists for a given time frame it is assumed that the contribution to $W_{\mathbf{T}}$ is 0, which is also reflected in the definition of superpixel edge weights $W_{\mathfrak{s}}$ (see eq. 4.18).

Due to the varying length of superpixel strands, the length of the common time interval $|t_{pq}|$ may vary. This has the effect that edge weights for edges based on a short common time interval have a lower confidence than those where the common time interval is longer. For simplicity no further measure is taken to balance this effect.

The superpixel strand graph is an temporal-spatial augmentation of the spatial superpixel graphs which in addition to spatial relations also captures temporal relation. However for superpixel strands temporal relations are modeled in a different way than spatial relations. Instead of treating time equal to space as it would be the case for generic supervoxels, a kind of local tracking is employed where each superpixel tries to track itself over time. The combination of many such local tracking results are collected in the superpixel strand graph. In the following section the superpixel strand graph is segmented with the usual spectral graph segmentation framework to collect superpixel strands into segments and compute and assign temporal-stable labels to segments.

5.4 Streaming graph segmentation

The superpixel strand graph $\mathscr{G}_{\mathbf{T}}$ is a segmentation of the spatio-temporal video stream voxel graph $\mathscr{G}_V$ which already greatly reduces the number of vertices by grouping similar voxels into superpixel strands. But in analogy to the segmentation of the superpixel graph into high-level superpixel segments in §4, a high-level segmentation of the superpixel strand graph into high-level superpixel strand segments is required for further analysis. In this section the methods of spectral graph theory (see §4.2) will be used again to compute high-level partitions of the superpixel strand graph. In general, any graph segmentation framework which can handle non-regular graphs could be used for computing superpixel strand partitions.

Previously in §4, the superpixel graph $\mathscr{G}_S$ was computed for one static RGB-D image at a time. In contrast to this one-time segmentation of the superpixel graph, the segmentation of the superpixel strand graph $\mathscr{G}_{\mathbf{T}}^{(t)}$ needs to be repeated for each new timestep t as the strand graph is extended with each new frame from the video stream. An important differentiation has to be made: One can consider a finite video where all frames are known at the time of analysis. In this case the superpixel strand graph need to be constructed only once from superpixels of all frames, superpixel strands are computed only once and the segmentation of the superpixel strand graph is computed once. On the other side, one can consider a video stream where only frames in the past are known and it is unknown how and for how long the video will continue. In this case the superpixel strand graph is continuously extended with each new frame and segmentations need to be computed for each new frame.

The computation of superpixel strand segments for video streams poses three problems which are discussed in the following:

1. Limiting the size of the superpixel strand graph by focusing on recent frames to avoid huge memory requirements.

2. Using global methods to improve detection of spatio-temporal segment boundaries.

3. Continuous segment labels from one frame to the next.

5.4.1 Adaptive reduction of the strand graph

First, the size of the superpixel strand graph needs to be reduced to focus on recent frames to follow the spirit of stream segmentation and to avoid huge memory consumption. Additionally, a downside of spectral graph theory is the time-consuming computation of eigenvalues and eigenvectors. Although only the lowest eigenvalues are required, the process still has a theoretical runtime of $O(n^{\frac{4}{3}})$. For superpixel segmentation the number of superpixels could be controlled directly and a reasonable number of superpixels around 1000 is enough for most image analysis and segmentation purposes. However in the case of spatio-temporal superpixel strands the number of strands grows with time. For the used dataset experiments have shown that the average length of superpixel strands varies from 10 to 40 superpixels, i.e. frames. This indicates that the number of superpixel strands increases very fast over time: Using 1000 superpixels and assuming an average strand length of 30, after one minute of video there would be already 60000 strands in the graph which would exceed a reasonable number of nodes for spectral graph theory to run in realtime applications.

To solve this issue an approximate solution was chosen where old strands are deleted from the superpixel graph $\mathscr{G}_{\mathbf{T}}$ to produce a "reduced" superpixel strand graph $\mathscr{G}^*_{\mathbf{T}}$. This procedure highlights the fact that a realtime video stream segmentation should put the most interest on the current time frame. Information from past time frames is used in the sense of improving the segmentation of the current timeframe. Strands are deleted from the strand graph in a simple procedure where strands with the lowest "end time", i.e. $l_l + t_l - 1$ are deleted until only a predefined number of maximal strands are left. The maximal number of strands must be greater than the number of superpixels to guarantee that each superpixel is represented by a strand.

For a low number of strands only superpixels which are connected via a strand to the current superpixels are considered, and depending on the dynamic of the scene, and thus the average length of superpixels, strands may be short. In this case, the tempo-spatial strand segmentation will have a very short "memory" and will be similar to a non-temporal segmentation which considers only the current superpixel graph. When more strands are considered or the strand length is longer in average, the tempo-spatial segmentation can include more and more information in the segmentation process. This can be essential to resolve occlusions or to guarantee a more stable segmentation in ambiguous situations.

5.4.2 Globalization of strand graph weights

Edge weights $W_{\mathbf{T}}$ express local similarity of superpixel strands. As explained in §4, local boundary detectors suffer from a couple of issues, and in general, provide poor performance for the detection of high-level segments. Here the globalization method described in §4.3.2 on the superpixel graph is re-used and employed on the superpixel strand graph. Due to the reduction explained in the previous section, the superpixel strand graph $\mathscr{G}^*_{\mathbf{T}}$ is sufficiently small to be analysed with spectral graph theory.

Using the reduced superpixel strand graph $\mathscr{G}^*_{\mathbf{T}}$ with weight matrix $W^*_{\mathbf{T}}$ the generalized eigenvalue problem eq. 4.11 is solved:

$$L(\mathscr{G}^*_{\mathbf{T}})\,x = \lambda\,D(\mathscr{G}^*_{\mathbf{T}})\,x \tag{5.13}$$

with the diagonal matrix

$$D(\mathscr{G}^*_{\mathbf{T}})_i = \sum_j W^*_{\mathbf{T}}(i,j) \tag{5.14}$$

after eq. 4.4 and the Laplacian

$$L(\mathscr{G}^*_{\mathbf{T}}) = D(\mathscr{G}^*_{\mathbf{T}}) - W^*_{\mathbf{T}} \tag{5.15}$$

after eq. 4.5. As explained in §4.2.3 the generalized eigenvalue problem can be transformed into a standard eigenvalue problem under certain conditions. For the sake of spectral graph theory only smallest eigenvalues and corresponding eigenvectors are required, which simplifies the algorithmic computation task as specialized and thus faster solvers can be chosen.

In analogy to eq. 4.22, new graph weights are derived from eigenvalues λ_k and eigenvectors v_k:

$$W_{\mathbf{T}}^{\text{global}}(i,j) := \begin{cases} \sum_k \frac{1}{\sqrt{\lambda_k}} |v_{ki} - v_{kj}| & \text{if } (i,j) \in E(\mathscr{G}_{\mathbf{T}}^{*}) \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

These new graph weights take into account global information over the whole graph. They form a globalization of the local boundary descriptor which only takes into account direct strand-to-strand similarity.

The next step to compute a segmentation on the superpixel strand graph is to derive segments using the global weights $W_{\mathbf{T}}^{\text{global}}$. In analogy to §4, the weights $W_{\mathbf{T}}^{\text{global}}$ induce an ultrametric contour graph (see §4.3.2) on the superpixel strand graph. The graph is formed by using superpixel pixel boundaries to compute superpixel strand voxel boundaries. Spatial voxel boundaries can be copied directly from frame superpixel boundaries. Temporal voxel boundaries are established between superpixels of adjacent frames if they do not belong to the same superpixel strand.

5.4.3 Label propagation

The superpixel strand graph could be segmented in analogy to the static case by thresholding the ultrametric contour graph derived from the weights $W_{\mathbf{T}}^{\text{global}}$. However at this point an issues arises, because such an independent segmentation does not consider the segmentation of the previous timestep and thus does not provide stable segments. Theoretically, a segmentation of the previous strand graph $\mathscr{G}_{\mathbf{T}}^{(t-1)}$ could be quite different from a segmentation of the graph $\mathscr{G}_{\mathbf{T}}^{(t)}$ which additionally considers the current video frame and which may have forgotten some strands in the past. This can be due to the appearance or disappearance of geometry from the field of view, due to the change of occlusions or other phenomena.

StreamGBH [85] solves this problem by adapting the process in which segments are computed from a ultrametric contour graph. A semi-supervised method is used where a segmentation from the previous timestep supervises the segmentation of the current timestep (see §5.1.2). For the segmentation of superpixel strands a similar semi-supervised method can be used. To show the analogy to StreamGBH, the current segmentation process can be interpreted as a three layer segmentation: The base layer $\mathfrak{H}_0$ is the layer of tempo-spatial voxels where each voxel is in its own segment. The first segmentation layer $\mathfrak{H}_1$ is the segmentation of voxels into superpixel strands as explained in the previous chapter. The second layer $\mathfrak{H}_2$ is a segmentation of superpixel strands which is of interest here and discussed in the following. Later in this section the segmentation process is adapted to the special properties of *Temporal Depth-Adaptive Superpixels*.

5.4.4 Semi-supervised labelling for *t-DASP*

For simplicity a different notation for segmentations will be used now: Let $\Gamma : \mathscr{G}_{\mathbf{T}} \to \mathbb{N}$ be a function which assigns an integer as a label to each superpixel strand. Thus for a superpixel strand $\mathbf{T} \in \mathscr{G}_{\mathbf{T}}$ the integer $\Gamma(\mathbf{T})$ indicates the segment to which the strand is assigned. By defining that two strands are in the same segment if and only if they are assigned the same label, Γ directly fulfils the requirements for a partition.

In contrast to StreamGBH, superpixel strands are mostly consistent over several timesteps, thus segmentation labels $\Gamma^{(t-1)}$ for $\mathscr{G}_{\mathbf{T}}^{(t-1)}$ can be directly transferred to $\mathscr{G}_{\mathbf{T}}^{(t)}$. Each superpixel strand $\mathbf{T} \in \mathscr{G}_{\mathbf{T}}^{(t-1)} \cap \mathscr{G}_{\mathbf{T}}^{(t)}$ which has been continued from the previous timestep, has been labelled in the previous timestep with a label $\Gamma^{(t-1)}(\mathbf{T})$ indicating to which superpixel strand segment the superpixel strand was assigned previously. This strand labelling $\Gamma^{(t-1)}$ is considered to be information which is used to supervise the strand labelling $\Gamma^{(t)}$ of the current frame. If a strand has only been added in this timestep, or if no prior labelling exists, as it is the case for the first timestep, the strand is assigned a new label. Theses strands are considered to be not supervised. It is important to keep track which labels are supervised and which are not: Let

$$U^{(t)} := \max(\{\Gamma^{(t-1)}(\mathbf{T}) \mid \mathbf{T} \in \mathscr{G}_{\mathbf{T}}^{(t-1)} \cap \mathscr{G}_{\mathbf{T}}^{(t)}\}) \tag{5.17}$$

the maximal label integer for all supervised strands. The transfer of information between timesteps is indicated by forming an initial labelling of superpixel strands $\Gamma_0^{(t)}$ defined as:

$$\Gamma_0^{(t)} : \mathscr{G}_{\mathbf{T}}^{(t)} \to \mathbb{N}, \mathbf{T} \mapsto \begin{cases} \Gamma^{(t-1)}(\mathbf{T}) & \text{if } \mathbf{T} \in \mathscr{G}_{\mathbf{T}}^{(t-1)} \cap \mathscr{G}_{\mathbf{T}}^{(t-1)} \text{ (supervised)} \\ \text{a new integer } > U^{(t)} & \text{otherwise (not supervised)} \end{cases} \tag{5.18}$$

The semi-supervised segmentation method now iteratively joins segments depending on edge weight and labels and thus builds a tree of segmentations $\Gamma_i^{(t)}$. The method proceeds as follows: Edges in the graph $\mathscr{G}_{\mathbf{T}}^{(t)}$ are sorted by globalized edge weight $W_{\mathbf{T}}^{\text{global}}$ (see eq. 5.16) and processed in ascending order. Weights $W_{\mathbf{T}}^{\text{global}}$ represent a distance measure, so edges which connect the most similar segments are processed first. We assume a edge has been picked which connects the superpixel strands $\mathbf{T}_p$ and $\mathbf{T}_q$ and the corresponding segment labels are thus $a_p := \Gamma_i^{(t)}(\mathbf{T}_p)$ and $a_q := \Gamma_i^{(t)}(\mathbf{T}_q)$. Based on a_p and a_q, labels are either joined or not joined:

$$\text{labels } a_p \text{ and } a_q \text{ are joined iff. } a_p > U^{(t)} \vee a_q > U^{(t)}, \tag{5.19}$$

The joining rule prevents the merging of supervised labels which should not be allowed as supervised information shall not be altered. If labels are joined, a new labelling $\Gamma_{i+1}^{(t)}$ is constructed by replacing all occurrences of the bigger label index with the smaller label index. This guarantees that the supervised label is preserved in the case that a supervised and an unsupervised label are joined.

5.4.5 Hysteresis based label propagation

The proposed labelling method so far has been a direct adaption of the semi--supervised segmentation method presented in StreamGBH. In the context of streaming segmentation the method was originally also responsible for establishing temporal connections between frames. For *Temporal Depth-Adaptive Superpixels* this task is already taken over in a more direct way by building superpixel strands which greatly simplifies the problem of label continuation. Experimental evaluation has shown that for *Temporal Depth-Adaptive Superpixels* the label joining mechanism is too restrictive and can be further improved. The main segmentation process has already been carried out by spectral graph theory and is expressed in the globalized edge weights $W_{\mathbf{T}}^{\text{global}}$. The only task remaining is a reasonable continuation of labels and the assignment of new superpixel strands. Especially for the latter it is problematic if short strands are assigned to a segment without the possibility to revoke that decision once more information is available, i.e. the strand has grown longer. In general due to the Markov assumption made by StreamGBH, the process has a very short ”memory” which does not allow to change poor labelling decisions made in the past. Superpixel strands provide a kind of mid-term memory, as the information of several frames are used to compute segmentations. In this scenario it is beneficial to allow that labels are changed based on new information.

Towards this goal, *Semi-Supervised Hysteresis Segmentation* is presented: A hysteresis mechanism is added into the labelling method where the weight used to decide if segments should be joined is based on their current label. Additionally it is allowed to join supervised segments when the connecting weight is low enough or to split them if no further connections are present. *Semi-Supervised Hysteresis Segmentation* proceeds as follows:

1. Start with a graph G with nodes from $\mathcal{G}_{\mathbf{T}}^{(t)}$ and no edges.

2. Sort edges in $\mathcal{G}_{\mathbf{T}}^{(t)}$ by weight w, process all edges in ascending order and apply the following rules:

   - ”Forced merge”: Add edge to G if $w \leq t_{\text{force}}$.
   - Normal merge: Add edge to G if $w \leq t_{\text{normal}}$ and if not both strand labels are supervised (eq. 5.19).
   - ”Granted merge”: Add edge to G if $w \leq t_{\text{grant}}$ and both labels are supervised and identical.

3. Compute connected components of G.

4. Label components sorted by size in descending order:

    - Count how often each label appears in the component and find the most frequent label
    - If the label has not been assigned to a component yet assign it to the component,
    - else create a new label for the component.

For the joining process, *Semi-Supervised Hysteresis Segmentation* uses three thresholds $t_{\text{force}} < t_{\text{normal}} < t_{\text{grant}}$ when merging edges which are used depending on the level of confidence that labels represent the same segment based on supervised information. The three levels of confidence are:

1. Both labels are supervised and have different labels (low level of confidence)

2. One or both labels are not supervised (medium level of confidence)

3. Both labels are supervised and have the same label (high level of confidence)

5.5 Evaluation

Video segmentation algorithms are not easy to evaluate as most metrics require ground truth annotations for every video frame which requires a tremendous amount of work to get significant results. Thus for this evaluation three metrics are measured which do not require ground truth annotations: explained colour variation, explained depth variation and segment compactness. In addition, a qualitative comparison to the method of Xu et al. [85] is presented.

Explained variation (see §A.1) measures how much of the deviation of individual pixels is explained by the mean value of the whole segment and compactness (see §A.1) measures if segments are shaped like a disk or elongated and scattered over the whole image. As *t-DASP* consists of a two-layer segmentation process, metrics can be evaluated for both the superpixel and the superpixel segment label. Here superpixels and superpxiel segments are evaluated individually for every frame of the video sequence.

As the size of segments plays a crucial role in the quality of superpixels as it is much easier for small superpixels to give good results, it is expected that the quality of superpixel segments is lower than the quality of superpixels. Fig. 5.9 shows results as the mean of the respective metric over a dataset of video sequences. The figure shows results for a varying number of superpixels of 800, 1000 and 1200 (red line) and it is visible how the quality of superpixels increase with smaller superpixels. The blue line indicates results for superpixel segments for a varying

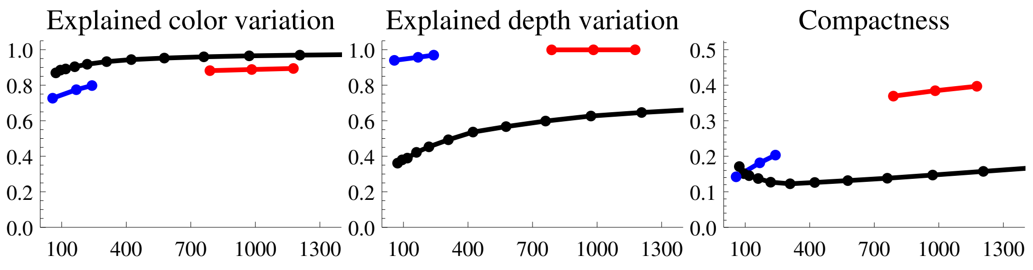


Figure 5.9: Evaluation of three metrics for StreamGBH (black) and T-DASP (red: DASP superpixels, blue: final segments) against the average number of clusters used per video frame. **Left:** Explained variation for colour values. **Middle:** Explained variation for depth values. **Right:** Compactness of segments measured with the isoperimetric quotient.

threshold for cutting the ultrametric contour graph on superpixel strands. A lower threshold gives more segments and thus again better results.

As a comparison towards state-of-the-art, additional results are reported for the StreamGBH video segmentation method of Xu et al. which does not use depth information. This method generates a hierarchy of segmentations with increasing number of segments towards the leaves of the tree and fig. 5.9 shows results for all of theses layers (black line). It can be seen that StreamGBH gives better results for explained colour variation, but much poorer results for explained depth variation. Additionally segments are much less compact than segments produced by *t-DASP*. This highlights the fact that StreamGBH overfits on the available colour information at the expense of preserving 3D geometry edges. *t-DASP* which additionally considers depth information can find a much better balance between satisfying colour and depth constraints while producing segments with a more compact shape.

Fig. 5.10 shows this fact quantitatively by comparing a segmentation computed with *t-DASP* to a segmentation computed with StreamGBH. StreamGBH segments respect colour information very well but have problems with noise, shadows and invisible geometry edges. Fig. 5.11 displays a temporal sequence of segmentations for both *t-DASP* and StreamGBH. It is visible that both methods compute sequences with similar temporal coherence, but *t-DASP* segments respect 3D geometry edges better.

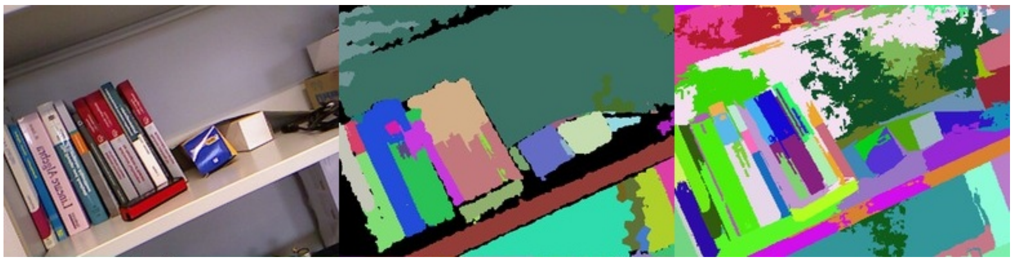


Figure 5.10: A close-up comparison of segments computed by *t-DASP* (middle) and StreamGBH (right). The colour image (left) is provided for reference.

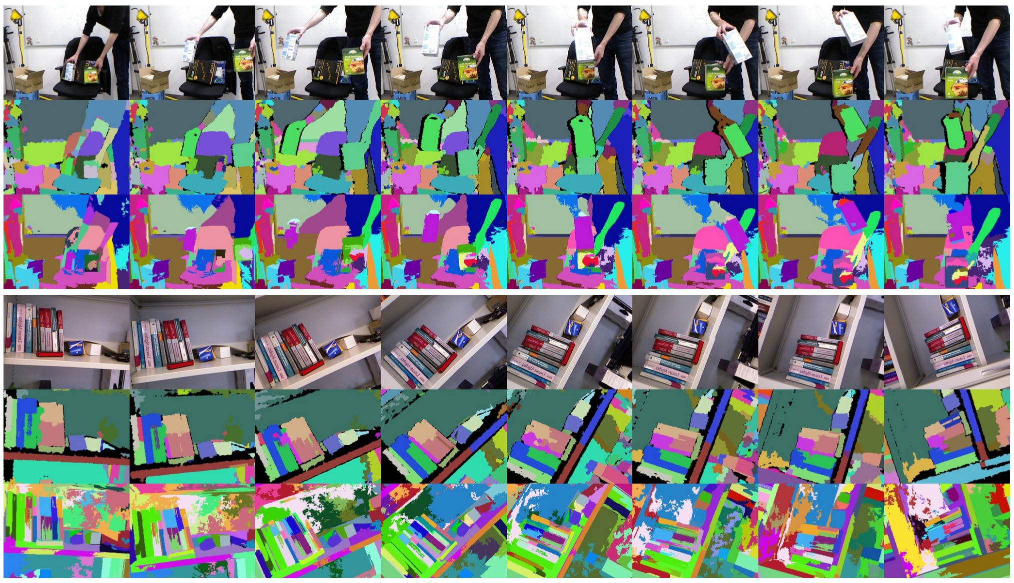


Figure 5.11: Results for two example scenarios - each shows input colour images (first row), *t-DASP* segmentation results (second row) and a comparison to StreamGBH (third row).

Part II

Event-Based SLAM

6 Event-based vision

A dynamic vision sensor consists of independently and asynchronously operating pixels. When a pixel notes a change in illumination it reports its pixel location, resulting in a sparse stream of pixel events. This stands in contrast to classic cameras which provide dense images at a fixed framerate and opens a new field of computer vision – event-based vision.

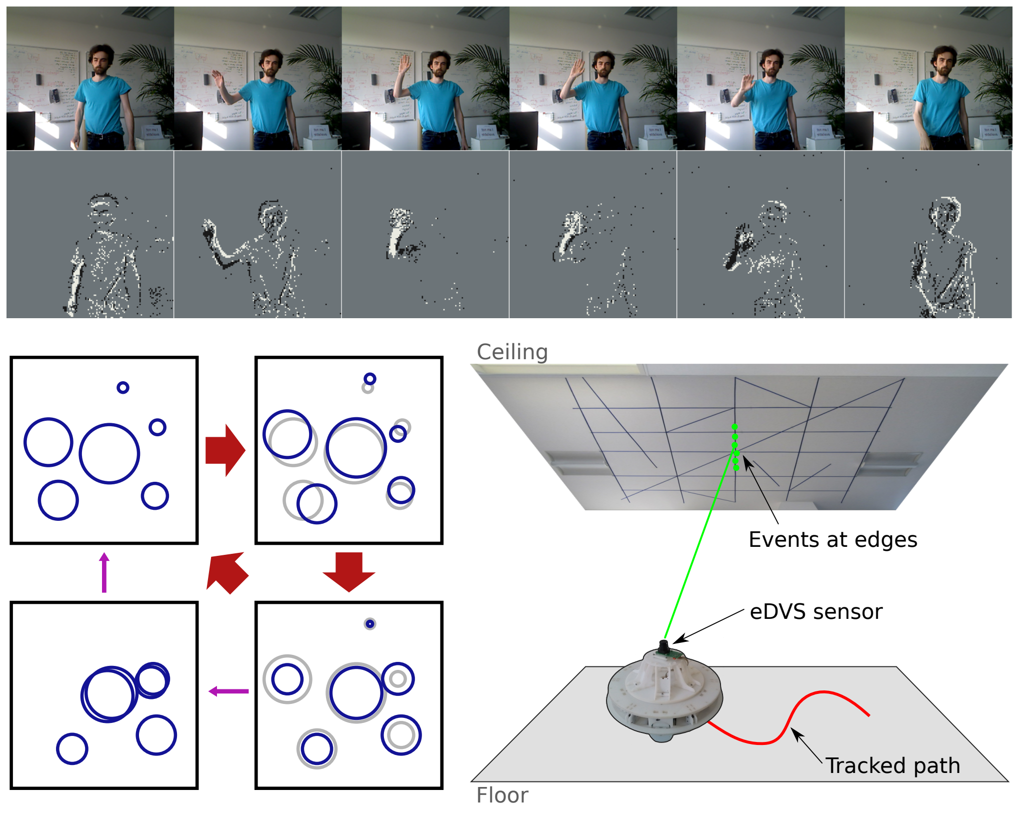


Figure 6.1: **Top**: Comparison of a frame-based video stream and pixel events representing only dynamic changes. **Bottom**: Processing such a sparse stream of pixel events requires novel algorithms like *Event-based Particle Filter* (**left**) and enables real-time applications for embedded systems (**right**).

6.1 Event-based vision sensors

In the first part of this thesis a sparse data model was computed based on dense sensor input. While high-level algorithms like *s-DASP* (§4) and *t-DASP* (§5) can work efficiently with *DASP* (§3), still considerable effort has to be made to convert the dense sensor data into the sparse representation. In this part of the thesis another approach to efficient sparse models will be taken, by considering dynamic vision sensors which directly produce sparse data streams *in hardware*.

The family of dynamic vision sensors [50, 19] produces a stream of pixel events which represents only dynamic changes in the perceived brightness. Each pixel operates asynchronously and independent from all other pixels, and continuously integrates the measured pixel brightness. When the change in brightness exceeds a certain dynamically adapted threshold, the pixel fires an event consisting of its pixel location, a timestamp, and a parity flag indicating if brightness increased or decreased. This stands in contrast to classic camera sensors which produce a sequence of still images. All pixels integrate illumination over a fixed time span defined by the framerate and shutter speed, and for each frame complete measurements of the whole pixel grid are reported. In the following the word **event** will always indicate a pixel event which carries the information when and where on the pixel grid it occurred.

This main difference between a frame-based and an event-based sensor is highlighted in fig. 6.2. In vision sensors pixels measure a continuously changing function value like lighting intensity or red, green and blue intensity for colour perception. A frame-based sensor samples the function at fixed time-intervals and reports the current value. The actual change in measured values from one sample to the next can vary greatly dependent on the rate of change. i.e. the slope, of the underlying function. On the contrary, an event-based sensor continuously observes the change of the measured function and reports only if the difference with respect to the last time a value was reported lies over a given threshold. Here, the defining quantity is no longer the value itself but the time difference since the

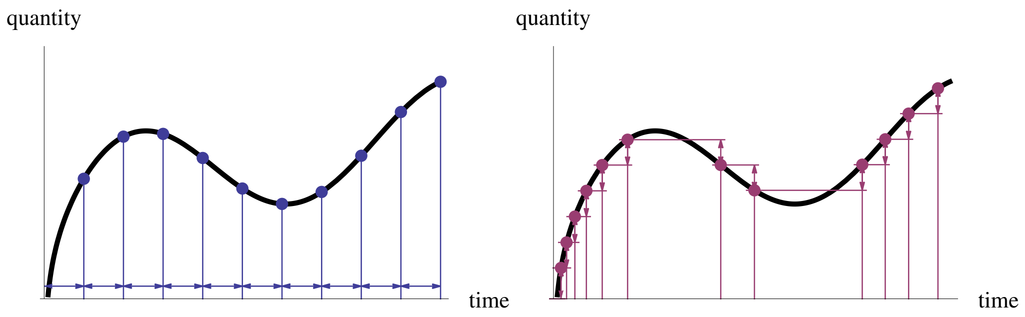


Figure 6.2: Measuring a quantity with a sensor taking samples at fixed time steps (**left**) and with an event-based sensor that only fires when the quantity has changed for a specific amount (**right**).

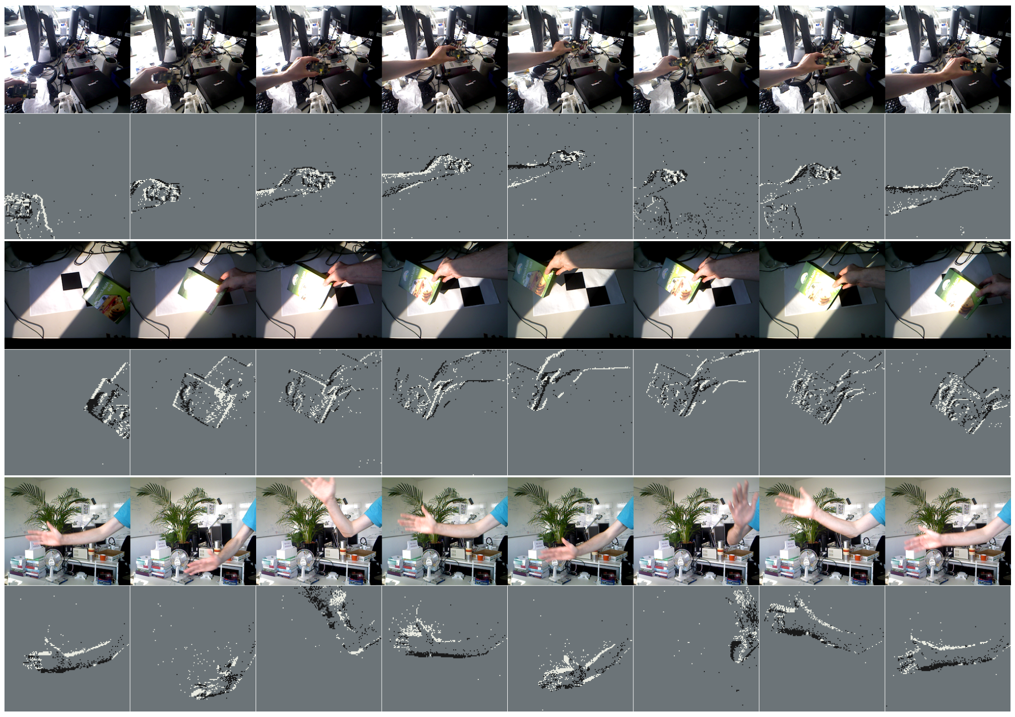


Figure 6.3: Demonstrating the advantages of dynamic vision sensors by comparing event images to corresponding color images in three scenarios: cluttered background (top), high contrast (middle), and fast motions (bottom).

last time an event was reported.

An impression of the data produced from an event-based dynamic vision sensor is given in fig. 6.3. Here, a classic image sensor is compared to the data stream generated by a dynamic vision sensor. For the classic image sensor some of the fixed frames are displayed, which is the normal way to show a video stream in print. For the visualization of pixel events, a number of consecutive events is collected, and theses events are display together in form of an image. The colour of an event pixel indicates if this pixel has reported an increase or decrease in lighting: white indicates increased lighting, black decreased lighting, and grey that no event has been reported for this pixel. It is important to note that even though events are displayed together in images, they actually occur one after another with continuous timesteps. The ”integrated” visualization in form of frames serves only as demonstration.

Dynamic vision sensors have four main advantages over classic, frame-based image sensors. First, the amount of data generated by the sensor is much smaller as only changes are transmitted and static, redundant information is omitted. This is especially advantageous for embedded real-time systems. Second, event-generation has sub-millisecond time resolution making it suitable for high-speed control applications. A similar framerate may be achieved with conventional high-speed camera at a much higher financial and computation costs. Third, the automatic threshold used for event generation is chosen by every pixel itself. Thus pixels can reliable detect illumination changes even in the presence of very high contrast changes within one image. Finally, the sensor implicitly provides pre-processing for tracking applications which often rely on detecting changes in the image. Detecting changes in conventional image streams, using for example background subtraction [44, 91], requires a high bandwith and is computational expensive. Of course, dynamic vision sensors have the obvious disadvantage that they can not see static objects, thus limiting it's capabilities to detect objects when neither moving the sensor or the objects. This disadvantage can be alleviated by combining the dynamic vision sensor with a conventional image sensor or even a combined colour and depth sensor. For example in §8 an event-based sensor will be combined with a PrimeSense active depth sensor to produce an stream of 3D point events.

The reduced bandwidth is one of the main issues which lead to a much higher computational efficiency of algorithms working with dynamic vision sensors. Table 6.1 lists the specifications of some common video formats and compares their bandwidth. The amount of data produced by a the embedded dynamic vision sensor (*eDVS*) is two magnitudes lower than the bandwidth of a classic camera sensor. Even a depth-augmented dynamic vision sensor as presented in §8 only marginally increases the bandwidth. Bandwidth computation assumes uncompressed data and three 8 bit integers per pixel for red, green and blue colour value each. For the PrimeSense device (RGB-D) an additional 16 bit integer is used for the depth information.

Two examples for dynamic vision sensors are the *Dynamic Vision Sensor* (*DVS*) [19] and the *Embedded Dynamic Vision Sensor* [50] (*eDVS*) (see fig. 6.4). The *DVS* is a commercially available product with excellent optics and can handle up to one million events per second. The *eDVS* is a small embedded version of the *DVS*, which consumes a marginal amount of power and can handle up to 100,000 events per second. This is sufficient for most practical applications and in the following only the *eDVS* will be used. Table 6.2 shows detailed specifications of the *eDVS* and compares them to a classic colour camera sensor.

Dynamic vision sensors have been successfully used in various applications. In [19], it is used to track the pose of a pencil and balance it in realtime on the tip of a

Name	**Resolution**	**Framerate**	**Pixel data**	**Bandwidth**
PAL	576 x 432	25 Hz	RGB / 24 bit	18 MB/s
PrimeSense	640 x 480	30 Hz	RGB-D / 40 bit	44 MB/s
HD-TV	1920 x 1080	30 Hz	RGB / 24 bit	178 MB/s
FASTCAM SA-X2	640 x 468	40000 Hz	RGB / 12 bit	17140 MB/s
eDVS	128 x 128	event-based	XY-P / 16 bit	0.2 MB/s
D-eDVS	128 x 128	event-based	D-XY-P / 32 bit	0.4 MB/s

Table 6.1: Video stream bandwidth for a selection of video formats

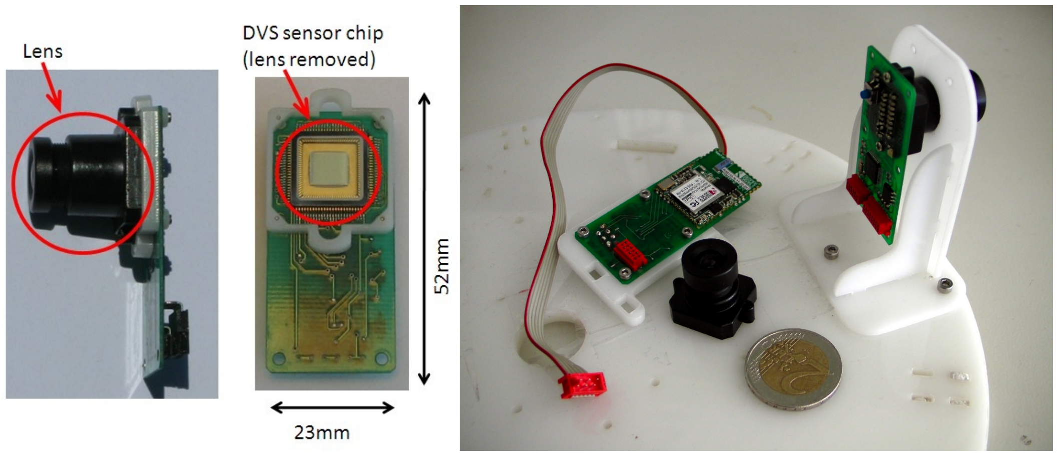


Figure 6.4: *eDVS* sensor and various accessories like lens mount, WiFi-module and *eDVS* mount.

small actuated robotic arm. In another application, a real-time 3D visual tracking system was developed [55]. The authors used several high-speed LEDs blinking at different frequencies as active markers which are detected by an eDVS sensor and a high-speed 2D tracking algorithm was proposed to identified LEDs and track them over time. Other research [71, 65] tries to estimate depth information from a stereo *DVS* setup. The event-based paradigm is also relevant in a completely different context. In [67] a method is proposed for detecting the location of natural disasters using messages from the social media Twitter, so called "tweets", as events. They propose an event-based tracking algorithm which enforces a synchronized timing and does not consider the asynchronous nature of events.

In this chapter, an event-based particle filter algorithm [80] will be presented

	Point Grey Flea 3	**eDVS**
Dimensions	29 x 29 x 30 mm (without optics)	23 x 52 x 5 mm (without optics)
Specification	FL3-U3-13S3C-CS Sony IMX035 CMOS 1/3", 3.63 μm	4M 2P 0.25 μm 6 mm x 6.3 mm
Connection	USB 3.0	Serial / USB 2.0
Pixel resolution	1328 × 1048	128 × 128
Time resolution	120 fps (= 1 frame every 8.3 ms)	reaction time ca. 10 μs deadtime time ca. 100 μs
Bandwidth	1274 MBit/s	4 MBit/s (max.)
Price	ca. USD 600	ca. USD 50

Table 6.2: Comparison of a classic image sensor and an *Embedded Dynamic Vision Sensor*

which uses only the sparse stream of pixel events to continuously track the state of an observed system. The algorithm is very highly efficient and produces a state estimate for every event, thus working with very high temporal accuracy. The capabilities of the algorithm are demonstrated in a robot self-localization scenario, where a robot moves on the ground and observes features on the ceiling.

This chapter is continued with an investigation into the characteristics of event-based vision in §6.2 and some remarks to the integration of event-based sensor information in §6.3. In §6.4 the event-based particle filter will be presented and in §6.5 it will be applied exemplary to a 2D self-localization scenario.

6.2 Characteristics of event-based vision

6.2.1 Sensor model

For this thesis the *eDVS* sensor is used as an event-based vision sensor. While many of the following considerations are true for general dynamic vision sensors, concrete specifications or examples are always given for the *eDVS*. The *eDVS* consists of an $S \times S$ pixel grid, here $S = 128$, and follows a regular pinhole camera model. The pinhole camera model states that a point $p = (p_x, p_y, p_z) \in \mathbb{R}^3$ is projected into the pixel location $e = (e_x, e_y) \in \mathcal{R}$ following the equation

$$e = \left(f \frac{p_x}{p_z} + c_x, f \frac{p_y}{p_z} + c_y \right) , \tag{6.1}$$

where $c = (c_x, c_y) \in \mathcal{R}$ is the optical centre of the projection in pixel coordinates and f is the pixel focal length parameter. $\mathcal{R} = [0 \,|\, S]^2 \subset \mathbb{R}^2$ is the possible space of pixel events. f can be computed from the opening angle α_{fov} with

$$f = \frac{S}{\tan\left(\frac{1}{2}\alpha_{\text{fov}}\right)} . \tag{6.2}$$

Due to limitation of optical lenses, especially small ones, the projection is additionally distorted. A simple distortion model like

$$\mathcal{R} \to \mathcal{R},\, e \mapsto c + (e - c) \left(1 + \kappa_1 \|e - c\| + \kappa_2 \|e - c\|^2\right) \tag{6.3}$$

proves to be sufficient. In the following it is assumed, that the camera parameters c and f are known and that event coordinates are already distorted.

6.2.2 Event generation model

The main difference of an event-based image sensor in comparison with a classical image sensor is the way sensor readings depend on observations. In order to better understand the nature of event generation, let’s make a thought experiment for a very simple scenario. Assume the camera is positioned in front of a uniformly coloured piece of cloth at a fixed distance D. On the piece of cloth there are several singular points with a high contrast with respect to the colour of the cloth. The size S_W of such a ”feature point” should be very small, such that it always hits exactly one pixel when projected onto the pixel grid. Remember, for a pinhole projection the size S_{px} of such a point on the pixel grid can be computed with

$$S_{px} = S_W \frac{f}{D}. \tag{6.4}$$

Event generation is influenced by two factors: On the one hand, an event is generated whenever the projection of a feature point moves from one pixel on the sensor screen to a neighbouring pixel. If the point moves parallel to the sensor at distance D this happens in average when the point has moved a world distance of

$$W_{\text{trig}} = \frac{D}{f} \tag{6.5}$$

(from eq. 6.4 with $S_{px} = 1$). On the other hand, the number of generated events depends directly on the number of feature points G_0. Here we assume, that the sensor generates exactly one event when a feature point traverses from one pixel to the next. In reality, the effective number of feature points is related to the contrast and several events may be generated, but this effect can be compensated with a linear factor and for simplicity is ignored for now.

If there are G_0 effective feature points which moved together for a distance of x, e.g. due to a linear motion of the sensor, the expected number of generated events is

$$\mathrm{E}_x = x \frac{G_0}{W_{\text{trig}}} = x \frac{f\, G_0}{D}. \tag{6.6}$$

For rotations of the sensor about its projection centre c a similar consideration can be made. A point which is projected into a pixel with distance r_i from the sensor centre c travels a distance of $\theta\, r_i$ pixels when the sensor rotates for an angle of θ (angles are measured in radians). This quantity is directly equal to the number of generated events.

$$\mathrm{E}_\theta = \sum_{i=1}^{G_0} \theta\, r_i \tag{6.7}$$

Assuming that feature points are uniformly distributed over a disc with radius $\frac{S}{2}$ around the pixel centre, then the expected number of generated events can be averaged with

$$\mathrm{E}_\theta \approx \theta \frac{S\, G_0}{3} \tag{6.8}$$

The approximation in eq. 6.8 is not fully correct as the disc does not cover the full rectangular pixel grid. A more accurate approximation computed empirically is $\mathrm{E}_\theta \approx 0.38247\, \theta\, S\, G_0$.

The expected quantities E_x and E_θ express the number of generated events when the sensor has moved for a given distance x or rotated around its axis for a given angle θ. Conversely the expected movement Δ_x and rotation Δ_θ can be computed when a given number of events have occurred. As the equations are linear, the expected movement is expressed with respect to exactly one event: The average distance moved for one event is

$$\Delta_x = \frac{D}{f\,G_0}\,. \tag{6.9}$$

and the average rotation per event is

$$\Delta_\theta = \frac{3}{S\,G_0}\,. \tag{6.10}$$

For a dynamic model of event generation one sees directly from eq. 6.6 and eq. 6.8 that

$$\mathrm{E}_x \propto x \text{ and } \mathrm{E}_\theta \propto \theta\,. \tag{6.11}$$

This highlights the fact that for a static environment the number of generated events depends only on the amount of movement or rotation of the image sensor, and not on its linear and angular velocity. For a fixed distance a slow motion will generate the same events as a fast motion, only the rate at which events occur will differ. The velocity of the sensor has only an influence on the speed with which events are generated, i.e. the event rate is proportional to the sensor velocity. In practice this property may not be entirely true due to practical limitations of the *eDVS*.

6.3 Information integration

Event-based vision sensors provide a stream of singular pixel events where each pixel event on its own carries only little information. Only a batch of many events carries enough information to track objects or to compute ego-motion. Thus when working with a stream of events a crucial question is how to collect and process, i.e. ”integrate”, knowledge of individual events to form high-level models.

A primitive mechanism to integrate individual events is ”windowing”. Windowing collects events for a fixed time interval and processes them at once as a batch. However this approach is undesirable as it neglects several of the advantages of event-based vision. The time windows has to be big enough to capture enough information and events have to be memorized until evaluation which effectively returns to a frame-based approach. Additionally waiting some time for enough events can not provide a realtime answer and annihilates the chance to operate with a high temporal precision. In the following a more suitable event integration mechanism is presented which works on individual events.

We consider a stream of pixel events $(e_1,\dots,e_n)$ where each event $e_i \in \mathscr{R}$ is a pixel coordinate. For the following analysis we assume that the function which extracts information from the event stream is of additive nature. This means that a function f which is defined on a arbitrary set of events $E \subset \mathscr{R}$ can be also be evaluated by applying it to all events individually and summing up the results:

$$f(\{e_1,\ldots,e_n\}) = \sum_{i=1}^{n} f(e_i) \tag{6.12}$$

In computer vision many simple object detection or tracking algorithms are of additive nature and check for each pixel of an image how well it satisfies a given model.

The crucial point which shall be analysed here is the temporal influence of pixel events on the whole measurement. Each event e_i is annotated with a timestamp $t_i \in \mathbb{R}_+$, thus the index i does not indicate the time of an event but an continuously incremented unique index. The current time is given by t_n, where n is the number of events observed so far. This extends the simple additive model from eq. 6.12 to a weighted additive model

$$f(\{e_1,\ldots,e_n\}) = \frac{\sum_{i=1}^{n} w(t_n - t_i)\, f(e_i)}{\sum_{i=1}^{n} w(t_n - t_i)} \tag{6.13}$$

where $w(t_n - t_i)$ is the weight given to an event depending on the elapsed time since the event happened. Events from a long time ago, i.e. $t_n - t_i$ is big, should have less influence on the current evaluation of the system state. For simplicity we define $r_i := f(e_i)$ and $s_n := f(\{e_1,\ldots,e_n\})$, thus

$$s_n = \frac{\sum_{i=1}^{n} w(t_n - t_i)\, r_i}{\sum_{i=1}^{n} w(t_n - t_i)}\,. \tag{6.14}$$

On the model from eq. 6.14 various possible models for temporal weighting can be analysed. The simple approach of windowing is realized with

$$w_{\text{win}}(\Delta t \,|\, T) := \begin{cases} 1 & \Delta t < T \\ 0 & \text{else} \end{cases} \tag{6.15}$$

weighting all events in the given time interval with 1 and ignoring all other events.

6.3.1 Exponential decay

In this context, an approach is investigated where the temporal weight function is continuous and monotonic, i.e. continuously decreasing. We would like to have a weight which guarantees

$$w(x) > w(y) \text{ iff } x > y \text{ and } w(x) = w(y) \Rightarrow x = y\,. \tag{6.16}$$

This guarantees that as time proceeds recent partial scores are to some degree more relevant to the total score. A possible model is exponential decay

$$w_{\text{exp}}(x \,|\, \lambda) := e^{-\lambda x} \tag{6.17}$$

with a decay constant $0 < \lambda \in \mathbb{R}$. This model clearly provides monotonic decreasing weights with increasing time difference.

For computation of an additive information function the exponential model is especially interesting as it allows to simplify eq. 6.14 by computing the total score s_n by only using the previous total score s_{n-1} and the current partial score r_n in a linear equation:

$$s_n = (1-\alpha)\, s_{n-1} + \alpha\, r_n \tag{6.18}$$

Such an iterative model has minimal memory footprint as only one variable need to be memorized at all times and it is computational efficient as the integration of a new event only requires a simple linear computation.

Proposition 3. *For the exponential decay model $w(x) = e^{-\lambda x}$ the total quantity s_n can be computed iteratively as*

$$s_n = (1-\alpha_n)\, s_{n-1} + \alpha_n\, r_n\,, \;\; s_0 = 0 \tag{6.19}$$

where $\alpha_n \in \mathbb{R}_+$ are constants which only depend on the event timestamps and can be computed iteratively as

$$\alpha_n = \frac{\alpha_{n-1}}{\alpha_{n-1} + e^{-\lambda(t_n - t_{n-1})}}\,, \;\; \alpha_1 = 1\,. \tag{6.20}$$

Proof. For simplicity define $q_{i,j} := e^{-\lambda(t_i - t_j)}$. The trivial facts $q_{i,i} = 1$ and $q_{i,k}\, q_{k,j} = q_{i,j}$ follow directly. Additionally define

$$A_n := \sum_{j=1}^{n} e^{-\lambda(t_n - t_j)} = \sum_{j=1}^{n} q_{n,j}\,.$$

Again the trivial facts $A_1 = 1$ and $A_{n+1} = 1 + q_{n+1,n}\, A_n$ follow directly by inspection. Eq. 6.14 can be transformed to

$$s_n = \frac{\sum_{i=1}^{n} e^{-\lambda(t_n - t_i)}\, r_i}{\sum_{j=1}^{n} e^{-\lambda(t_n - t_j)}} = \frac{\sum_{i=1}^{n} q_{n,i}\, r_i}{A_n}\,. \tag{6.21}$$

The proof continues by induction. First check $n = 1$: From eq. 6.19 we have $s_1 = (1-\alpha_1)\, s_0 + \alpha_1\, r_1 = r_1$ which corresponds to $s_1 = r_1$ from eq. 6.21.

Now for the induction $n \to n+1$ assume that the assumption is true for s_n. From eq. 6.21 we have:

$$s_{n+1} = \frac{1}{A_{n+1}} \sum_{i=1}^{n+1} q_{n+1,i}\, r_i = \frac{1}{A_{n+1}} \sum_{i=1}^{n+1} q_{n+1,n}\, q_{n,i}\, r_i = \frac{q_{n+1,n}}{A_{n+1}} \sum_{i=1}^{n+1} q_{n,i}\, r_i$$

$$= \frac{q_{n+1,n}}{A_{n+1}} \left(\sum_{i=1}^{n} q_{n,i}\, r_i + q_{n,n+1}\, r_{n+1} \right) = \frac{q_{n+1,n}}{A_{n+1}} \sum_{i=1}^{n} q_{n,i}\, r_i + \frac{1}{A_{n+1}}\, r_{n+1}$$

$$= \frac{q_{n+1,n} A_n}{A_{n+1}} s_n + \frac{1}{A_{n+1}} r_{n+1} = \left(1 - \frac{1}{A_{n+1}}\right) s_n + \frac{1}{A_{n+1}} r_{n+1} .$$

The last line follows with the assumption about s_n and the iterative rule for A_{n+1}.

It remains to be shown that $\alpha_{n+1} = \frac{1}{A_{n+1}}$ can be computed by the iterative formula eq. 6.20. This can be deduced by another induction were the essential part is the computation

$$\frac{\alpha_n}{\alpha_n + q_{n,n-1}} = \frac{\frac{1}{A_n}}{\frac{1}{A_n} + q_{n,n-1}} = \frac{1}{1 + q_{n,n-1} A_n} = \frac{1}{A_{n+1}} = \alpha_{n+1} .$$

□

Note that α_n can also be expressed as

$$\alpha_n = L\Big(\lambda(t_n - t_{n-1}) + \ln \alpha_{n-1}\Big) \tag{6.22}$$

where L is the logistic function

$$L(x) := \frac{1}{1 + e^{-x}} \tag{6.23}$$

While the iterative computation of s_n with eq. 6.19 seems simple, the computation of the factor α_n after equation eq. 6.20 is daunting. For illustration some special cases are presented:

1. All events happen at the same time: $t_i = t \; \forall i \Rightarrow A_n = n$ and $\alpha_n = \frac{1}{n}$. Thus the whole process iteratively computes the mean of all samples. Note that the influence of the last sample r_n gets smaller over time as the constant α_n decreases with the number of samples already recorded.

2. Events happen in fixed time intervals: $t_i = c + t_{i-1}$, $t_1 = 0$, $c \in \mathbb{R}_+$. Let $\lambda > 0$ and $\mu := e^{-\lambda c}$, then $A_n = 1 + \mu A_{n-1}$, $A_1 = 1$, so $A_n = 1 + \mu + \mu^2 + \ldots + \mu^n = \frac{1-\mu^{n+1}}{1-\mu}$ and $\alpha_n = \frac{1-\mu}{1-\mu^{n+1}}$. As $\mu < 1$, we have $\lim_{n\to\infty} \alpha_n = 1 - \mu$. This illustrates that a constant rate of events requires a constant rate of decay and thus a constant "forgetting" of all information.

3. $\mu = \frac{1}{2}$ in the case above. This gives $\alpha_n = \frac{1}{2-\frac{1}{2^n}} \approx \frac{1}{2}$ and thus $s_n \approx \frac{1}{2}(s_{n-1} + r_n)$.

6.3.2 Decay constant for dynamic vision sensors

The decay constant λ in the exponential decay function models how fast information is forgotten from one event to the next based on the time since the last event. This has a direct application for a system in which the state is changing slowly over time and events occur at random timestamps which are not related any further to how the system state changes. However in case of dynamic vision sensors the number of generated events is directly related to the total movement of the sensor if the environment is static (see §6.2.2).

From equations eq. 6.11 it follows that the rate of events is proportional to the velocity of the sensor. This has a strong implication when the state, i.e. position and rotation, of the dynamic vision sensor should be estimated. When the time difference between events is large the previous state should not be forgotten as fast as when the time difference is small. Large time differences indicate that the sensor has moved only little and that the previous state estimate is still quite valid. Thus for a dynamic vision sensor it is reasonable to choose the decay constant based on the velocity which itself is proportional to the event rate:

$$\lambda(t) \propto \|\partial_t x(t)\| + |\partial_t \theta(t)| \propto \partial_t E(t) \tag{6.24}$$

Proposition 4. *If the decay constant $\lambda(t)$ is proportional to the rate of events $\partial_t E(t)$ the weight function w in the exponential decay model (see eq. 6.17) is constant.*

Proof. The time between events ΔT is directly related to the event rate:

$$\Delta T(t) \propto \left(\partial_t E(t)\right)^{-1}$$

We get

$$\lambda(t) \propto \partial_t E(t) \propto \left(\Delta T(t)\right)^{-1}$$

which gives $w(\Delta T(t)) = e^{-\lambda(t)\Delta T(t)} \propto 1$. □

We see that for information integration with a dynamic vision sensor all events are weighted equally and independently from their timesteps. With eq. 6.20 this results in

$$\alpha_n = \frac{\alpha_{n-1}}{\alpha_{n-1} + C} \tag{6.25}$$

for the iterative update model in proposition 3 with a constant $0 < C < 1$. Additionally:

$$\lim_{n\to\infty} \alpha_n = 1 - C = \text{const.} \tag{6.26}$$

as seen previously in the examples in §6.3.1.

The choice of the constant C and thus the decay constant α is more or less a free variable. A low decay introduces a smoothing effect while a high decay may be prone to noise. The decay constant can be transformed to a quantity which has an intuitive explanation by requiring that the last G_0 events have a total relative influence on the total score of β. By inspections one sees that this results in

$$\alpha = 1 - \sqrt[G_0]{1 - \beta}. \tag{6.27}$$

As an example: For a choice of $\beta = 0.95$ and $G_0 = 250$ this gives $\alpha \approx 0.012$.

6.4 Event-based particle filtering

As seen previously in §6.2 and §6.3 dynamic vision sensors introduce a new principle of visual processing. The implications and benefits are further investigated here by adapting the classic particle filter algorithm [37] to the characteristics of dynamic vision sensors. The resulting *Event-based Particle Filter* (*EB-PF*) algorithm [80] is a novel particle filter algorithm suitable for ego-motion or object tracking.

In the following, a general introduction into temporal Bayesian networks and the classical particle filter algorithm is given in §6.4.1. Afterwards the *Event-based Particle Filter* algorithm is presented in §6.4.2.

6.4.1 Dynamic Bayesian networks and Condensation

A Bayesian network is a probabilistic graphical model for representing random variables and their conditional dependencies. In contrast to other networks like a Markov network, a Bayesian network explicitly requires that dependencies between its random variables are describes as a directed acyclic graph. This limitation simplifies inference and learning of Bayesian networks.

A dynamic Bayesian network models the temporal relation of random variables which change in discrete steps over time. For a dynamic Bayesian network a Markov assumption is used which states that variables only depend on other variables from the same timestep or on variables from the previous timestep. Under a Markov assumption random variables may not depend on random variables which lie more than one timestep in the past or even in the future.

For this thesis the visual tracking of a system state is investigated. The systems state is denoted with X_t where the subscript t indicates the discrete timestep. For the estimation of ego-motion or tracking of an object the system state would be the position and the orientation of the sensor or object. The system state shall be deduced with the help of an observation Z_t which was made at the same timestep t. The series of all observations until now is denoted with $\mathcal{Z}_t = (Z_1, \dots, Z_t)$. The corresponding dynamic temporal network is depicted in fig. 6.5.

Since the system state shall be deduced from the set of observations one is interested in $P(X_t|\mathcal{Z}_t)$, which indicates the probability of a state given a specific sequence of observations. As explained in [37] the Markov assumption can be used to transform this into

$$P(X_t|\mathcal{Z}_t) \propto {\color{red}P(Z_t|X_t)} \int {\color{blue}P(X_t|X_{t-1})}\, P(X_{t-1}|\mathcal{Z}_{t-1})\, dX_{t-1} \,. \tag{6.28}$$

This equation expresses an iterative model to compute the desired probability using an estimate from the previous timestep and probabilistic models describing the system.

The **motion model** $P(X_t|X_{t-1})$ of the system describes how the system develops naturally from one timestep to the next. For example, when computing the

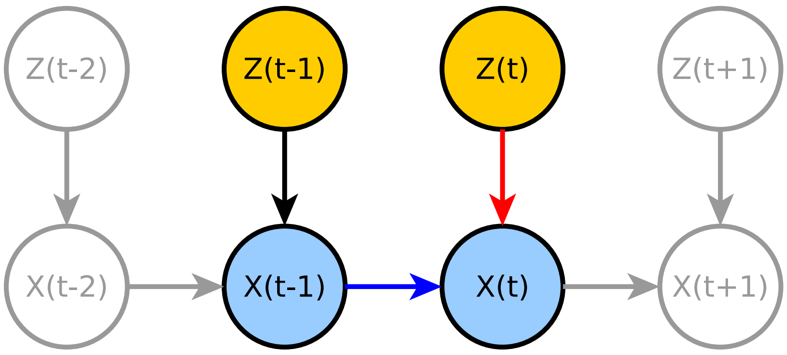


Figure 6.5: A temporal Bayesian network for tracking purposes. The blue arrow corresponds to the motion model $P(X_t|X_{t-1})$ and the red arrow to the sensor model $P(Z_t|X_t)$.

pose of a moving robot this would be an estimate on the dynamics of the robot, i.e. the estimated or maximum possible distance the robot can move between two timesteps. $P(X_{t-1}|\mathcal{Z}_{t-1})$ is the **sensor model** which expresses the probability that a measurement occurs given a specific system state. For the example of a moving robot, the sensor model would describe how likely the current measured image from the visual sensor is, given that the robot is at a hypothetical pose. Theses two models correspond directly to the highlighted conditional probabilities in fig. 6.5.

For an algorithm which computes concrete probabilities for the state estimate, one has to decide on a feasible method to represent probability distributions over a possibly high-dimensional state space. One possible way is to represent a probability distribution by a weighted set of sample points. The corresponding computational framework is a particle filter which has been used in the Condensation algorithm for tracking articulated objects [37].

The Condensation algorithm represents a probability distribution over the state space Ω by a set of samples $\{p_1,\dots,p_n\} \subset \Omega$. Each sample p_i is weighted by a sample weight $w_i \in \mathbb{R}_+$. The combination of sample and weight in the context of particle filters is often called **particle**. Here two representations are possible. The direct way is expressing the probability $P(x)$ of a specific subset $U \in \Omega$ by representing it by one sample and setting the weight proportional to the integrated probability $\int_U P(x)$ over this subset. Another way is using equally weighted samples which are distributed over the domain Ω such that each sample point represents a subset of equal integrated probability. The Condensation algorithm alternates between theses two representations.

The Condensation particle filter algorithm proceeds as follows:

1. **Resampling:** New samples $\{p_1^{(t)},\dots,p_n^{(t)}\}$ are drawn from the probability distribution represented by particle samples $\{p_1^{(t-1)},\dots,p_n^{(t-1)}\}$ and particle weights $\{w_1^{(t-1)},\dots,w_n^{(t-1)}\}$ from the previous timestep. For the very first

timestep an random distribution is chosen or an initial guess is provided manually.

2. **Motion model:** Each sample $p_i^{(t)}$ is propagated according to the given motion model $P(X_t|X_{t-1})$. A simple form of a motion model which can be sufficient for many applications is white noise: $p_i^{(t)} = \mathcal{N}(p_i^{(t)}, \Sigma)$.

3. **Sensor model:** Sample weights are updated to adjust the current belief using the information gained from the current observation Z_t: $w_i^{(t)} = P(Z_t|p_i^{(t)})$. Afterwards the weights are normalized.

The representation of the state space probability distribution by a set of particles has the main advantage that multi-modal distributions can be represented easily and that the distributions focuses on regions in the state space which have a high probability. Particle filters are also strikingly easy to implement – one ”for loop” is enough. Moreover it has been shown that particle filters give optimal solutions when the number of particles goes towards infinity.

However for a high-dimensional state space, also a large number of particles is required, and when the number of dimensions increases the number of particles basically has to increases exponentially. The problems gets worse when poor motion models or ambiguous sensor models are used. When the function of the motion model is neglected the problem of tracking in a high-dimensional state space is basically a search problem. To find the optimum solution all possibilities have to be checked and to find a solution which is optimal up to some error the state space has to be covered sufficiently dense. With particle filters, this problem can be reduced with factored sampling, but this requires additional knowledge or additional constraints limiting the generality.

6.4.2 The event-based particle filter algorithm

A typical application of a particle filter is tracking an object which is observed by a camera. For each new frame the last state estimate is updated using the motion model and the current image is used in the sensor model to update particle scores accordingly. The particle filter algorithm provides a new state estimate for every new measurement. Particle filters can work with ambiguous sensor information, e.g. partial occlusion or ambiguous sensor information, but in a classical application for most frames fairly accurate conclusions about the system state are possible.

This procedure could be translated to dynamic vision sensors by using individual events as measurements and running one iteration of the Condensation algorithm for each new event. Due to two main reasons this is disadvantageous. For an event based sensor an individual measurement consists of only one pixel event. The information from one individual pixel is highly ambiguous and additionally may be misleading due to sensor noise. Thus if sample weights are computed based on only one event, this may have very strong and undesired

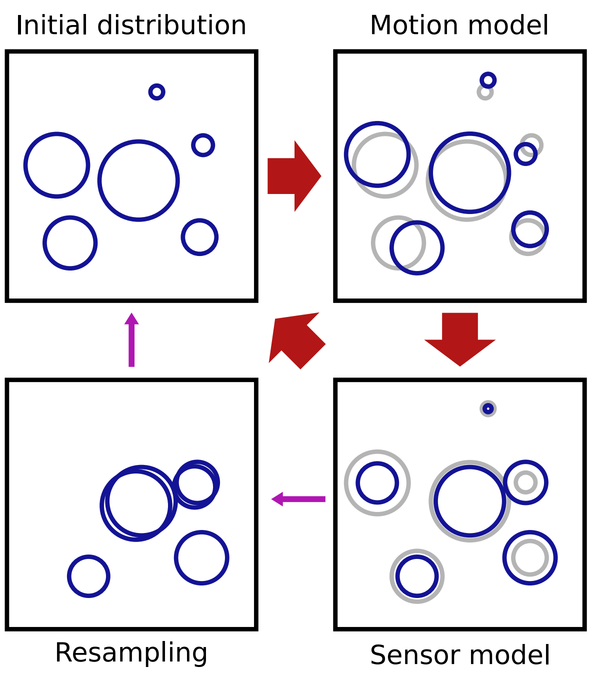


Figure 6.6: The process of the *Event-based Particle Filter* algorithm. Motion model and sensor model are applied for every new observation, i.e. retina pixel event. Particle score updates due to observations are smoothed by using an exponential decay model. Resampling is only executed periodically when a certain amount of new information, i.e. a certain number of new events, has been gathered.

implications on the reselection of particles. The second disadvantage is that the resampling step is executed for every event. For a classic application the computational overhead of the Condensation algorithm compared to the complexity of the motion model and especially the sensor model is very low. For an event based sensor only one pixel needs to be evaluated with the sensor model which normally has negligible runtime and thus constant resampling is a waste of time.

In the following the *Event-based Particle Filter* (*EB-PF*) algorithm [80] is presented which adapts the classic particle filter algorithm to yield an efficient computational model for an event-based stream of pixel events. *EB-PF* changes the classic Condensation algorithm at two key points. First, the sample weight uses the method of information integration derived for event-based sensors in §6.3. Second, the resampling step is delayed until a specific number of events has been observed. Fig. 6.6 visualizes the procedure of the *EB-PF* algorithm and details are explained in the following.

As explained earlier exponential decay provides a reasonable and easy way to evaluate a separable evaluation function over a set of events where individual contributions are weighted based on the elapsed time since the event happened. For the computation of sample weights the exponential decay model is used to provide a smooth update rule for sample weights:

$$s_i^{(t)} = (1-\alpha)\, s_i^{(t-1)} + \alpha\, P(e_t|X_t = p_i^{(t)})\,,\ \ s_i^{(0)} = 1\,. \tag{6.29}$$

Here e_t is the current event, i.e. observation, which is used to update sample weights.

The decay constant α models how large the influence of recent changes in the state should be in comparison to the past. As observed earlier in §6.3.2, the decay constant α is (almost) constant and independent of event timesteps for dynamic vision sensors. It can be computed for example by specifying how big the relative influence of the recent events on the total score should be (see eq. 6.27). It is important to note that the decay model from eq. 6.29 is executed *per event*, and not in fixed time intervals, and thus the rate of change in particle scores is proportional to the number of events and thus the amount of change perceived by the sensor. For a fast motion many events are generated in a short time interval, thus particle scores quickly adapt to the new state over time. For slower motions, when only few events happen in a longer time interval, particle scores are only changing slowly.

The resampling step basically transforms between the two representations of probability distributions mentioned earlier. The transformation is necessary to optimize the representation of the probability distribution by moving samples to areas with high probability. However as individual events carry only little information, this optimization step is only required after a sufficient amount of information has been gained. When assuming that the information content of each event is similar, which is indeed a good assumption as events are produced due to changes in the scene, it is reasonable to execute resampling only every n-th event.

The motion model step is similar to the classic particle filter, but with an interesting twist. For dynamic vision sensors the event rate depends on the linear and angular velocity of the tracked entity. As explained in §6.2.2, the amount of change in position and rotation can be estimated for an individual event. For an application where the pose of the sensor or a static object is tracked and for a constant depth D or a small depth-range, this allows to express the motion model independently from the concrete dynamics, e.g. from the maximum possible velocity. This is a major simplification as the dynamics of an object are in general difficult to estimate and may vary greatly.

The *Event-based Particle Filter* algorithm is summarized in alg. 7. The algorithm can be formulated with very few lines of codes due to the fact that most of the application specific complexity is represented in the motion model $\mathrm{MM}(\cdot\,|\,p_i)$ and the sensor model $\mathscr{P}(e_k\,|\,p_i)$. A suitable choice for the sensor model when the underlying dynamics are too complex is random diffusion: $\mathrm{MM}(\cdot\,|\,p_i) := \mathscr{N}(\cdot\,|\,p_i, \Sigma)$. The resampling step which is executed only every K-th event is not further specified in the algorithm. By using binary search it has a straight forward implementation with runtime $O(\log(n))$. Other implementations like a resampling wheel are even faster but give only approximative results. The algorithm can report a state estimate for every event by choosing for example the currently best particle. Different strategies like the weighted mean of the best particles are also possible. It would be best if the probability distribution could be used directly by the encapsulating application which further processes the state estimates.

In order to increase runtime performance particles are collected in small batches of size B and the event loop in algorithm alg. 7 is execute per batch. Motion model and decay constant have to be adapted accordingly. For a diffusion motion model using a normal distribution this change is simple. The distribution variance of the sum of several normally distributed random variables can be found as the sum of the individual variances. The score update also has to be changed as several events are processed together as one bit of information:

$$s_i = (1-\alpha)^B s_i + \frac{1-(1-\alpha)^B}{B} \sum_{j=0}^{B-1} P(e_{k+j}|p_i) \tag{6.30}$$

Eq. 6.30 follows by inspection when assuming that all events in the batch happen at the same time. Batching is solely used as a performance measure and a does not alter the nature of the *EB-PF* algorithm.

Algorithm 7 *Event-based Particle Filter* (*EB-PF*)

```
Require: $N$ number of particles
Require: $K$ number of events to wait until resampling
Require: $p_{\text{start}}$ initial state estimate
  for $i = 1 \to N$ do
      $p_i = p_{\text{start}}$
      $s_i = 1$
  end for
  for each event $e_k$ do
      if $k \equiv 0 \bmod K$ then
          resample($\{p_i\}, \{s_i\}$)
      end if
      for $i = 1 \to N$ do
          Sample new $p_i$ from MM($\cdot|p_i$)
          $s_i = (1-\alpha)\, s_i + \alpha\, P(e_k|p_i)$
      end for
      $i^* = \text{argmin}_{1 \le i \le N}\, s_i$
      report $p_{i^*}$ as current estimate
  end for
```

6.5 Application: Robot self localization (2D)

For a first example how the *Event-based Particle Filter* algorithm can be used in an embedded robotic system, *EB-PF* is used in a 2D robot self-localization scenario with a pre-build fixed map. A small robot with an *Embedded Dynamic Vision Sensor* mounted on top drives on a flat floor and observes features on the ceiling (see fig. 6.7). Due to the general uniform appearance of office ceilings, additional artificial features were attached onto the ceiling to provide enough tracking evidence for this experiment. The *Event-based Particle Filter* algorithm itself is not able to create a model of the object it is tracking, i.e. a map of the environment in this case. This drawback is addressed later in §7 and for now a pre-build map has to be provided manually. The ceiling map is taken as a photograph from the ground and converted into a black and white edge map suitable to the sensor model of the *eDVS* (see fig. 6.8).

As the robot is limited to driving on the ground, the system state Ω is the pose space of a two-dimensional object:

$$\Omega := \mathrm{SE}(2) = \mathbb{R}^2 \times \mathrm{SO}(2)\,. \tag{6.31}$$

In general SE(n) is the special Euclidean group which describes orientation preserving isometries of the n-dimensional space, i.e. exactly the possible motions of a rigid n-dimensional object. SO(n) is the special orthogonal group which describes orientation preserving rotations in n-dimensional space. For the two-dimensional case, SO(2) can be represented simply by an angle $\theta \in \mathbb{R}$ with the additional constraint that the angle ”wraps” after one full turn, i.e. θ and $\theta + 2\pi$ represent the same rotation. Thus a pose $p \in \Omega$ can be represented as $p = (x, y, \theta) \in \mathbb{R}^3$ where $(x, y) \in \mathbb{R}^2$ is the position and $\theta \in \mathbb{R}$ the rotation of the robotic entity driving on the ground.

In the following the two models used by the particle filter are discussed: The sensor model in §6.5.1 and the motion model in §6.5.2. Afterwards the system is evaluated in §6.5.3 using a simulation and ground truth data from an overhead-tracking system.

6.5.1 The sensor model

For a retina event $e = (e_u, e_v) \in \mathscr{R}$ and a pose $p = (p_x, p_y, \theta) \in \mathrm{SE}(2)$ one can compute a ray in world coordinates which starts at the sensor's optical centre and goes through the sensor pixel location e. Using eq. 6.1, the ray direction in camera coordinates for event e is:

$$\mathrm{gaze}(e) := \begin{pmatrix} e_u - c_u \\ e_v - c_v \\ f \end{pmatrix} \tag{6.32}$$

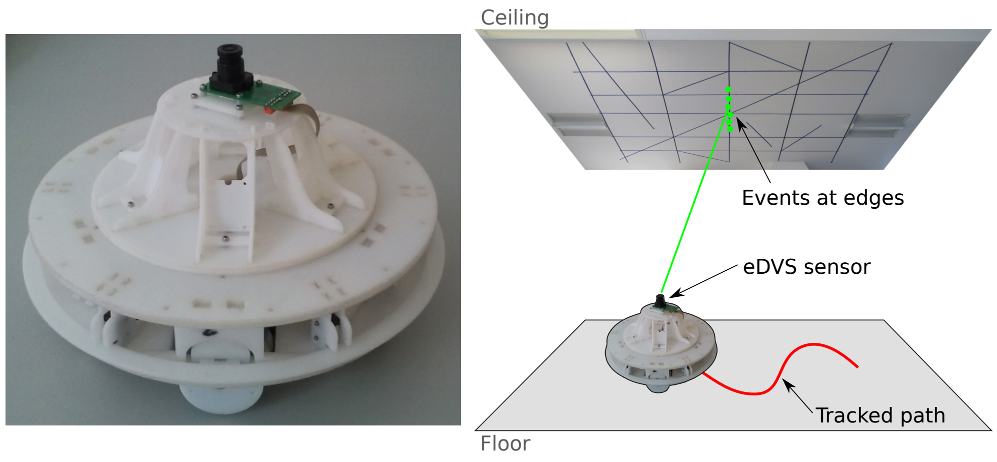


Figure 6.7: Application scenario for the *Event-based Particle Filter* algorithm. A robot drives on the floor and observes features on the ceiling. It localizes itself by using only the sparse event stream from the *eDVS* sensor.

The robot pose p defines the affine transformation from camera to world coordinates as:

$$T_p(v) := \begin{pmatrix} p_x \\ p_y \\ 0 \end{pmatrix} + R_z(\theta)\, v \tag{6.33}$$

where $R_z \in \mathbb{R}^{3\times 3}$ is the rotation matrix which rotates around the z-axis.

$$R_z(\theta) := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.34}$$

Together this yields the ray in world coordinates as

$$\text{ray}(e,p) := \left\{ T_p(\lambda\, \text{gaze}(e)) \mid \lambda \in \mathbb{R}_+ \right\} . \tag{6.35}$$

To evaluate how well an event e explains a hypothetical pose p, the minimal distance from $\text{ray}(e,p)$ to all features on the map $\mathscr{M}$ is computed:

$$f_{dist}(e,p) := \min_{s \in \mathscr{M}} d\left(\text{ray}(e,p), s\right) . \tag{6.36}$$

For this application the map $\mathscr{M}$ is realized as a dense grid which contains the manually provided map data – see fig. 6.8 for an example. As the map is in general only sparsely populated a more condensed representation as a set of lines or a sparse spatial data structure like an quadtree could be chosen.

With the distance objective function f_{dist}, the final sensor model function $\mathscr{P}(e\,|\,p)$ is defined as

$$\mathscr{P}(e\,|\,p) \propto \exp\left(-\frac{1}{2} \left(\frac{f_{dist}(e,p)}{\gamma\, W_{\text{trig}}} \right)^2 \right) . \tag{6.37}$$

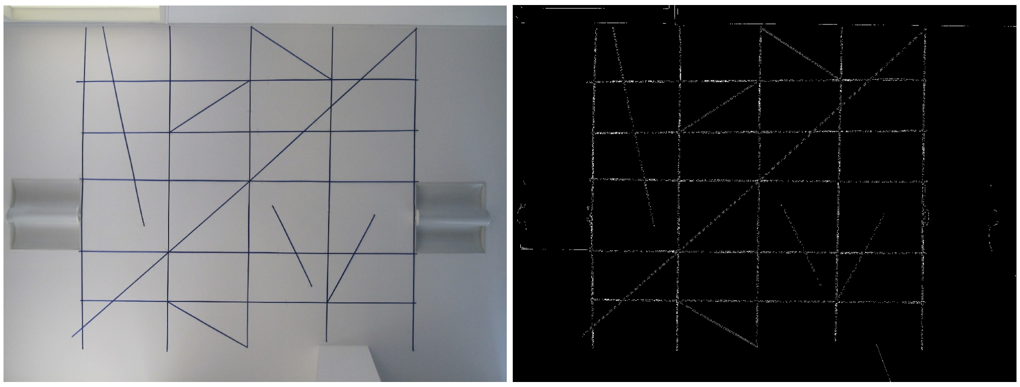


Figure 6.8: **Left:** Actual colour photo from the ceiling. **Right:** Corresponding edge map after Canny edge detector used as a ceiling map for event-based robot self-localization.

The normalization factor for the distance f_{dist} is expressed as a multiple of the distance W_{trig} corresponding to the size of one pixel of the retina projected onto the ceiling (see eq. 6.5). We chose $\gamma = 0.5$ to represent the fact that pixel events are accurate up to half a pixel. The use of the Gaussian function in eq. 6.37 is a common choice to model pixel discretization errors and a certain amount of measurement uncertainty.

As the retina sensor is pointed at the ceiling, is is reasonable to assume that all events occure only due to edges on the ceiling. We assume that the ceiling is flat, parallel to the ground and at a fixed height D above ground. All edges on the ceiling are known and carthographed in the grid map. In this scenario, the computation process can be simplified by caching the distance function f_{dist} and thus the sensor model function. The intersection of camera rays $\text{ray}(e,p)$ with the ceiling plane are found at $\lambda = \frac{D}{f}$. Thus the value of $\mathscr{P}(e\,|\,p)$ can be computed directly with just a simple lookup in the cached grid map.

6.5.2 The motion model

It is difficult to derive an accurate motion model for dynamic vision sensors as individual events may be due to rotation *or* translation and no prior assumption about the relative occurrence of translation and rotation can be made in general. A typical choice in such an ambiguous scenario with imperfect information is a random diffusion model. Here, a normal distribution $\mathcal{N}(\cdot\,|\,0,\Sigma)$ with zero mean and covariance matrix Σ is used to form the motion model and to distribute new particles around the existing particle positions:

$$\text{MM}(\cdot\,|\,p) := \mathcal{N}(\cdot\,|\,p,\Sigma) \text{ with } \Sigma = \begin{pmatrix} \Delta_x^2 & 0 & 0 \\ 0 & \Delta_x^2 & 0 \\ 0 & 0 & \Delta_\theta^2 \end{pmatrix}. \tag{6.38}$$

Δ_x is the moved distance (see eq. 6.9) and Δ_θ the rotated angle (see eq. 6.10) which happens in average for one singular event.

Note that the motion model does *not* depend on the elapsed time. For dynamic vision sensors the amount of movement is proportional to the number of generated events. Thus in a 2D scenario it is possible to directly estimate the possible movement due to one event.

The motion model MM_B for the optimized batch version can be derived as

$$\mathrm{MM}_B(p) = \mathcal{N}(p, B\Sigma) \tag{6.39}$$

This is due to the well known fact that the variance of the sum of two normal distributed variables is the sum of the two individual variances:

$$\mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)\ . \tag{6.40}$$

Odometry is not used to estimate the pose of the robot based on information from rotational encoders in the robot wheels. While odometry can be used successfully, it requires a thorough calibration of sensors and is prone to many errors like varying friction coefficients depending on the nature of the ground or even wheel slip, and noise and integration errors in general. Additionally it is not straight forward to provide good movement estimates from odometry for singular events as the time difference between events is in general very small and varies greatly depending on the actual speed of the robot.

6.5.3 Evaluation

The proposed *Event-based Particle Filter* algorithm is evaluated in two ways: with artificial simulated events for a manually given path and map, and with experimental data from the real sensor using ground truth data for evaluation. In both cases we assume that a robot is driving on the floor, that the sensor is mounted to point directly to the ceiling, which is $D = 2.59$ meters over the sensor, and that all event generating features are on the ceiling. Tracking was performed with a batch size of 3, a particle count of 50 and resampling after 100 events if not mentioned otherwise.

Fig. 6.9 (right) shows an example result from experimental data. In this figure the black dots corresponds to individual pose estimates after each resampling step. Using all pose estimates after each event is infeasible to plot as there are millions of events. The figure demonstrated that individual pose estimates are distributed along the path with a low mean deviation. This random deviation can be explained perfectly by the coarse resolution of the sensor. One pixel on the sensor corresponds to a distance of approximately 3 cm on the ceiling, i.e. $W_{\mathrm{trig}} = 0.03$ (given the fixed distance of $D = 2.59$ m). Due to the randomness in the computed path, a mean filter is used to smooth the path which is shown as a blue line. The ground truth path, which was either given for simulation or tracked with

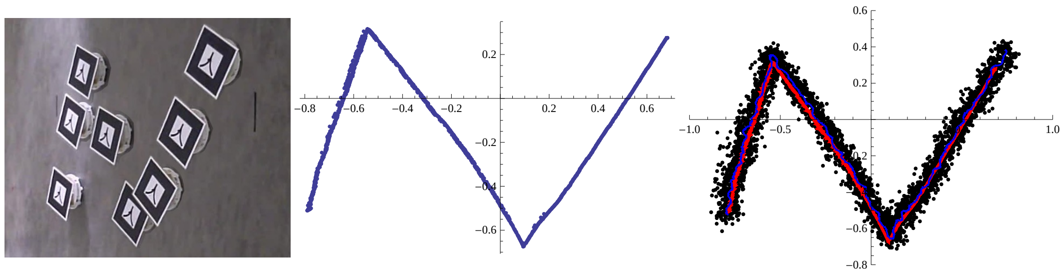


Figure 6.9: Driving robot with a marker at different positions along its path (left), captured ground truth data (middle), and tracking results of *EB-SLAM-2D* (right). Here black dots are path results for individual events.

an external marker-based tracking system, is plotted in red. Results show that the smoothed path is very close to the original path for results from simulation.

For the simulation a random map was generated consisting of several randomly placed lines and circles. Several paths were created manually and used to simulate event generation for the robot driving along the path. Event generation was performed by simulating each pixel independently and observing the change in illumination projected into the pixel using a model similar to the sensor model from eq. 6.37. Afterwards the tracker was executed using map and events to perform the event-based tracking *without* any knowledge about the original path. Fig. B.8 shows results for several different paths using a random map, artificially generated path and sensor simulation. For theses scenarios the average root mean square error in position was 0.6 cm.

To proof the very good quality of the *Event-based Particle Filter* in practical applications, experimental data is collected by driving a real robot manually over the ground and recording the actual movement with an overhead tracking system. The robot used in the experiments is a small wheeled robot (see fig. 6.7) controlled remotely by a human using a gamepad. In order to capture ground truth data from the moving robot a marker was installed on it and tracked using the ARToolKit [40]. Fig. 6.9 depicts the path of the robot driving on the floor and tracked ground truth data together with tracking results. An artificial line pattern was attached on the ceiling and a map was generated during a pre-processing step (see fig. 6.8). This map was used during particle evaluation in the objective function (see eq. 6.37). The special planar structure allowed an efficient caching of the objective for higher tracking performance. Fig. B.9 shows tracking results for experimental data. The conditions are similar to the simulation environment as the parameters for simulation have been chosen to match the real scenario as close as possible. For theses scenarios the average root mean square error in position was 5.4 cm.

In table 6.3 the mean runtime of the *Event-based Particle Filter* algorithm are reported for varying parameters. Results were measured on a normal single-core 2.53 GHz CPU. The event-based sensor normally produces less than 40000 events per second, thus even an unoptimized implementation with overhead for debugging and visualization can easily run in realtime. Additionally the memory requirements of *EB-PF* itself are very low: for each particle a state estimate and a weight needs to be memorized. This makes *EB-PF* well suitable for direct implementation on an embedded platform.

Table 6.3: Algorithm runtime for selected parameters

Batch Size	**Particle Count**	**Speed [events/s]**
	50	34400
1	100	18100
	250	7500
	50	74800
3	100	40400
	250	17200

7 Event-based SLAM

EB-SLAM is a novel simultaneous localization and mapping algorithm for dynamic vision sensors like the *eDVS*. The algorithm works completely event-based and makes efficient use of the sparsity of the event-based data stream.

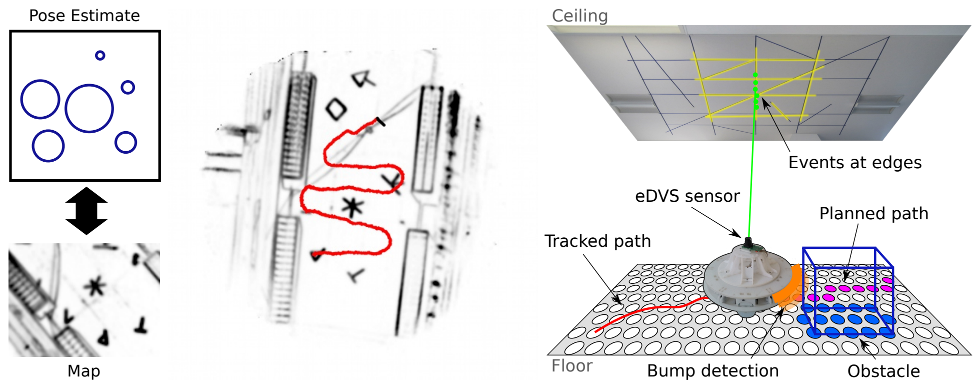


Figure 7.1: **Left:** Simultaneous localization and mapping is a chicken and egg problem which is fundamental to most robotic applications. **Middle:** Pixel events are enough for *EB-SLAM* to generate a sparse grid map and to track the robot trajectory in realtime. **Right:** *EB-SLAM* is suitable for embedded applications like autonomous exploration.

7.1 Simultaneous Localization and Mapping

The *Event-based Particle Filter* algorithm from §6.4 can be used for robot self-localization as demonstrated in §6.5 for a 2D scenario. However the localization within an environment is only possible if an adequate map of the environment is available. While in the previous chapter a pre-build map was used to demonstrate the capabilities of the *Event-based Particle Filter*, here the full problem of creating a map of an unknown environment while simultaneously tracking the pose, i.e. position and rotation, is considered.

Simultaneous localization and mapping (SLAM) is a hard problem due to several factors. On the one hand, sensor measurement errors limit the accuracy of pose estimates and in general a probabilistic approach is beneficial. Much worse, even small local errors can sum up globally and result in large deviations when the global structure of the map or the path is compared to ground truth. This problem can already be observed even on small scale, when the robotic system is driving in a circle and is only capable of observing a part of the whole environment at once – hence the name ”loop-closure” problem. Other challenges arise on the question how to efficiently represent a complex three-dimensional environment, and how to find methods which run fast enough with minimal resource requirements.

In the past SLAM systems used 2D laser scanners relying on laser distance measurement to create 2D maps of the environment. A typical example is the SICK Laser Measurement Sensor (SICK LMS) which has a very long measurement range and a wide field of view. The choice of laser scanners avoids the difficulty of estimating depth values from stereo camera setups and safes computation resources as distance measurements are directly available in hardware. With these sensors 2D ”bird view” maps of the environment were created which are suitable for navigating a robot on the ground.

This section will focus on the case of 2D SLAM where a robotic entity is driving on the floor while it tries to localize itself and detects obstacles in the environment. However the *EB-SLAM* algorithm is formulated in a general way which allows generalization to 3D. The case of full 3D SLAM, where the robot creates a map of the full 3D environment and in addition to driving on the ground can also move in all six degrees of freedom like a flying drone, is considered in the next chapter (§8).

In the following, two state-of-the-art algorithms for 2D SLAM are presented: FastSLAM [54] and ”Grid mapping with Rao-Blackwellized Particle Filters” [29]. The next section §7.2 will present the novel *Event-Based SLAM* (*EB-SLAM*) algorithm which uses only the sparse data stream from the *eDVS* sensor. This novel event-based approach requires the adaption of classical SLAM concepts to event-based vision, but also brings with it several key advantages like increased performance and the general benefits discussed earlier in §6.

FastSLAM

FastSLAM by Montemerlo, Thrun, Koller and Wegbreit [54] is a SLAM algorithm which uses an extended Kalman filter (EKF) to exactly model the probability distributions between the robot poses as a function of robot controls and landmark observations. The covariance matrix used by the EKF has a number of elements quadratic in the number of landmarks resulting in a high computational complexity which is one of the key limitations of an EKF. FastSLAM tackles this problem by recursively estimating the full posterior distribution over the robot pose and landmark locations by using an exact factorization of the posterior into a product of conditional distributions for landmark position and the robot path. It uses a tree structure for efficient representation of landmark estimations and individual Rao-Blackwellized particle filters (see next part) for the EKF. This decomposition yields a very efficient SLAM algorithm capable of handling as many as 50000 landmarks.

Grid mapping with Rao-Blackwellized Particle Filters

Grisetti, Stachniss and Burgard presented a method which uses a Rao-Blackwellized particle filter to solve the SLAM problem [29]. Particles filters [37] are a powerful sampling-based tool to solve Dynamic Baysian Networks which in comparison to a Kalman filter or HMM filter are also feasible for non-linear, non-Gaussian models. The Rao-Blackwellised particle filter [22] uses importance sampling to improve particle spread and is an improvement over the classic particle filter. When using particle filters in SLAM a probability distribution over the joint pose and map space has to be estimated. By applying the idea of factorization, the problem can be split into first estimating the pose based on a fixed map, and then in a second step estimating the map based on a fixed pose. This ”chicken and egg” approach to SLAM is one of the key methods used in SLAM today. In [29] this technique is used to efficiently map a large outdoor environment using laser distance measurement sensors. Additionally the odometry of the robot, i.e. information about actual wheel rotation and commands sent to the robot, are used in factored resampling to further reduce the number of particles and improve tracking results.

7.2 Event-based SLAM

7.2.1 Event-based simultaenous localization and mapping

Simultaneous localization and mapping is a coupled problem where a state estimate needs to be build over the joint space of possible poses and possible maps. The problem of mapping the environment is additionally complicated by the fact, that a dynamic vision sensor has very special properties which have not been considered in SLAM before:

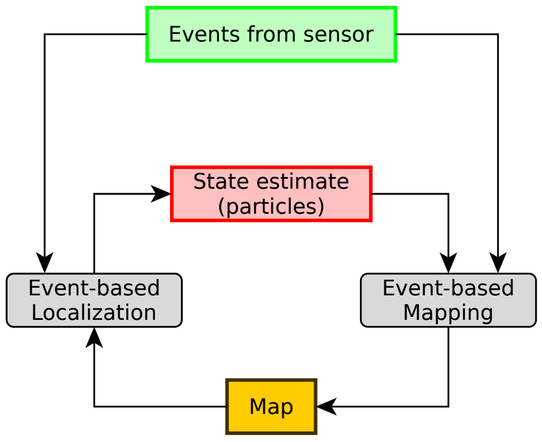


Figure 7.2: Event-based Simultaneous Localization and Mapping for dynamic vision sensors uses the *Event-based Particle Filter* for localization and a novel event-based mapping method.

- Information is delivered as a sparse data stream in the form of singular pixel events. This information has to be integrated correctly in order to construct a meaningful environment map.

- Pixel events are only generated through movement. This implies that measurements about the environment can not be repeated without changing the pose of the sensor.

- Individual events have a μs time resolution. The SLAM method should be suitable for fast moving systems like flying robots. Ideally all computations should be executed per event to allow low latency while additionally guaranteeing low computational requirements.

In the following a novel SLAM algorithm for dynamic vision sensors like the *eDVS* – *Event-Based SLAM* (*EB-SLAM*) – is presented which will provide solutions for all three problems. *EB-SLAM* creates a map which describes event generation probability, works on individual events and runs several times faster than realtime on a desktop computer. The method has been presented at a peer-reviewed robotics conference by the author [82].

EB-SLAM computes localization estimates and map updates alternately for each incoming event (see fig. 7.2). For each event, the *EB-PF* algorithm uses the last environment map to update a probability distribution for the robot pose represented by a set of weighted particles. Theses particles can be used to compute an actual pose estimate. Particles and the pose estimate are used in a second step to incrementally build an environment map which represents the probability that a mapped location generates an event under movement.

EB-SLAM is formulated in an general mathematical way which applies to 2D and 3D scenarios. However due to the complexity of three-dimensional environments several questions regarding 3D SLAM are postponed until §8. To demonstrate the capabilities and robustness of *Event-Based SLAM*, in §7.3 the method is applied in the two-dimensional scenario from §6.5. Results for created pose trajectories and maps are reported for several scenarios and compared against ground truth data. In §7.4, *Event-Based SLAM* is used for autonomous exploration and where it produces accurate and robust results over a long timespan of 10 to 30 minutes.

7.2.2 State space, map space and projection

For self-localization the state space is the special Euclidean group which describes the linear and rotational motions of a rigid body, i.e. $\Omega = \mathrm{SE}(n)$. The two interesting cases are $n = 2$ and $n = 3$. The two-dimensional scenario has already be explored in §6 and consists of a robot driving on a flat ground and observing features on the ceiling above it. The three-dimensional scenario could be the case of a flying robot which can move freely in three-dimensional space and thus has to map the complete 3D space while trying to track its own path. Of course mixed scenarios exist, most notably the ”2.5D” case where a robot drives on a flat ground but explores a three-dimensional world, or even simpler when only a specific heigh ”slice” is used for mapping. The latter scenario is the most widely application of SLAM and is often realized with a laser range finder.

The mapped space Γ can be two-dimensional as in §6.5 where a map of the flat ceiling is created. A flat surface can be easily parametrised and is thus represented by $\Gamma = \mathbb{R}^2$. For computations, the map can be represented by a discretisation of all possible positions in a region of interest. The resolution of the discretisation defines the maximal possible accuracy of results. But depending on the resolution of the sensor, there is a reasonable lower bound of grid cell dimensions. For the three-dimensional space, i.e. $\Gamma = \mathbb{R}^3$, the same principals apply. The realization for concrete computations is a three-dimensional grid of voxels. However the required memory is already proportional to the number of grid cells in one direction to the power of three. Even a coarse grids with 256^3 voxels already uses 64 MB.

A crucial component in visual SLAM is the projection of a feature point at a specific position in the world onto the two-dimensional image plane of the vision sensor. While in the two-dimensional case as investigated in §6.5 this ”projection” was just a translation and scaling, for the general three-dimensional case a projection looses the depth information.

A point in camera coordinates is projected by using the pinhole camera model in eq. 6.1 together with several camera parameters which are assumed to be known. The transformation from world to camera parameters is defined by the pose of the vision sensor $p \in \Omega$. In the following the function μ will be used to describe the projections of a position $u \in \Gamma$ in the map space onto a pixel coordinate on the image sensor:

$$\mu : \Gamma \times \Omega \to \mathscr{R},\, (u,p) \mapsto \mu(u,p) \tag{7.1}$$

μ consists of a transformation of u from world into camera coordinates followed by a projection.

For a particle filter algorithm the sensor model describes the probability that a given measurement has occurred given the current system state. In the case of SLAM for dynamic vision sensors, this is the probability that a given event $e \in \mathscr{R}$ occurs given a robot pose $p \in \Omega$. This probability has to be computed by ”back-projecting” the event into the world and checking the corresponding map location $\mu^{-1}(e\,|\,p)$:

$$\mathscr{P}(e\,|\,p) \propto \mathscr{M}(\mu^{-1}(e\,|\,p))\,. \tag{7.2}$$

Here a huge problem arises due to the nature of projections. While the projection μ itself is well-defined, in general, it looses information. Thus the back-projection μ^{-1} is not a well-defined problem any more, as one specific event could be due to several possible locations in the map:

$$\mu^{-1} : \mathscr{R} \times \Omega \to \mathrm{Pow}(\Gamma),\, (e,p) \mapsto \mu^{-1}(e,p)\,. \tag{7.3}$$

Here Pow(Γ) is the power set of Γ and indicates that μ^{-1} in general returns a subset of Γ. This set of possible locations is meant by $\mu^{-1}(e\,|\,p)$, and a concrete definition of μ has to scope with the problem of evaluating the map for a set of points instead of a singular point.

A possible approach would be to marginalize over all points in the set:

$$\mathscr{M}(\mu^{-1}(e\,|\,p)) := \int_{\mu^{-1}(e\,|\,p)} \mathscr{M}(v)\,\mathrm{d}v\,. \tag{7.4}$$

Another approach will be investigated in §8.

7.2.3 Event-based mapping

The *Event-based Particle Filter* algorithm can be used for robot self-localization by providing an adequate method to model the environment. For the sensor model in eq. 7.2 the probability that a specific location could have generated an event is required. This poses the question when points in the map generate events. Due to the nature of dynamic vision sensors, events are generated when the illumination of a pixel increases or decreases over a specific threshold. For a static environment, this happens when the sensor observes areas of different intensity while moving. Whenever the back-projection of a pixel moves from a light area to a dark area, or the other way around, it generates an event. This is most often the case at geometry edges or distinguished texture features.

At this point it is important to investigate several issues of such a simple event generation model. First, the event generation probability depends on the

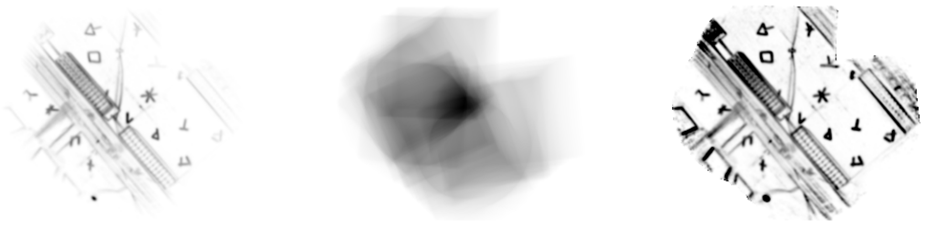


Figure 7.3: **From left to right:** The occurrence map (see eq. 7.6), the normalization map (see eq. 7.7) and the final map (see eq. 7.5).

direction into which the sensor is moving. For example if the sensor moves in parallel to a very long line, it would create almost no events as each pixel repeatedly either hits the line or does not hit it, no matter how far the sensor has moved into this direction. If on the opposite the sensor would move orthogonal to a line feature, many events would be generated as all pixels traverses the line in parallel. Another issue which arises in three-dimensional environments is occlusion. Due to different relative object positions, features are visible from one viewpoint but not from another. These phenomena complicate the modelling of the environment map and have to be considered accordingly in the sensor model and the update rules for the map. The three-dimensional case will be further investigated in §8.

The value $\mathscr{M}(u)$ stored in the map should indicate the probability that a location $u \in \Gamma$ creates an event. Here a very important property of event generation comes into play. There are no measurements which indicate that a specific location should *not* generate events. Thus if a pixel location has generated an event once this does not mean it always creates an event and the corresponding probability should be 1. Additionally, one needs to consider how often the pixel *could have* generated an event. The first quantity, how often a location has actually generated an event, is stored in the **occurrence map** $\mathscr{O}$. The second quantity, how often a location could have generated an event, is stored in the **normalization map** $\mathscr{Z}$. Together this gives the required event generation probability for the map $\mathscr{M}$:

$$\mathscr{M}(u) = \frac{\#\text{ of occurred events for } u}{\#\text{ of possible obervations for } u} =: \frac{\mathscr{O}(u)}{\mathscr{Z}(u)} \tag{7.5}$$

Fig. 7.3 depicts an example for occurrence and normalization map and the resulting final map probability. It can be seen how regions with few events are normalized to get equally likely map probabilities for the whole map space.

For the computation of the occurrence map $\mathscr{O}$ each event is back-projected into the corresponding location $\mu^{-1}(e|p)$ by using the current position estimates $p_i \in \Omega$ provided by the particle filter. To model a certain degree of measurement uncertainty a Gaussian model is used to diffuse the current event $e^{(k)} \in \mathscr{R}$ over nearby locations in the map. The occurrence map can be computed iteratively as:

$$\mathcal{O}^{(k)}(u) = \mathcal{O}^{(k-1)}(u) + \sum_{i=1}^{n} {s_i}^{(k)} \mathcal{N}\left(u \,\middle|\, \mu^{-1}(e^{(k)} \mid {p_i}^{(k)}), \sigma\right), \mathcal{O}^{(0)} = 0\,. \tag{7.6}$$

Here the standard deviation σ is a multiple of the size W_{trig} of a sensor pixel when back-projected into the map. As the occurrence map uses back-projection it has to scope with the problem that the inverse μ^{-1} of the projection function μ may not be well-defined and create a whole set of points in the map space.

The particle weights s_i used to model the probability distribution over the state space are used to weight the contributions of individual particles based on their confidence. In equation eq. 7.6 all particles are used to update the map but as normally done in SLAM algorithms, it may be beneficial to only use the top 10% of particles or a mean of the best particles.

The number of possible observations for the normalization map $\mathcal{Z}$ can be computed by considering again the special properties of the event-based sensor. Assuming a strong edge in the perceived light intensity, the sensor will generate one event for every pixel which passes over this edge. Thus the fractional number of possible generated events for a map location $u \in \Gamma$ is proportional to the length of its path on the sensor in pixel coordinates. Given a state estimate $x \in \Omega$, the corresponding fractional pixel position on the sensor using the projection function μ is computed. Note that this position does not necessarily lie inside the sensor boundaries as not all areas of the map are visible by the sensor at all times. If a map point is not visible by the sensor under the current or previous state estimate the normalization map is not updated at this map location. Otherwise the normalization map is computed as

$$\mathcal{Z}^{(k)}(u) = \mathcal{Z}^{(k-1)}(u) + \|\mu(u \mid {p_*}^{(k)}) - \mu(u \mid {p_*}^{(k-1)})\|, \mathcal{Z}^{(0)} = 0\,. \tag{7.7}$$

p_* denotes the expected state which is computed as the weighted mean of the whole particle set provided by *EB-PF*. Due to noise in the expected state and the high rate at which events are generated by the sensor, it is sensible to update the normalization map only periodically and not for every event.

7.2.4 The *EB-SLAM* algorithm

The *EB-SLAM* algorithm is summarized in alg. 8. It starts by initializing the environment maps $\mathcal{O}$, $\mathcal{Z}$ and $\mathcal{M}$ for the desired region of interest. Maps are set to 0 as no prior knowledge is available and required. In a concrete implementation, the maps can be easily realized as dense grid-maps with a fixed resolution. For a 2D map a reasonable choice for the resolution is in the order of W_{trig}. One can also chose an optimized spatial data structure like a quadtree or a kd-tree to optimize the memory footprint. While this is beneficial for 3D maps, for a 2D map a dense grid map provides better performance as it has constant read and write access

Algorithm 8 *Event-Based SLAM (EB-SLAM)*

```
▷ Initialize
$\forall u \in \Gamma : \mathcal{O}(u) = 0, \mathcal{Z}(u) = 0, \mathcal{M}(u) = 0$
$\forall 1 \le i \le N : p_i = 0, s_i = 1, p_* = 0$
▷ Run
for each new event $e$ do
    ▷ Update particles
    for $i = 1 \to N$ do
        $p_i = \mathrm{MM}(p_i)$
        $s_i = (1-\alpha)\, s_i + \alpha\, \mathcal{M}(\mu^{-1}(e \mid p_i))$
    end for
    Resample if necessary
    ▷ Compute current pose estimate
    $p_{**} = p_*$
    $p_* = \frac{\sum_{i=1}^{N} s_i\, p_i}{\sum_{i=1}^{N} s_i}$
    ▷ Update map
    $\forall u \in \Gamma : \mathcal{O}(u) = \mathcal{O}(u) + \mathcal{N}\left(u \mid \mu^{-1}(e \mid p_i), \sigma\right)$
    $\forall u \in \Gamma : \mathcal{Z}(u) = \mathcal{Z}(u) + \left\| \mu(u \mid p_*) - \mu(u \mid p_{**}) \right\|$
    $\forall u \in \Gamma : \mathcal{M}(u) = \begin{cases} \frac{\mathcal{O}(u)}{\mathcal{Z}(u)} & \mathcal{Z}(u) > 0 \\ 0 & \text{otherwise} \end{cases}$
end for
```

compared to spatial data structures which have a logarithmic runtime based on the size of the map.

As no prior knowledge is provided to the algorithm, no special coordinate system is given. During the first steps, *EB-SLAM* chooses an arbitrary coordinate system for the map space Γ. This implies that the initial particles can all be set to 0 which indicates the origin of the coordinate system chosen by the algorithm. It may be advantageous to provide a small amount of noise to the initial state estimates. Particle weights are initialized to 1 to start information integration (see §6.3).

After initialization, the two steps of localization and mapping are iterated for each incoming event. First the particle set is updated accordingly to *EB-PF* explained in §6.4. The motion model is executed for each particle state estimate – here it is simply a random diffusion – and the sensor model is used to update the particle weight. The second step uses the particle set to update the three maps required for mapping.

Theoretically, updating the occurrence map $\mathcal{O}$ and the normalization map $\mathcal{Z}$ requires an update of every grid cell in the map. In practice this can be optimized in two ways. For the occurrence map, the shape of the Gaussian functions allows to update only a small region within reach of the Gaussian kernel. The kernel itself is identical for all events and can thus be precomputed. In comparison, the update of the normalization map does not have a local character. However, here the defining factor is the length of the path described by sensor pixels in the map. This path is very short for only one individual events thus the computation can be delayed until a sufficient number of events have occurred. Moreover the normalization map only needs to be updated for the part of the map which is currently visible. The update of the final map $\mathscr{M}$ can be optimized accordingly.

A final remark should be made about the interaction between localization and mapping. The computation of the motion model diffusion to update localization estimates depends on the number of effective "feature" points G_0 which are currently visible (see eq. 6.9, eq. 6.10 and eq. 6.38). G_0 can be estimated from the map by integrating over the map $\mathscr{M}$ as the map exactly described the probability of event generation. A low number for G_0 leads to a higher diffusion, which in turn leads to higher values for the normalization map. This results in low values for $\mathscr{M}$ which decreases G_0 further. This can lead to a self-reinforcing loop which provides very poor location estimates. The problem can be avoided by choosing a good starting value for G_0 and reducing the influence of $\mathscr{M}$ on it, however further investigation is required to tackle this problem.

7.3 *EB-SLAM* in 2D

The *EB-SLAM* method is applied to the two-dimensional scenario which was introduced in §6.5 as an application for the *EB-PF* algorithm (see fig. 7.4). A small embedded robot (see fig. 6.7) drives on the flat floor and has an *eDVS* sensor mounted on top which observes features on the ceiling. In comparison to §6.5 where the map was pre-build manually and provided to the algorithm as is, now the map is initially empty, and created and refined autonomously by *EB-SLAM*. The map is build while the robot is driving around only by using the sparse data stream of pixel events provided by the *Embedded Dynamic Vision Sensor* sensor.

As the robot does not have any high-level reasoning for selecting targets or planning paths it was manuall controlled by a human using a gamepad – an application in autonomous exploration is presented in §7.4. The robot trajectory was tracked externally by the marker-based motion capture system OptiTrack V100:R2. Simultaneously the events generated by the *Embedded Dynamic Vision Sensor* sensor were transmitted over WiFi to a host computed and stored for further processing. The method was tested for various paths and a dataset of 40 trials was recorded. The *EB-SLAM* algorithm was executed afterwards on the recorded data, and the tracked trajectory and the created map was recorded over time. Fig. 7.7 shows the development of trajectory and map for one example from the dataset.

To evaluate the quality of the *EB-SLAM* algorithm, the tracked trajectory was compared against ground truth data. As the coordinate systems is chosen randomly by the mapper and does not correspond to the coordinate system used by the external tracker, the tracked path and the ground truth path needs to be

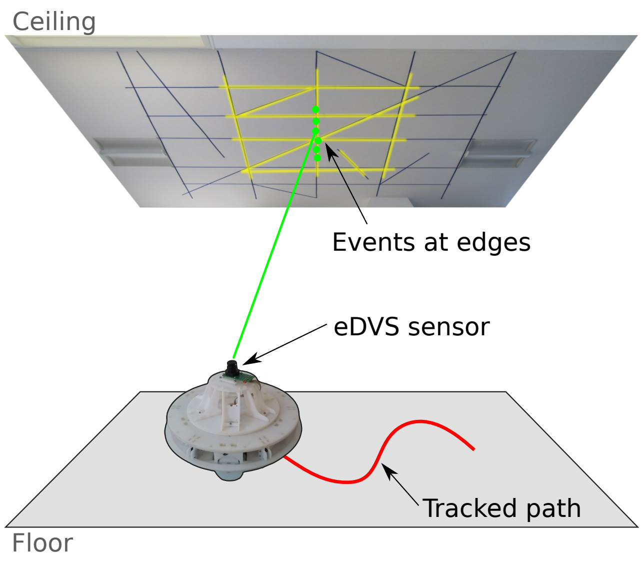


Figure 7.4: Individual events are processed one after another to track the robot while simultaneously creating a map of the environment.

aligned. The alignment was done automatically be minimizing the root mean square error between temporal corresponding points from the two paths. As the path from the ground truth system has a much lower resolution, linear interpolation was used to compute corresponding points for the ground truth path. In addition to a spatial alignment, the data also needs to be aligned in time as the external tracker and event generation could not be started simultaneously with sufficient accuracy. This was achieved by introducing and optimizing a simple constant time offset between paths.

The spatial-temporal alignment process does not weaken the significance of the evaluation. To required a shift in translation, rotation and time is perfectly reasonable as the corresponding coordinate systems can be chosen arbitrarily. The important point is not the global correspondence between the two trajectories but the correct relative consistency of their points. Errors in mapping and localization manifest as sudden turns in the path, a relative shift over time, or general disorientation. For ill chosen parameters, especially when the particle count is too low, all theses phenomena can be observed. Theses errors can not be compensated by a global trajectory alignment. For reasonable parameters the method provides very good results and the final root mean square error after alignment is in the order of several centimetres.

Fig. 7.5 shows an example path from the dataset and more results can be seen in fig. B.10 and fig. B.11. To the left the final map $\mathscr{M}$ and the corresponding path are displayed. The middle column shows a comparison of the tracked path and ground truth after alignment. The absolute error in translation and orientation is reported over time, i.e. over the number of events, in the right column. Results show, that the error is comparable to W_{trig} which was 0.05 m in this scenario.

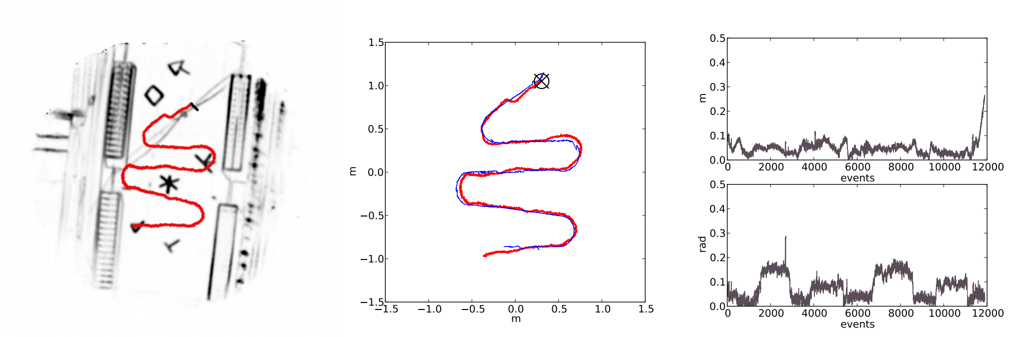


Figure 7.5: **Left:** Map and path as created by our method. **Middle:** Trajectories resulting from our method (red) and the external tracking system (blue). The trajectory starting point is marked with a X. **Right:** Positional and rotational error over number of resamples.

Table 7.1 shows an overview over the root mean square error in position and rotation over the whole dataset for several parameters. Additionally the failure rate is reported, i.e. the number of times the algorithm could not successfully create a reasonable path and diverged tremendously from the actual path. The processing speed is denoted in events per second (e/s) which gives the number of events *EB-SLAM* can process per second on a normal single-core desktop computer. For more than 10 particles the RMSE is very good overall and the number of failures is under 10%. A closer inspection reveals that most failures are due to invalid assumptions about G_0 and the amplification problem explained previously in §7.2.4.

In addition to the path the generated map should be evaluated as well. Fig. 7.6 shows a manual photo from the ceiling and the map generated by *EB-SLAM*. This side-by-side comparison demonstrates that both maps coincides and that the created map captures the intensity edges of features on the ceiling. The overlay shown to the right in fig. 7.6 strengthens this impression. Note that the maps where aligned manually, as again, the coordinate system origins are arbitrary.

The examples shown so far visualize only the final results of map and trajectory at the end of the movement. To demonstrate how the map is created over time,

Table 7.1: Positional and rotational root-mean-square error (RMSE), failure rate and processing speed for a varying number of particles.

Particles	RMSE pos.	RMSE rot.	Failure rate	Runtime
5	35.4 cm	51.2°	18/40	87800 e/s
10	5.9 cm	5.5°	5/40	80700 e/s
25	6.0 cm	5.5°	4/40	65600 e/s
75	6.0 cm	5.4°	3/40	38800 e/s

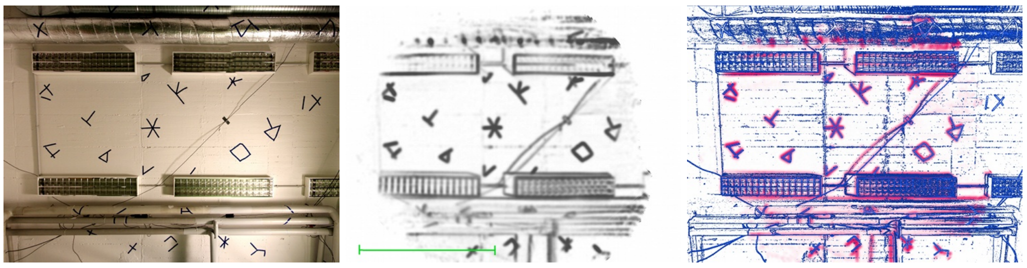


Figure 7.6: **Left:** Photo of the ceiling. **Middle:** Resulting map from our method. Darker spots indicating a higher likelihood of events. The green scale bar indicates the size of the field of view of the sensor on the ceiling (ca. 2 meters). **Right:** Overlay of our map (magenta) and the edge map of the ceiling photo (blue).

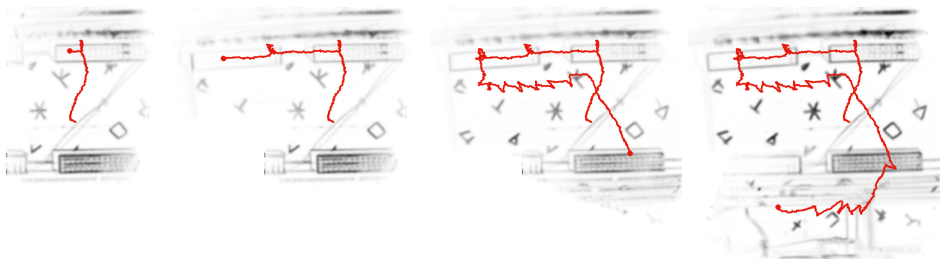


Figure 7.7: Example for map and path generation over time. In this scenario the robot moved on its own while exploring the environment. The wiggly parts of the trajectory indicate that the robot hit an environment obstacle on the ground.

fig. 7.7, fig. B.12 and fig. B.13 show a time series of maps and corresponding trajectories. The examples are taken from an application of *Event-Based SLAM* in autonomous exploration which is presented in more detail in §7.4.

7.4 Autonomous exploration with *EB-SLAM*

To demonstrate the capabilities and accuracy of the *EB-SLAM* algorithm, it was applied in an autonomous exploration scenario [32] depicted in fig. 7.8. Here the path is no longer given by a human operator, but chosen automatically and autonomously by the robot itself. It has to be emphasized that no external tracking was used and that self-localization was completely provided by *Event-Based SLAM* which observes features on the ceiling. To detect objects on the ground a ”bump sensor” is used. The robot is surrounded with a 360 degree ring of contact sensors which detects if the robot is touching an object. In this scenario it is not possible to detect objects with the *eDVS* sensor as no three-dimensional reasoning is available.

The governing principal for path selection is the maximization of knowledge over the environment. For successful exploration the robot has to gather two kinds of information. On the one hand, it has to observe features on the ceiling to achieve good self-localization. The normalization map $\mathcal{Z}$ (see eq. 7.7) is a

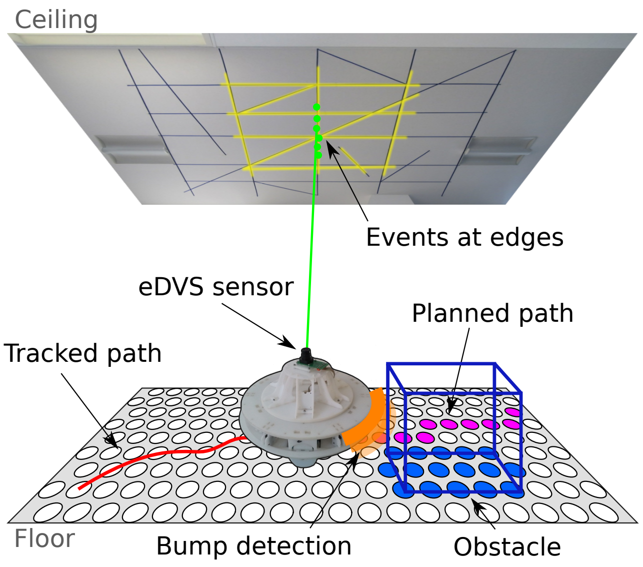


Figure 7.8: The autonomous exploration scenario used to demonstrate the capabilities of *EB-SLAM*. The robot localizes itself using features on the ceiling while detecting obstacles using a simple bump sensor.

suitable measure which indicates how much visual cues have been gathered for a given area. To avoid confusion, $\mathscr{I}$ is called ”localization exploration map” for this section. On the other hand, the robot needs to find obstacles to be able to plan efficient paths. As a contact sensor is used to detect objects, the robot has to drive to each position in the world to determine if it is passable or not. This information is gathered in the ”obstacle exploration map” $\mathscr{E}$.

To simplify the computation of the optimal path, all maps are discretised to a hexagonal grid. Each grid node has six neighbours and the robot plans its path by computing a list of neighbouring grid nodes. The hexagonal grid is preferred over a rectangular grid as all neighbours of a node have equal distance in a hexagonal grid.

The exploration map $\mathscr{E}$ is computed over time and updated each time the robot moves to a new grid cell. The grid cells which are currently covered by the robot are computed and the corresponding values in the obstacle exploration map are incremented to indicate that theses cells have been explored. Thus the higher the value in $\mathscr{E}$, the more often a node was already explored and marked as passable terrain. If the robot bumps into an object, it stops and marks the grid cells located in driving direction as impassable. To avoid excessive bumping, additionally neighbouring cells in a small cone-shaped corridor in driving direction are marked as impassable.

The aim of exploration is the maximization of knowledge over the environment, thus the maximization of the localization exploration map $\mathscr{I}$ and the obstacle exploration map $\mathscr{E}$. To fuse theses two kinds of information, each map is first normalized to distinguish between areas which are explored above average and

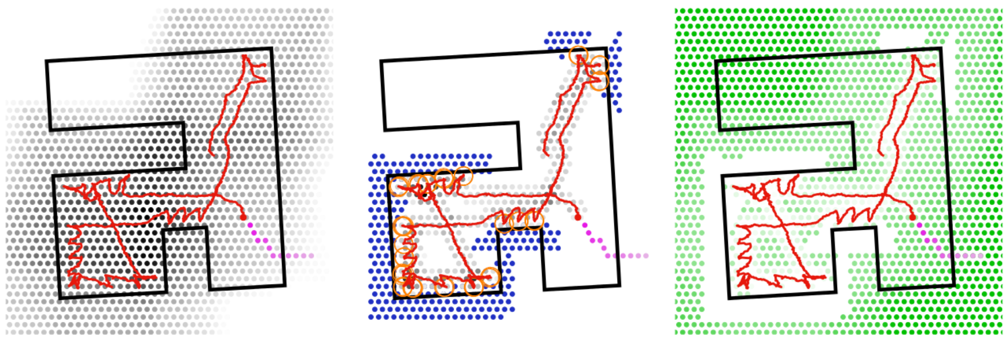


Figure 7.9: Ground truth room layout (black lines), actual robot path (red) and planned robot path (magenta). **Left:** Nodes are coloured based on the localization exploration map $\mathscr{I}$ (darker means higher). **Middle:** Nodes are coloured based on the obstacle exploration map $\mathscr{E}$ (darker means higher) and blocked nodes are marked in blue. The detected bumps are marked in orange. **Right:** Nodes are coloured based on the expected information gain $\mathscr{G}$ (darker means higher).

those which are explored below average. The normalization M_0 of a map M is chosen as:

$$M_0(v) := \frac{M(v) - \text{mean}(M)}{\max(M) - \min(M)} \tag{7.8}$$

here $\max(M) := \max_{u\in\Omega} M(u)$ is the maximum of M, $\min(M) := \min_{u\in\Omega} M(u)$ is the minimum of M and $\text{mean}(M) := \frac{1}{|\Omega|}\sum_{u\in\Omega} M(u)$ the mean value of M over the complete domain Ω. The normalization provides a scale invariant measure for the gained information which additionally does not required a target goal value to indicate the end of the exploration process. Instead the exploration continuously increases the knowledge over the environment.

The information of the localization and the obstacle exploration maps are fused into the ”information gain” map $\mathscr{G}$ using the normalization from eq. 7.8 as follows:

$$\mathscr{G}(v) := w_{\mathscr{I}}\, L_\alpha(-\mathscr{I}_0(v)) + w_{\mathscr{E}}\, L_\alpha(-\mathscr{E}_0(v)) \tag{7.9}$$

where L_α is the logistic function defined as

$$L_\alpha(x) := \frac{1}{1 + e^{-\alpha x}} \tag{7.10}$$

and $w_{\mathscr{I}}, w_{\mathscr{E}} \in \mathbb{R}_+$ are weighting factors. The slope factor α specifies how much areas with poor exploration are preferred over areas with above-average exploration. The higher α the harder the distinction between the two classes which results in a more aggressive behaviour to move to unexplored areas. For the experiments the values $w_{\mathscr{I}} = w_{\mathscr{E}} = 1$ and $\alpha = 10$ were chosen.

The path is planed by the robot by trying to maximize the gain $\mathscr{G}$ over a path which is as short as possible. Additionally the path should not lead through nodes which are marked as impassable. To plan an optimal path a breadth first algorithm

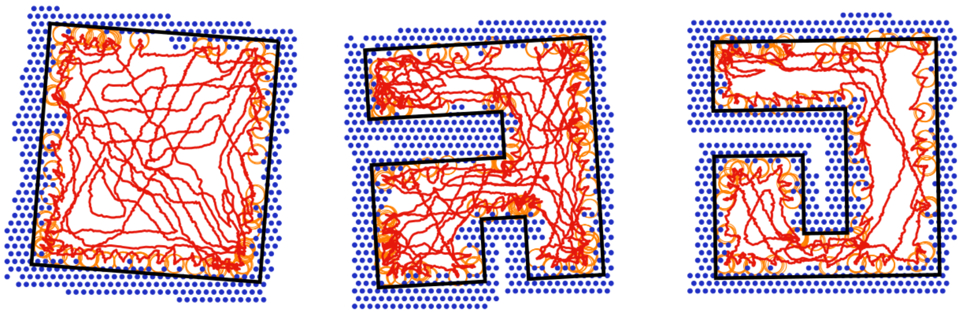


Figure 7.10: Chosen trajectory (red), bumbs (orange), impassable nodes (blue) overlayed with a manually created room layout (black) for three different room layouts.

with a search heuristic similar to A* is used. The node at the current robot position is expanded and nodes with a high value of gain per path length are explored first. Here the relative gain value plays the role of the heuristic which in A* leads to the desired goal. As no goal is given, the path is searched for a given amount of time or until a maximum number of nodes have been visited.

The computation of $\mathscr{G}$ is visualized in fig. 7.9. The two exploration maps are shown together with the robot path. The expected gain is visualized to the right, and it is visible how already explored nodes or nodes which are marked as blocked are assigned a lower value than areas which are unexplored. The corresponding path chosen by the robot is marked in magenta and leads out into the unknown as fast as possible.

The proposed exploration method was used in several indoor room scenarios of which three are shown in fig. 7.10. The robot has successfully found the room boundary and created a map of the room. The red path tries to cover as much of the room as possible to rule out the possibility of additional objects scattered throughout the room.

Fig. 7.11 shows how the robot explores a previously unknown room over time. The selected path tries to lead the robot into unknown areas to maximize the gain of information over the environment while avoiding obstacles.

In opposite to §7.3, the *EB-SLAM* algorithm was executed in realtime to provide realtime localization and mapping capabilities for the robot. All computations were executed on a normal desktop computer and the stream of pixel events and control commands were send from and to the robot over a wireless connection. The software used for tracking has a rich graphical user interface for debugging and observing the behaviour of the robot. However as *EB-SLAM* runs several times faster than realtime on a desktop computer and the code for autonomous exploration only adds simple tasks like maintaining the obstacle exploration map and replanning a path if a bump occurs it should be possible to implement the algorithm on an embedded, low-cost computation platform.

An example platform which should be able to handle the computations could

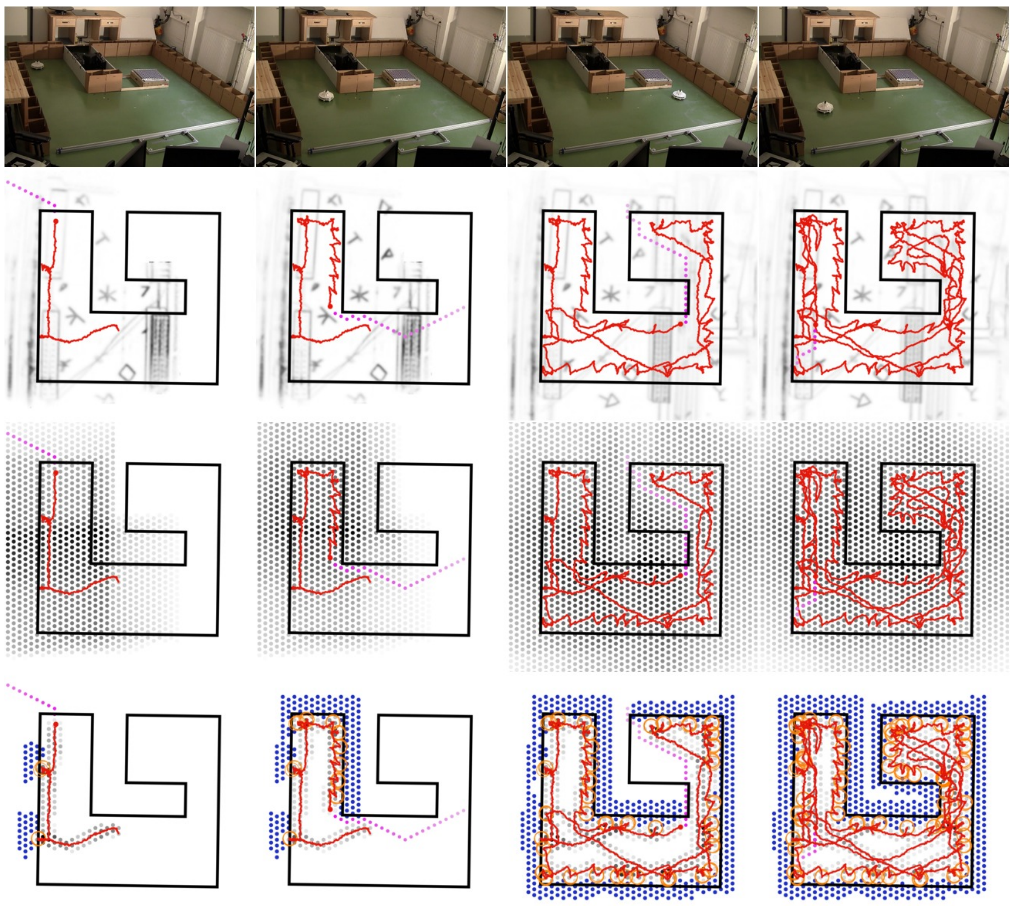


Figure 7.11: **Top row:** Images from an external camera for demonstration of the general scenario. **Other rows top to bottom**: Development over time of the feature map $\mathscr{M}$, the localization normalization map $\mathscr{Z}$ and the obstacle exploration map $\mathscr{E}$. In each image the chosen trajectory (red), the planned path (magenta), bumps (orange), impassable nodes (blue) and an overlayed manually created room layout (black) are displayed.

be a Mini-PCs like the Rikomagic MK802IV. This Mini-PC can be set up with a Linux operating system and has astonishing specifications: a Rockchip RK3188 1.8GHz quad-core Cortex-A9 with 2 GB RAM in a small casing with dimensions $90 \times 40 \times 13$ mm weighting less than 40 gram in total. The power consumption is less than 5 Watts (5 Volt) which allows operation with a small lithium polymer battery pack. A 2000 mAh two cell pack (7.4 Volt) which has approximative the same size as the Mini-PC and weights 100 gram would last for more than four hours.

8 Event-based 3D SLAM

In this chapter a dynamic vision sensor will be combined with an active depth-sensor to form the *D-eDVS* sensor which provides a stream of 3D point events. This sparse stream of 3D points will be used in the novel *Event-Based 3D SLAM* algorithm which has a very low computational footprint while at the same time provides very good results.

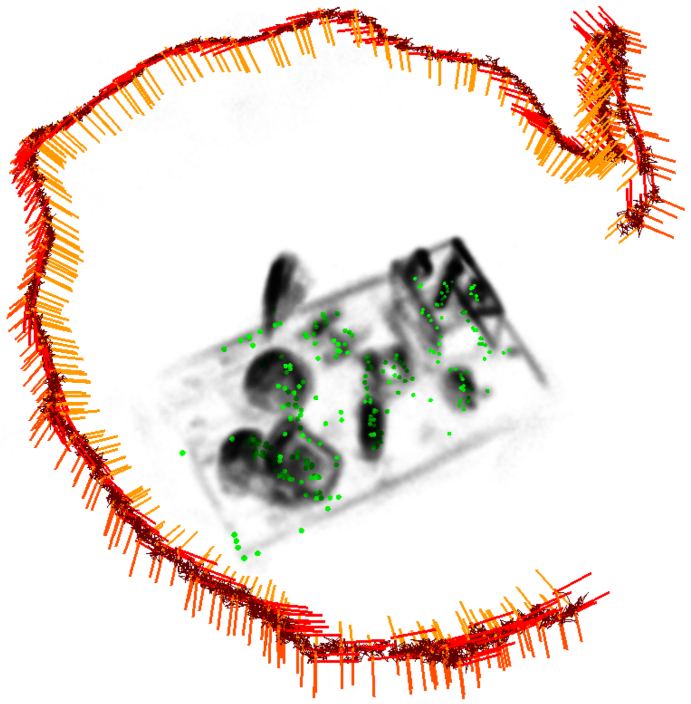


Figure 8.1: The camera trajectory (red) and the sparse 3D map (shades of grey) are created simultaneously in realtime using only a sparse stream of 3D point events (green).

8.1 Introduction

In this chapter the previously developed *Event-Based SLAM* algorithm is extended to a full 3D SLAM algorithm. In contrast to 2D SLAM, a 3D SLAM algorithm creates a map of the three-dimensional environment which is suitable for tracking all six degrees of freedoms of three-dimensional motions. 3D algorithms are much more powerful than their 2D counterparts, for example especially allowing flying robots, but also have a much higher theoretical and computational complexity.

Recent 3D SLAM algorithms [12, 24, 34, 42, 43, 56] almost exclusively use a 3D sensor like PrimeSense (see §3.1.3) to get depth measurements. Such sensors provide depth measurements in a range of 1-10 meters at an accuracy of 1-20 cm at a framerate of 30 or 60 frames per second. Additionally, a colour image is provided which may be an essential part in the SLAM algorithm or simply be used to augment the environment map.

KinectFusion [56] demonstrated how dense surface mapping and tracking can be accomplished with a Microsoft Kinect combined colour and depth sensor in realtime using GPU accelerated hardware. The algorithm proceeds in four steps. First a mesh, i.e. surface vertices and surface normals, of the currently observed surface is reconstructed from the measured depth values. This current measurement is compared towards the constructed scene model, and an iterated closest point (ICP) algorithm is used to align the mesh towards the model in order to derive the current camera pose in a global coordinate system. This pose estimate is used to integrate the current surface measurement into the scene model which is represented by a truncated volumetric signed distance function. This procedure is an example of an iterative approach to SLAM where the current map is used to compute a pose estimate and the new pose estimate is used to update the map. An evaluation of KinectFusion showed that it can produce detailed and drift-free environment models and that it can run in realtime if supported by GPU hardware.

Endres et al. [24] present a different method which matches frames based on image feature points from SIFT or SURF, and uses a RANSAC algorithm to compute possible transformations which are further optimized globally using a pose graph optimization like g^2o. Kerl et al. [43] try to optimize the frame-to-frame change in pose based on a warped representation of colour and depth image space by analytically deriving an optimizing procedure over the lie algebra $\mathfrak{se}(3)$. Bylow et al. [12] use a similar optimization over $\mathfrak{se}(3)$ to improve frame-to-frame matching using a similar signed distance function like in KinectFusion.

State-of-the-art algorithms have been quite successful in generating detailed dense environment maps, but still required a high amount of computation power – in many cases in form of dedicated GPU hardware. In this chapter the benefits of dynamic vision sensors will be exploited to develop an event-based 3D SLAM algorithm which builds a highly efficient sparse environment map fig. 8.2 and has the same robustness and quality than dense 3D algorithms. While event-based

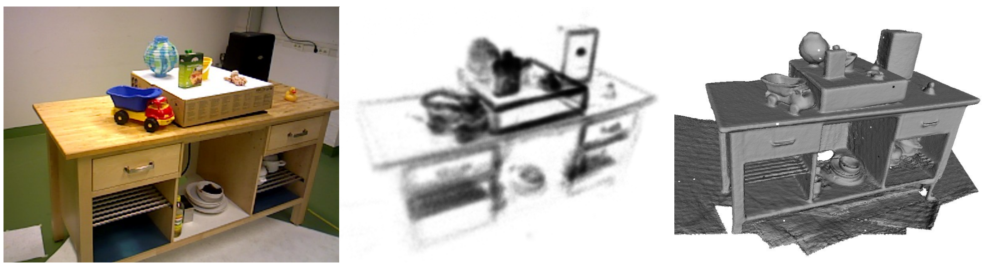


Figure 8.2: Comparison of sparse event-based map (middle) against a mesh created by the dense mapping algorithm KinectFusion (right). The color image (left) is provided for reference.

vision is a novel approach to design efficient algorithms for embedded systems, it has the same fundamental problems as classic computer vision with respect to estimating depth information. In classic computer vision this problem was solved for now by ignoring it and using active depth sensors. For the *Event-Based 3D SLAM* (*EB-SLAM-3D*) algorithm the same approach will be used, and the *eDVS* is combined with a PrimeSense active depth sensor to provide depth information for events.

Towards this goal an *eDVS* sensor is mounted on top of a standard PrimeSense sensor to form a depth-augmented dynamic vision sensor, the *D-eDVS*. Both sensors are calibrated such that for each individual event of the *eDVS* the corresponding depth can be computed using the depth stream from PrimeSense. Thus the *EB-SLAM-3D* algorithm directly works on a sparse stream of 3D point events – compared to the *EB-SLAM-2D* algorithm which works on a stream of 2D pixel events. This especially solves the problem of map back-projection stated in §7.2.2.

Due to the complexity of 3D environments and the memory consumption of dense grid maps, the *EB-SLAM-3D* algorithm will use an optimized data structure for the map and other simplifications over *EB-SLAM-2D* to assure realtime performance. In the end the algorithm presents itself in a strikingly simple formulation which can easily be implemented with only a few lines of source code.

This chapter will be continued with a presentation of the construction and calibration of the *D-eDVS* sensor in §8.2. After a presentation and discussion of the *EB-SLAM-3D* algorithm in §8.3, the chapter is concluded with an thorough comparison of the proposed *EB-SLAM-3D* algorithm towards ground truth of an external tracking system and the state-of-the-art algorithm KinectFusion in §8.4.

8.2 The *D-eDVS* sensor

To build the *D-eDVS*, an *eDVS* and a Asus Xtion camera using the PrimeSense sensor are combined physically by designing and constructing a small camera mount with the aid of a laser cutter (see fig. 8.3). The whole construction is

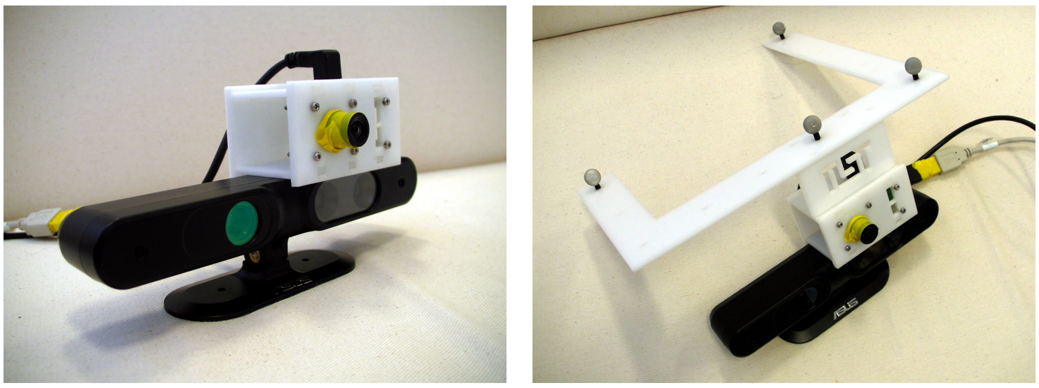


Figure 8.3: The *D-eDVS*: The *eDVS* is mounted on to of an Asus Xtion using a laser cutted casing (**left**) and for ground truth recordings an additional mount for markers can be added on top (**right**).

fastened onto the aluminium casing of the Asus Xtion and is quite robust to external forces. For evaluation purposes an additional modular mount for markers used by the external camera tracker is built. The lens for the *eDVS* sensor was chosen to match the vertical field of view of the depth sensor. This implies that the horizontal field of view of PrimeSense is not fully covered by the *eDVS* as the aspect ratio of the *eDVS* and the depth sensor are different.

In general a pinhole camera model is defined by the following equation eq. 8.1 which describes the projection of a 3D point $p \in \mathbb{R}^3$ onto the image sensor.

$$\mu\left(p \,|\, f, c, \kappa_1, \kappa_2\right) := L(\|s\| \,|\, \kappa_1, \kappa_2)^{-1}\, s + c \text{ with } s = \frac{f}{p_z}\begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{8.1}$$

The camera model has several parameters: f the focal length parameter, c the optical centre of the projection and κ_1, κ_2 for a simple but sufficient lens distortion model:

$$L(r \,|\, \kappa_1, \kappa_2) := 1 + \kappa_1\, r + \kappa_2\, r^2 \tag{8.2}$$

When depth values for pixel coordinates are available, the projection can be undone by inverting the projection:

$$\mu^{-1}\left(u, D \,|\, f, c, \kappa_1, \kappa_2\right) := \frac{D}{f}\begin{pmatrix} s_x \\ s_y \\ f \end{pmatrix} \text{ with } s = L(\|u - c\| \,|\, \kappa_1, \kappa_2)\,(u - c) \tag{8.3}$$

When a PrimeSense and an *eDVS* sensor are combined one can take a pixel coordinate u on the depth sensor with corresponding depth value D and compute the corresponding pixel coordinate v on the *eDVS* sensor by combining equations eq. 8.1 and eq. 8.3:

$$v = f(u, D \,|\, R, t, P_{\mathrm{PS}}, P_{\mathrm{eDVS}}) := \mu\left(R\,\mu^{-1}(u, D \,|\, P_{\mathrm{PS}}) + t \,|\, P_{\mathrm{eDVS}}\right) \tag{8.4}$$

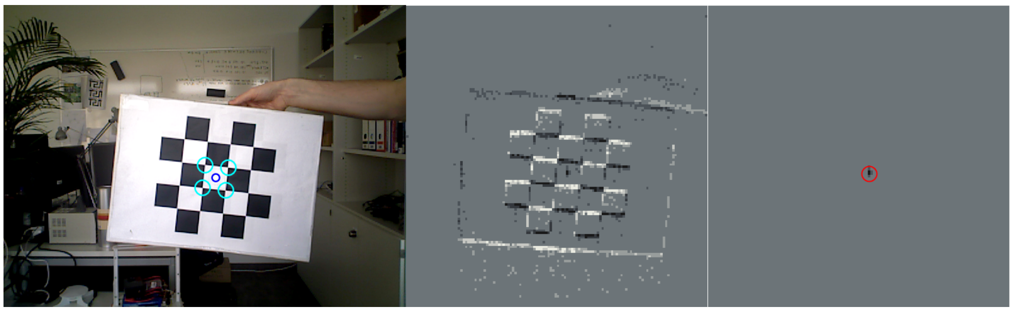


Figure 8.4: **Left:** The corners of a calibration plate are tracked with OpenCV (light blue) and the position of the diode blue is computed. **Middle:** The calibration plate viewed by the *eDVS*. **Right:** Events after frequency filtering and the computed position of the diode (red).

Here P_{PS} resp. P_{eDVS} are the set of camera parameters of the PrimeSense resp. *eDVS* sensors and $R \in \mathbb{R}^{3\times 3}$ and $t \in \mathbb{R}^3$ are the spatial transformation between the two sensors.

The camera parameters and the transformation can be found by numerical optimization of the non-linear least-square problem

$$\min_{R,t,P_{\text{PS}},P_{\text{eDVS}}} \sum_{i=1}^{n} \left\| v_i - f(u_i, D_i \,|\, R, t, P_{\text{PS}}, P_{\text{eDVS}}) \right\|^2 \tag{8.5}$$

for a set of measured data points $(u_i, D_i, v_i)_{1\le i\le n}$ over the parameters $R, t, P_{\text{PS}}, P_{\text{eDVS}}$. It has to be noted that the two focal length parameters are coupled together and it only makes sense to optimize them with respect to each other. Thus the focal length parameter for the PrimeSense sensor was fixed to the reference value of 520.

As the two cameras are very close together and approximately point into the same direction, the identity transformation is a good initial guess for the transformation parameters R and t. Initial values for the focal length projection parameters can be estimated from the opening angle and initial values for the barrel distortion are chosen as 0. The projection centre parameters can be initialized well by choosing the optical centre. This results in the initial parameters

$$R_0 = I,\ t_0 = 0,\ P_{\text{PS},0} = (520, (320, 240), 0, 0),\ P_{\text{eDVS},0} = (160, (64, 64), 0, 0) \tag{8.6}$$

Data points for the least square problem are captured using a calibration plate and a diode emitting a pulsed stream of light. Classic camera calibration methods use a calibration plate, however it is not straight-forward to track a calibration plate in a dynamic vision sensor and up to now no standard algorithms or software libraries are available. Dynamic vision sensors can be handled much easier by using a temporally changing signal like a pulsed diode. Thus a combined approach is used where a single calibration point is tracked in the PrimeSense image using a standard calibration plate. In the calibration plate a pulsed diode was inserted

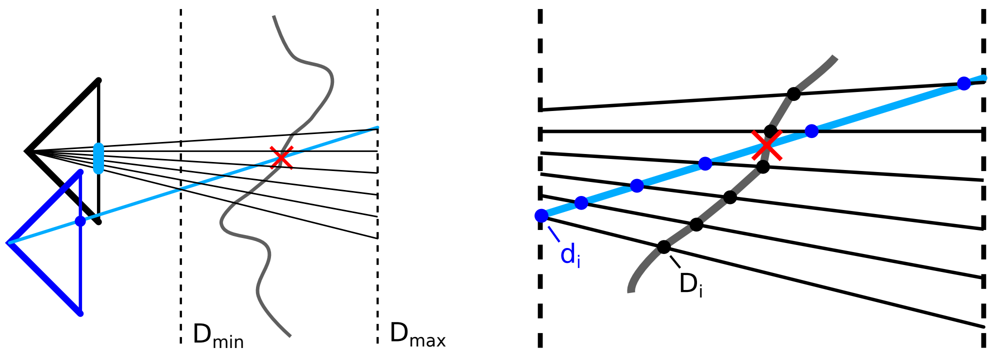


Figure 8.5: **Left:** An event pixel ray (cyan) from the *eDVS* sensor (blue) crosses several pixels from the PrimeSense depth sensor (black). The correct event depth value (red) is computed by intersecting the event ray with the measured depth surface (gray). **Right:** Close up.

which can be easily detected and tracked with a dynamic vision sensor (fig. 8.4). Here we exploit the fact that the pulse frequency is known and filter out all events which do not correspond to the fixed pulse frequency. This method reliably filters out all events which are due to the movement of the calibration plate and only preserves events from the diode (fig. 8.4).

With sufficient training points and due to the small deviation from the initial parameters a local optimization is sufficient for optimizing the camera parameters up to a root mean square error of 0.9 pixels in *eDVS* coordinates – the minimal possible error being 0.25 pixels due to rounding errors.

The optimized camera parameters can be used to compute an *eDVS* pixel coordinate for a given PrimeSense pixel coordinate with corresponding depth value. However to annotate *eDVS* pixel events with depth values the opposite direction is required. An *eDVS* pixel coordinate ray manifest as a short line segment when back-projected onto the PrimeSense sensor. The desired depth value lies on this line, but the correct value has to be computed. Fig. 8.5 shows a top-down schematic view of this problem. An *eDVS* event ray (blue) hits several PrimeSense pixel rays (black) and the correct intersection with the measured depth surface (grey) needs to be found. This problem is solved by pre-generating a look-up table where for each possible event ray pre-computed pixel coordinates together with depth values d_i (blue dots in figure) are stored. For each event during execution, these are compared to actual depth values D_i (black dots in figure) to find intersections, i.e. an index which satisfies $d_i \leq D_i \leq D_{i+1} \leq d_{i+1}$. In case of multiple intersection, the nearest has to be taken. As the line segment intersection takes place between singular pixels, the desired depth is computed by simply averaging the selected neighbouring depth values: $D_{\text{final}} = \frac{1}{2}(D_i + D_{i+1})$.

eDVS pixel coordinates and depth values can be used to compute corresponding 3D points in camera coordinates using again eq. 8.3. The result of calibration and event depth computation is an event-based dynamic vision sensor which

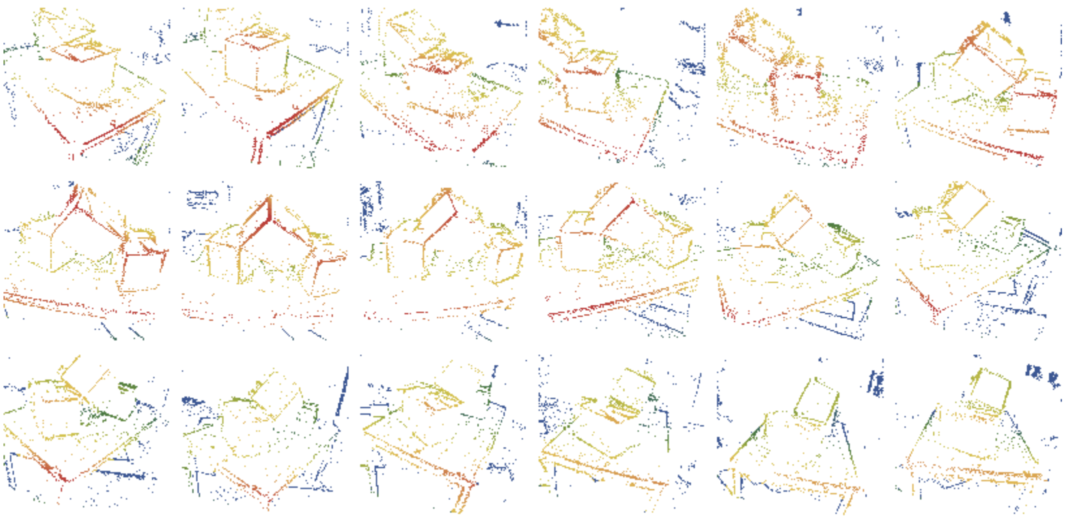


Figure 8.6: Stream of *eDVS* pixel events annotated with depth values (colour red to blue encodes distance near to far). Each images shows 1500 events and between images 30000 events are skipped.

generates a stream of 3D point events. Fig. 8.6 shows an example stream of depth-annotated events where the depth is encoded by colour. It is clearly visible how depth values further increase the information content of the event-stream.

8.3 The Event-based 3D SLAM algorithm

The event-based SLAM algorithm described in §7 builds a map based on the relative occurrence of events in specific map locations and uses a particle filter to continuously update an estimate of the camera pose. While the algorithm was only applied in the 2D case so far (see §7.3 and §7.4), it is formulated in a general way which in theory allows a direct application to the 2.5D or 3D case. The main challenge lies in the fact that the mapping between map locations and image coordinates is no longer a one-to-one mapping, but a ray through an image pixel hits a multitude of possible map locations. While the occurrence based approach could theoretically still be realized by using the whole pixel event ray and updating all corresponding pixels in the map which are hit by the ray, this approach is not investigated further here due to its slow runtime and possible instability. Instead the *D-eDVS* sensor which directly generates a stream of 3D point events is used to solve the event unprojection problem.

Following the notation from §7.2.2, we have $\Omega = SE(3)$ for the state space and $\Gamma = \mathbb{R}^3$ for the map space. Remember that the state space denotes the set of possible camera poses which is now a 3D rotation and a 3D position – thus described by the special Euclidean group – and the map space is space over which the map is built. By using the *D-eDVS* sensor, each 2D pixel events has a corresponding 3D point – denoted with $\mathscr{R}_D := \mathbb{R}^3$. However pixel coordinates are not required in the following, thus $\mathscr{R}_D$ will be used as event space. The projection function μ can be simplified to

$$\mu : \Gamma \times \Omega \to \mathscr{R}_D,\ \mu(u\,|\,p) = p\,u := p_R\,u + p_t \tag{8.7}$$

where p is treated as an Euclidean transformation defined by rotation $p_R \in \mathrm{SO}(3)$ and translation $p_t \in \mathbb{R}^3$. Now the inverse of μ is well defined as

$$\mu^{-1} : \mathscr{R}_D \times \Omega \to \Gamma,\ \mu^{-1}(e\,|\,p) = p^{-1}\,e = p_R^{-1}\,(e - p_t) \tag{8.8}$$

The *EB-SLAM* algorithm is using three maps: The occurrence map $\mathscr{O}$, the normalization map $\mathscr{Z}$ and the final map $\mathscr{M}$. With the projection function *eq.* 8.8, the occurrence map from eq. 7.6 can directly be extended to 3D:

$$\mathscr{O}^{(k)}(u) = \mathscr{O}^{(k-1)}(u) + \sum_{i=1}^{n} s_i{}^{(k)}\,\mathscr{N}\left(u \,\middle|\, (p_i{}^{(k)})^{-1} e^{(k)}, \sigma\right),\ \mathscr{O}^{(0)} = 0\,. \tag{8.9}$$

For the 2D occurrence map pixel events where integrate in a 2D grid and now for the 3D occurrence map 3D point events are integrate in a 3D grid – also called voxel grid.

The standard deviation σ in eq. 8.9 can be derived from the distance $D(e)$ of an event $e \in \mathscr{R}_D$ to the camera and the pixel focal length of the event-based sensor as

$$\sigma(e) := \frac{D(e)}{2\lambda f} \tag{8.10}$$

λ scales the normal distribution such that a given percentage of samples would be projected within one pixel on the *eDVS* sensor. A value of $\lambda = 2$ would indicate "2σ" thus 95%. Using a normal distribution is not an exact sensor model but a reasonable good approximation and values of $\lambda = 1.5$ have proven to be a good choice.

Fig. 8.7 shows an example run of the algorithm and demonstrates how path and map are created over time for an example scenario. The map itself is a sparse representation of the environment which unlike a mesh provided by dense methods is not necessarily directly accessible to the human eye. On closer inspection however it is visible how especially geometry borders and texture edges are represented in the event-based map. As another example, fig. 8.8 shows a bigger map for an office environment in comparison with colour photos. It is important to note that the map is not generated for direct visual processing by humans, but serves only the purpose of self-localization in the environment with the *Event-Based 3D SLAM* algorithm.

The final algorithm is listed in pseudo-code in alg. 9. *Event-Based 3D SLAM* can be implemented with only a few lines of code which is very short for a 3D SLAM algorithm, additionally no massively parallel operations are necessary. One can see, that the normalization from *EB-SLAM-2D* is not performed and the main

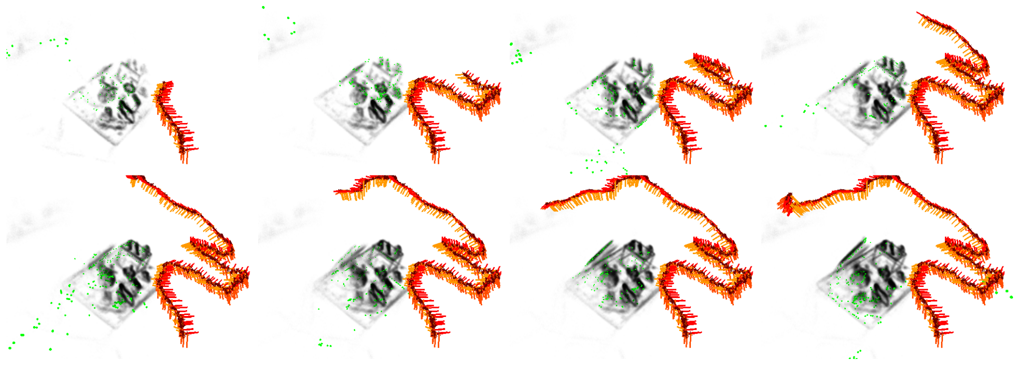


Figure 8.7: Computation of sparse 3D map (shades of grey) and path (red) over time for an example scenario. Current events are displayed as green dots.

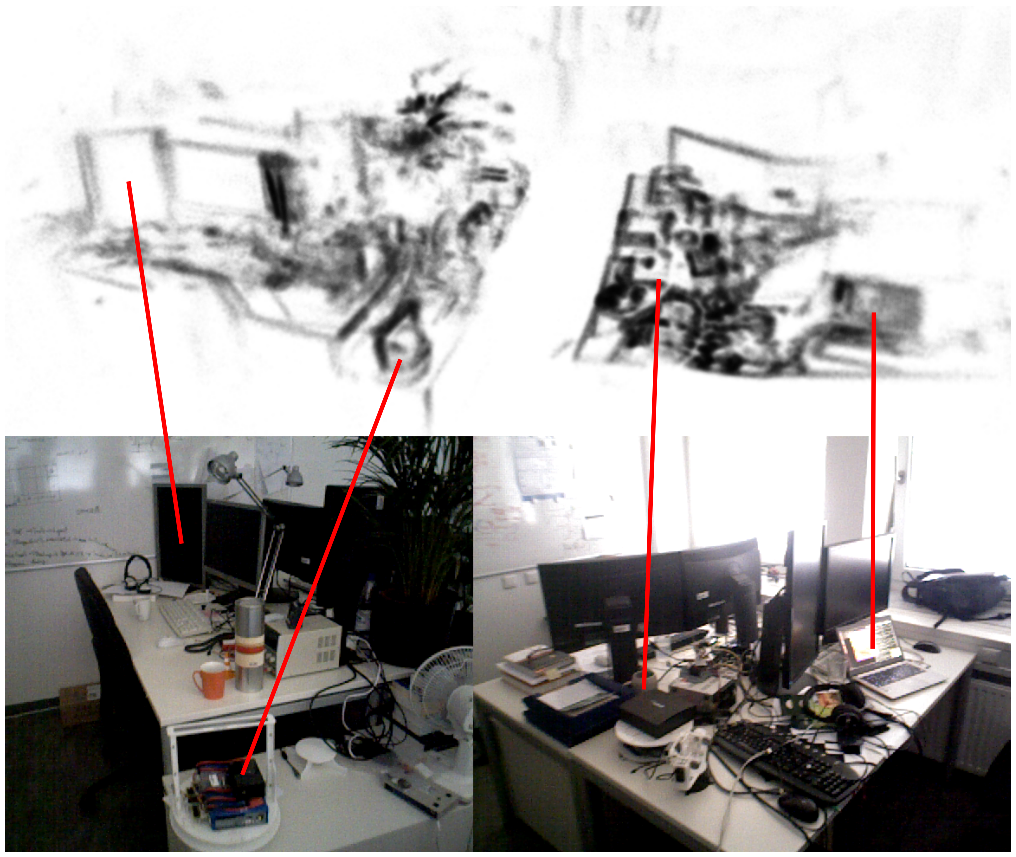


Figure 8.8: Sparse 3D map created by *Event-Based 3D SLAM* and photos of the same scenario from a different viewpoint. Corresponding key features are connected with lines for orientation.

reason is, that the normalization map is computational very intensive, can lead to instabilities and is actually quite smooth so it does not have the same impact as the occurrence map. More details are explained in the following considerations. The motion model is again realized as a diffusion process using a covariance matrix which models the possible movement for one event. Funnily enough, the particle resampling is actually the most complex operation. In the following several topics related to the *Event-Based 3D SLAM* algorithm are investigated in more detail.

Algorithm 9 *Event-Based 3D SLAM* (*EB-SLAM-3D*)

```
$\forall\, u \in \Gamma : \mathcal{O}(u) = 0$
$\forall\, 1 \le i \le N : p_i = 0, s_i = 1, p_* = 0$
for each new point event $e$ do
    for $i = 1 \to N$ do
        Sample $p_i$ from $\mathcal{N}_{\mathrm{SE}(3)}(p_i, \Sigma)$
        $s_i = (1-\alpha)\, s_i + \alpha\, \mathcal{O}(p_i^{-1}\, e)$
    end for
    for $i = 1 \to N$ do
        $\mathcal{O} = \mathcal{O} + \mathcal{N}\left(\cdot \mid p_i^{-1}\, e, \sigma\right)$
    end for
    if every K-th event then
        $p_* = \frac{\sum_{i=1}^{N} s_i\, p_i}{\sum_{i=1}^{N} s_i}$
        execute particle resampling
    end if
end for
```

Map normalization

The normalization map requires a continuous update of visible voxels to provide a relative measurement of how often a voxel has actually generated an event given the number of possibilities (see eq. 7.5). For a three-dimensional map this requires to update all voxels which are in the current view frustum. To avoid extensive visibility computations for individual voxels, the normalization map could be subsampled and the values for individual voxels could be computed using linear interpolation. However as already apparent in fig. 7.3, the normalization map is much smoother than the occurrence map. The occurrence map capture thin edges, singular points and other filigree features, and can thus change its value rapidly over a distance of only a few voxels. On the contrary, the normalization map captures if a voxel is visible, and as the frustum is normally quite wide in number of voxels the value of the normalization map as a function of space changes only slowly. This results in the important observation that for a given event and several similar particle poses the occurrence map has a much higher influence on the relative difference between particle scores than the normalization map. This is in general not true for a given pose and different events, but in a particle filter particles are only evaluated and selected relative to each other with respect to one measurement after another. If for the current event the local values in the occurrence map are generally low compared to other regions which have been observed longer, scores will be low for all particles and all particles will be subject to the same selection pressure. On the opposite if the values in the occurrence map are generally high because a lot of information has already been gathered, all particles will have this benefit and again all particles have the same chance to increase their score if they align well with the map.

Parametrization

The algorithm can be parametrized in several ways which will be explained in the following. In §8.4 an evaluation of theses parameters is presented and their impact on the tracking quality and the performance of the algorithm are qualitatively evaluated.

Particle count N: The number of particles used in the particle filter has a strong impact on the quality of the result path but also a direct negative impact on the runtime.

Events until resampling K: Resampling after every event has a huge impact on performance and yields much worse results. This parameter delays the resampling step after enough information has been gathered.

Exponential decay factor α: This constant can be computed conveniently with eq. 6.27. The constant G_0 is chosen to be K in the following – this has a reasonable interpretation and gives good results.

Batch size B: Batching as explained in §6.4.2 can be used also for *Event-Based 3D SLAM*. This parameter has a huge impact on performance for small values of B. A good choice is $B = 3$.

Event share: An additional trick to reduce runtime is to not use all events, but only a random subset. This directly increased performance but if too many events are discarded tracking quality suffers.

Diffusion: As *Event-Based 3D SLAM* uses random diffusion as a motion model and the movement per event is much more difficult to compute, the standard deviation for position and rotation diffusion should be chosen wisely. Evaluation will show that a range of values works well for a broad selection of scenarios.

Voxel map

As the map space is three-dimensional, a three-dimensional grid map of voxels has to be maintained for the occurrence map. The data structure for the voxel grid has to be chosen with care to avoid exessive memory consumption and slow access times. On the one hand a fast access mechanism is crucial to the runtime of the *EB-SLAM* algorithm as the map has to be updated for all events and several particles. On the other hand dense voxel grids require a huge amount of memory. A voxel grid which stores a 4 byte floating point value at each voxel and with has a side length of 512 voxels already requires 512 MB system memory – compare to a 2D grid with 512^2 pixels which only requires 1 MB.

A typical data structure for efficiently storing voxels is an octree. An octree covers a rectangular region of space by recursively splitting it into eight equally sized and axis aligned boxes down to the level of individual voxels. Only nodes which actually contain a non-zero voxel are created, thus saving memory for all voxels which are not used. Octrees have the disadvantage, that the covered space has to be known in advance and that access time is logarithmic in the side length of the covering box. For a box with side length of 10 meters and a resolution of 1 cm, this constant is already 10. Additionally as space is partitioned down to individual voxels and new nodes are created on-the-fly, memory access is not streamlined. Here a two-level approach similar to [3] is used, where the octree is only divided up to a specific size and dense ”chunks”, i.e. voxel grids of size 32^3, are used as leaf nodes instead of individual voxels. Fig. 8.9 shows an example of such a chunk voxel grid. In this example there are only 316 out of 1664 chunks filled with voxels saving 80% of memory. This ratio usually gets much higher the longer the SLAM process is running.

Diffusion

The diffusion model $\mathcal{N}_{\mathrm{SE}(3)}(p,\Sigma)$ for a 3D pose $p \in \mathrm{SE}(3)$ can actually be a bit tricky. It requires to diffuse a three-dimensional position $t \in \mathbb{R}^3$ and a three-dimensional rotation $R \in \mathrm{SO}(3)$. For the position this is straight forward and realized as

$$\big(x,y,z\big) \leftarrow \big(\mathcal{N}(x,\sigma_x), \mathcal{N}(y,\sigma_y), \mathcal{N}(z,\sigma_z)\big) \tag{8.11}$$

where $\mathcal{N}$ is a Gaussian normal distribution.

However, for a rotation the diffusion process is by no means straight forward [57]. A two-dimensional rotation has only one degree of freedom, the angle of rotation, and one could think of taking a normally distributed angle to create a normally distributed rotation matrix:

$$R \leftarrow R_2(\mathcal{N}(0,\sigma_\theta))\, R \tag{8.12}$$

where $R_2(\theta)$ is the 2D rotation matrix of angle θ. As SO(2) can be parametrized by $\mathbb{R}/[0,2\pi[$, it is identical to $\theta \leftarrow \mathcal{N}(\theta,\sigma_\theta)$. This approach has the issue that the

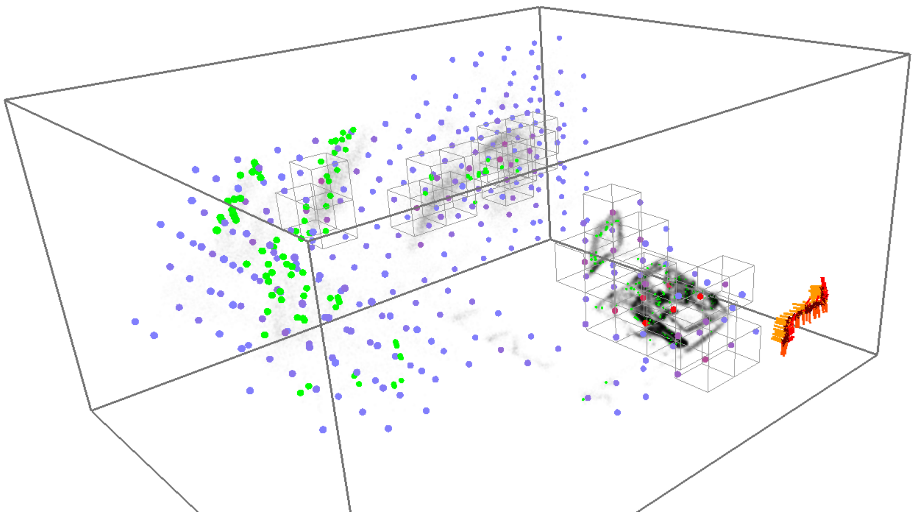


Figure 8.9: Visualization of voxel grid ”chunks”. Spheres coloured blue to red indicate centre position of chunks which contain at least one voxel. Chunks with a significant amount of voxels are rendered as grey boxes. The large box indicates the area covered by the whole voxel grid. Additionally the estimated path (red/orange), current events (green) and the current map (shades of grey) are displayed.

two tails of the normal distribution wrap around at an angle of π which may be desired or not. But for *EB-SLAM* this is not much of an issue as σ_θ is usually very small.

For three-dimensional rotations there are two additional degrees of freedom for the direction of rotation. A possible representation is axis/angle where a rotation is given by an unit vector for the axis of rotation and an angle. A normally distributed rotation can be sampled by sampling a random, uniformly distributed point on the 2-sphere as the axis and a half-normally distributed angle of rotation. The axis/angle representation can then be transformed to a rotation matrix. A random point on a 2-sphere can for example be samples with

$$a \leftarrow \frac{(x, y, z)}{\sqrt{x^2 + y^2 + z^2}} \tag{8.13}$$

where x, y, z are sampled from a standard normal distribution $\mathcal{N}(0,1)$. Given a uniformly distributed axis $a \in \mathbb{R}^3$ and a normally distributed angle $\theta \leftarrow \mathcal{N}(\theta, \sigma_\theta)$, a normally distributed quaternion can be computed as

$$\left(\cos\tfrac{\theta}{2}, \sin\tfrac{\theta}{2}\, a_x, \sin\tfrac{\theta}{2}\, a_y, \sin\tfrac{\theta}{2}\, a_z\right) \tag{8.14}$$

and for small angles this can be approximated with:

$$\left(1 - \tfrac{\theta^2}{4}, \tfrac{\theta}{2}\, a_x, \tfrac{\theta}{2}\, a_y, \tfrac{\theta}{2}\, a_z\right) \tag{8.15}$$

8.4 Evaluation

In order to compare paths from different trackers, theses paths first needs to be aligned towards each other. This step is necessary as most tracking algorithms choose an arbitrary global coordinate system for map and path and thus a transformation between the two systems need to be computed. A path is a sequence of tuples $u = (t, p, q) \in \mathbb{R} \times \mathbb{R}^3 \times \mathrm{SO}(3) =: U$, where $p \in \mathbb{R}^3$ indicates the current position, $q \in \mathrm{SO}(3)$ the current rotation represented by a quaternion, and $t \in \mathbb{R}_+$ the corresponding timestamp of the pose. Timestamps greatly simplify the alignment process as a point on one path only needs to be matched against the temporal nearest point on the other path. Thus the root-mean-square error function (RMSE) of two paths (u_i) and (u'_j) is defined as:

$$E\big((u_i), (u'_i) \,|\, a\big) := \sqrt{\frac{1}{n} \sum_{i=0}^{n} \big(e(A(u_i, a), u'_{\mu(j)}\big)^2} \tag{8.16}$$

where the matching function $\mu : \mathbb{N} \to \mathbb{N}$ is defined as $\mu(i) := \mathrm{argmin}_j\, |t_i - t'_j|$. Other matching mechanisms are possible, in particular it may be advantageous to reject points at the ends of the paths when the minimal temporal difference is too high. Here the concrete error function e measures the Euclidean distance between the tracked path point and the ground truth path point:

$$e(u, u') := \|p - p'\| \tag{8.17}$$

The alignment function $A : U \times U \to U$ applies the transformation of the parameter set a on a point on the path and is defined as

$$A\big((t, p, q), (t_a, p_a, q_a)\big) := (t + t_a, q_a\, p + p_a, q_a\, q) \tag{8.18}$$

The alignment includes a temporal offset t_a as the two data streams can be started at slightly different times. The optimal alignment

$$a^* = \underset{a \in U}{\mathrm{argmin}}\, E\big((u_i), (u'_i) \,|\, a\big) \tag{8.19}$$

can be found by a mathematical optimization. Possible methods are gradient descent or Particle Swarm Optimization (PSO) [41], or if the temporal offset is neglected a principal component analysis can be used. For the evaluation in this section accelerate particle swarm optimization [86] was used which could find very good alignments in a matter of seconds.

Additionally it is possible to consider the rotational alignment error

$$e_{\mathrm{rot}}(u, u') := |2 \cos^{-1}\big((q^{-1}\, q')_w\big)| \tag{8.20}$$

This definition of e_{q} is due to the fact that the w component of a unit quaternion directly encodes the amount of rotation. The rotational error additionally requires

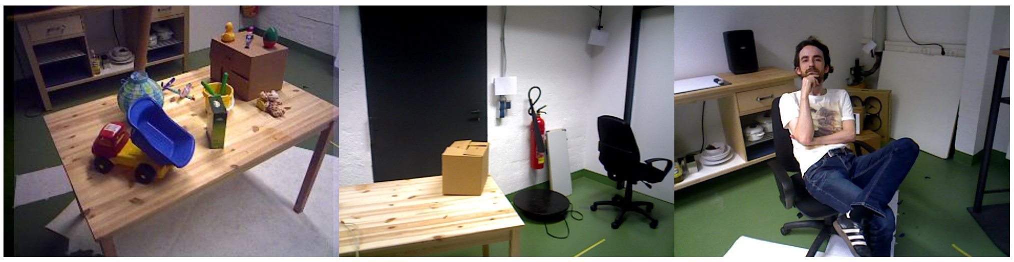


Figure 8.10: Colour images for some of the scenarios used in this evaluation. From left to right: Scenarios ”Table 1”, ”Room” and ”People”.

the alignment of the orientation as the coordinate frame chosen by for example an overhead tracker is not necessarily identical to the implicit camera coordinate frame where the x- and y-axis define the image plane and the z-axis points into the world.

For evaluation a dataset of 26 ”takes”, i.e. paths, in five different scenarios and the length of individual takes was 20 to 40 seconds. Fig. 8.10 shows colour images for an impression of several scenarios. To compare the tracking results against ground truth the pose of the camera was tracked with the marker-based overhead tracking system OptiTrack V100:R2. Due to the limitations of the PrimeSense depth sensor and the fact that the overhead tracker is immovable, room scenarios are limited to one room indoor settings. For this dataset, the depth information was computed from the high-resolution depth stream with a spatial resolution of 640x480 and a framerate of 30 Hz, with the exception of the scenario ”Table 1” which was recorded with a resolution of 320x240 and a framerate of 60 Hz. While the spatial resolution does not have an impact on the results of *Event-Based 3D SLAM* as it is downsampled to the *eDVS* sensor resolution of 128x128 pixels, the framerate has a large impact. With a framerate of 60 Hz the PrimeSense sensor provides reliable depth information only in a range up to approximately 1.5 m as it uses a low-resolution infrared image to compute depth. With the lower framerate of 30 Hz depth measurement are accurate up to 2.5 m.

The *EB-SLAM-3D* algorithm was executed with two sets of parameters: The ”default” set uses a reasonable choice of parameters which achieves very good tracking results and the ”fast” parameter set uses parameters which focus on very fast execution speed with only small impact on the RMSE. Table 8.1 shows an overview of the RMSE and the processing speed for the different scenarios. The RMSE is excellent when considering the complexity of the 3D matching problem and the low resolution of the *eDVS* sensor. The processing speed is measured as the ”relative realtime factor” which is the quotient between recording time and computation time. This is necessary as the rate at which events are generated depends on movement speed and visual complexity of the scene. However all takes are recorded with more or less constant movement and normal velocities with no particular breaks to improve tracking performance.

Fig. 8.11 shows two examples for paths tracked with *EB-SLAM-3D* compared against ground truth from an overhead tracking system. Six more examples are shown in the appendix in fig. B.14. The plots demonstrates the very good results which can be achieved with *EB-SLAM-3D*.

To give an insight into the influence of parameters of the *Event-Based 3D SLAM* algorithm, RMSE and runtime performance were measured for a variety of different parameter choices. Fig. 8.12 shows the RMSE and relative speed of for varying number of particles N and varying number of events until resampling K averaged over the whole dataset. Results for additional parameters are shown in the appendix in fig. B.15 and fig. B.16. For this evaluation, values for the non-modified parameters were chosen equal to the ”default” parameter set: $N = 100$, $K = 100$, $B = 3$, 100% used events, voxel size 0.01 cm and standard deviation for position and rotation diffusion equal to 0.008.

Table 8.1: Memory consumption, positional root-mean-square error (RMSE) and runtime performance for *EB-SLAM-3D* for the different scenarios.

Scenario	Takes	*EB-SLAM*	*EB-SLAM* (default)		*EB-SLAM* (fast)	
		RAM	RMSE	Speed	RMSE	Speed
1: ”Table 1”	2	25 MB	3.1 cm	2.0 x	4.0 cm	20 x
2: ”Sideboard”	4	14 MB	4.0 cm	2.2 x	5.2 cm	23 x
3: ”Table 2”	8	27 MB	4.9 cm	1.4 x	9.1 cm	16 x
4: ”Room”	8	21 MB	13.4 cm	2.5 x	13.3 cm	27 x
5: ”People”	4	15 MB	6.1 cm	2.2 x	7.0 cm	24 x

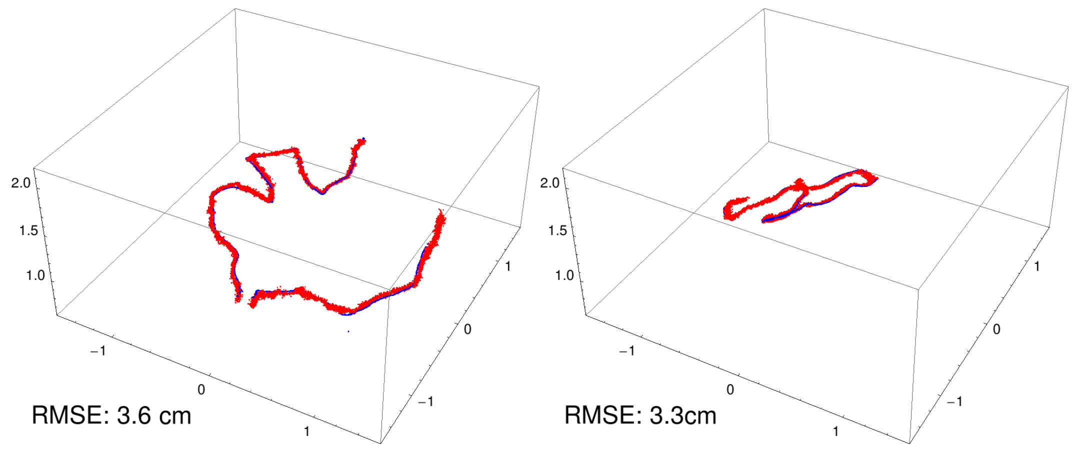


Figure 8.11: Two examples trajectories generated by *EB-SLAM-3D* (red) compared to ground truth (blue) from an overhead tracking system. The left path is from scenario 1 and the right path from scenario 2.

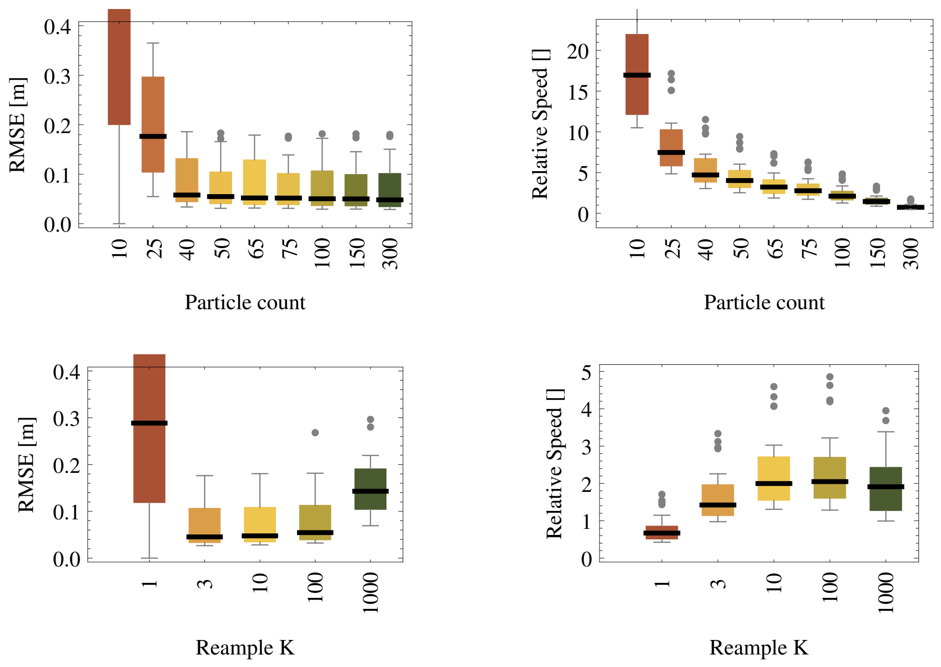


Figure 8.12: RMSE in position and execution time relative to take duration plotted for the following parameters: Number of particles N (top) and number of events until resampling K (bottom).

Part III

Conclusion and Appendix

9 Conclusions

The technological advance has increased our possibilities to collect huge amounts of data – but more data does not necessarily answer more questions or give deeper insight into problems. Up to now the skill to extract valuable and non-trivial information has only been truely mastered by biological systems. In the field of computer vision, images with a higher resolution or videos with a higher framerate do not necessarily simplify the analysis. On the opposite, the blind increase of the number of frames taken per second or the number of pixels in an image requires more computation power to deal with even more redundant information.

The two main topics presented in this thesis, depth-adpative superpixels and event-based SLAM, are examples of computer vision methods which work on a sparse representation of vision data which intelligently reduces the amount of data processed by the computer. Superpixels form an intermediate layer on top of the full image which condenses the redundant information of many similar pixels into only few clusters. Event-based SLAM uses an dynamic vision sensor which provides an intelligent, continuous representation of the dynamic changes in a scene. Both algorithm do not blindly reduce the resolution of the input data or try to compress data with complex mathematical models to take up less memory or disk space, but focus on the relevant aspects which are required to solve a specific task.

Both methods use different principals to decide which information is relevant. While superpixels mainly focus on the spatial redundancy in an image, with an excursion to possible application to temporal redundancy in form of the *t-DASP* algorithm, the work on dynamic vision sensors focuses on the temporal redundancy in a dynamic scene. These two topics have been treated separately, but of course the techniques could be combined to make the most use of both. An interesting future work could be the extension of the *EB-SLAM-3D* algorithm with superpixels to further enhance tracking qualities and to create a map which can be used for collision reasoning. Such a method could directly use the *D-eDVS* sensor as it already provides both event-based changes and frame-based RGB-D images.

Superpixels on their own can be used with normal colour cameras and the algorithm presented here for combined colour and depth sensors can be used in a wide variety of applications, for example robotics, computer games, or human-machine interfaces in general, if the PrimeSense sensor is available. The *DASP*, *s-DASP* and *t-DASP* algorithm can be downloaded and used by anyone as all software has been released to the public as open source. At the moment the necessary depth information must be provided by an active sensor which has a limited use case scenario and can for example not be used outside or in multitude as the active sensing interferes with the sun and with each other. However these algorithms give an insight in what can be possible in the near future when depth information will be easier to acquire.

One could critique that event-based vision requires an additional non-standard sensor which makes applications less feasible than algorithms which require only classic frame based vision. However compared to active depth sensors, dynamic vision sensors are passive sensors which work basically everywhere when light is available. Actually they can be used in even more scenarios as they are adapted better to strong differences in contrast, changing lighting conditions and fast movements. The main reason for the widespread use of frame-based cameras could be attributed to the fact that they are a natural development from the still photo, and that they are widely used today, thus cheap and a well-known technology. While classic frame-based computer vision has gone a long way, there is actually no reason that frame-based vision is the best choice in the long term. Quite the contrary, the only systems which have successfully solved computer vision problems are humans and animals which most probably do not work in the sense that they process one image snapshot after the next one. Evidence of biologists and neurologist hints in the general direction of a much more dynamic and event-based style of processing in biological systems. An opening of computer vision to new kinds of biologically inspired sensors and biologically inspired processing of sensor data could prove to be a fruitful exchange of ideas from different points of view.

The *EB-SLAM-3D* algorithm is a good example to highlight how a combination of different principles can lead to a superior, novel approach. Simultaneous localization and mapping has been investigated for a long time in robotics as it is a fundamental requirement to navigate in unknown environments. A creative adaption of established SLAM principles to event-based data streams of dynamic vision sensors results in an extremely easy and at the same time high-performing algorithm. Evaluation in several scenarios and comparison to state-of-the-art algorithms demonstrate how the intelligent reduction of the processed data yields very good results in minimal time. *EB-SLAM-3D* requires weaker assumptions about the movement speed, as the event-based model is superior to the frame-based model when measuring relative change of information. At the same time the algorithm does not require special processing hardware like a GPU and can run many times faster than realtime on average computer systems making it an ideal candidate for small or flying robots or embedded devices.

A Partition quality metrics

A.1 Quality metrics for superpixels

Superpixel properties can be divided in two groups of metric: supervised and unsupervised. Supervised metrics compare a superpixel segmentation against manually created ground truth. This includes the boundary recall measure (see def. 13) and the undersegmentation error (see def. 14). Unsupervised metrics measures intrinsic properties of superpixels and do not require manual ground truth. In this section are the following metrics: isoperimetric quotient (see def. 18), superpixel connectivity quotient (see def. 16), explained variation (see def. 21) and compression error (see def. 20).

In the following it is assumed that $\mathfrak{P}$ is a partition of a finite undirected graph. ∂S indicates the **boundary** of a segment $S \in \mathfrak{P}$ in the sense of eq. 8 and eq. 9. $|\partial S|$ denotes the **length of the boundary**, i.e. the number of vertices on the boundary of the segment $S \in \mathfrak{P}$, and $|S|$ the area of the segment, i.e. the number of vertices belonging to the S. The graph structure can for example be the regular pixel lattice graph or the irregular neighbourhood graph of superpixels.

Boundary Recall

Boundary recall measures how well the boundary of a partition matches the segment boundary of a given reference partition. It indicates how much of the boundary is superfluous or missing with respect to the reference.

Definition 13. Let $\mathfrak{P}$, $\mathfrak{X}$ be partitions. The **boundary recall (BR)** of $\mathfrak{P}$ with respect to $\mathfrak{X}$ and a maximum reach $\delta \in \mathbb{R}_+$ is defined as

$$\mathbf{BR}_\delta(\mathfrak{P}|\mathfrak{X}) := \frac{1}{|\partial\mathfrak{P}|} \sum_{v\in\partial\mathfrak{P}} \mathbf{1}_{\mathbb{B}}(\min_{x\in\partial\mathfrak{X}} \|v - x\| < \delta) \tag{A.1}$$

The Boolean indicator function $\mathbf{1}_{\mathbb{B}}$ is defined as:

$$\mathbf{1}_{\mathbb{B}} : \{\text{True}, \text{False}\} \to \{0,1\},\ \mathbf{1}_{\mathbb{B}}(x) := \begin{cases} 1 & \text{if } x = \text{True}, \\ 0 & \text{if } x = \text{False}. \end{cases} \tag{A.2}$$

The parameter δ controls how close the boundaries of the two partition must be to one another. For example, a value of 1 pixel indicates that every pixel of a partition border must be adjacent to a border pixel of the reference partition. The boundary recall value lies between 0 (worst) and 1 (best) and normally increases with the number of segments.

Undersegmentation Error

The undersegmentation error [5] measures how well segments from a reference partition are matched by segments from a given partition. For each reference segment all overlapping segments from the given partition are found and the area of theses segments are compared to the size of the reference segment. The smaller the difference, the better a partition represents the reference partition.

Definition 14. Let $\mathfrak{P}, \mathfrak{X}$ be partitions and $\gamma \in [0,1]$ a parameter. The **undersegmentation error (USE)** is defined as

$$\mathbf{USE}_\gamma(\mathfrak{P}|\mathfrak{X}) := \frac{1}{|\mathfrak{P}|} \sum_{T \in \mathfrak{X}} \sum_{S \in \mathfrak{P}} \mathrm{Area}(S)\, 1_{\mathbb{B}} \left[\frac{|S \cap T|}{|S|} > \gamma \right] - 1 \tag{A.3}$$

The parameter γ accounts for small errors in the boundary computation, a typical value is $\gamma = 0.05$. The undersegmentation error lies between 0 (best) and 1 (worst) and normally decreases with the number of segments.

Connectivity

For a lot of applications it is advantageous that superpixels are connected in the normal graph theoretic sense.

Definition 15. A segment $S \in \mathfrak{P}$ is called **connected** if for each two vertices $u, v \in S$ there is a path from u to v which lies completely in S. A partition $\mathfrak{P}$ is called **connected** if every segment $S \in \mathfrak{P}$ is connected.

Connectivity is sometimes a very strong assumption, as it forbids small, noisy enclaves. On the other side, often the criterion is too weak as it does not forbid stretched and widely distributed segments with narrow transitions.

In the following we will use a measure which compares the area of all connected components except the largest against the area of the segment.

Definition 16. Let $S \in \mathfrak{P}$ a segment and $\{c_i\} \subset S$ its connected components. The **connectivity quotient (CQ)** of a segment $S \in \mathfrak{P}$ is defined as

$$\mathbf{CQ}(S) := 1 - \frac{\max_i |c_i|}{|S|} \tag{A.4}$$

Using the area weighted mean this definition can be extended to the whole partition:

Definition 17. Let $\mathfrak{P}$ be a partition of a graph $G = (V, E)$. The **connectivity quotient (CQ)** of a partition is defined as

$$\mathbf{CQ}(\mathfrak{P}) := \frac{1}{|\mathfrak{P}|} \sum_{S \in \mathfrak{P}} |S| \, \mathbf{CQ}(S) \tag{A.5}$$

In the best case, where all segments are fully connected, the connectivity quotient is 0.

Isoperimetric quotient

Superpixels can have many shapes, but a local compact shape is benefitial for many applications. An easy method to compute the "compactness" of superpixels is the isoperimetric quotient.

In a classic, geometric sense, the isoperimetric quotient compares the area of a shape to its perimeter. The isoperimetric quotient is motivated from Euclidean geometry where it expresses similarity with a circle.

Definition 18. The **isoperimetric quotient (IPQ)** of a segment S is defined as

$$\mathbf{IPQ}(S) := \frac{4\pi |S|}{|\partial S|^2} \tag{A.6}$$

The tessellation of the plane with spheres is of course not possible. There exist three regular, homogeneous - so called Platonic - tessellations: triangles, rectangles, and hexagons. The isoperimetric quotient of a triangle is $\frac{\pi\sqrt{3}}{9} \approx 0.6046$, for a sphere it's $\frac{\pi}{4} \approx 0.7854$ and for a hexagon $\frac{\pi\sqrt{3}}{6} \approx 0.9069$.

In the classical case the isoperimetric quotient lies between 0 and 1, where the circle itself reaches the maximal value of 1. For lattice graphs, area and perimeter of segments are an approximation to the classical geometric definitions. Here the isoperimetric quotient can be bigger than 1, as boundaries have a finite thickness in contrast to spaces which allow infinitesimal thin boundaries.

The isoperimetric quotient of a partition is defined as the weighted mean of the isoperimetric quotient of its segments [68]:

Definition 19. The **isoperimetric quotient** of a partition $\mathfrak{P}$ is defined as

$$\mathbf{IPQ}(\mathfrak{P}) := \frac{1}{|\mathfrak{P}|} \sum_{S \in \mathfrak{P}} |S| \, \mathbf{IPQ}(S) \tag{A.7}$$

Compression Error

A good superpixel partition creates segments which represent individual elements well. For example, the compression quality of a superpixel partition can be measured by comparing segment mean values against individual pixel values. Here the root mean square error is used to compare the original image against the partitioned image where each pixel is assigned the value of the corresponding superpixel.

Definition 20. Let $f : \Omega \to \mathscr{F}$ be a feature annotation into a feature space $\mathscr{F}$ equipped with a metric $\|\cdot\|_{\mathscr{F}}$. Let $\mathfrak{f}_S$ be the selected segment feature for each segment $S \in \mathfrak{P}$ in the partition. The **compression error (CE)** is defined as

$$\mathbf{CE}(\mathfrak{P}, f, (\mathfrak{f}_S)_{S\in\mathfrak{P}}) := \sqrt{\frac{1}{|\mathfrak{P}|}\sum_{S\in\mathfrak{P}}\sum_{x\in S}\|f(x)-\mathfrak{f}_S\|_{\mathscr{F}}^2} \tag{A.8}$$

The smaller the compression error, the better a superpixel segmentation is able to represent the original image data. It is limited by the feature variation in the image and stands in direct competition to superpixel compactness.

Explained Variation

Another metric for the compression quality of a superpixel partition is the explained variation metric.

Definition 21. Let $f : V \to \mathscr{F}$ be a feature annotation into a feature space $\mathscr{F}$ equipped with a metric $\|\cdot\|_{\mathscr{F}}$ and a mechanism to compute mean values. Let $\mathfrak{f}_S$ be selected segment features for each segment $S \in \mathfrak{P}$ in the partition and $\bar{\mathfrak{f}} := \mathrm{mean}_{x\in\mathfrak{P}} f(x)$ the mean feature over the whole partition. The **explained variation** of the selected features $(\mathfrak{f}_S)_{S\in\mathfrak{P}}$ is defined as

$$\mathbf{EV}(\mathfrak{P}, f, (\mathfrak{f}_S)_{S\in\mathfrak{P}}) := \frac{\sum_{S\in\mathfrak{P}}|S|\|\mathfrak{f}_S-\bar{\mathfrak{f}}\|_{\mathscr{F}}^2}{\sum_{x\in\mathfrak{P}}\|f(x)-\bar{\mathfrak{f}}\|_{\mathscr{F}}^2} \tag{A.9}$$

The better segment features represent all pixels in the segment, the smaller the explained variation measure. The metric may have little information content and yield values near to 1 if the overall mean value lies to far away from the majority of individual feature values. For example if pixel feature values would be 3D positions on a sphere, the overall mean would be the centre of the sphere. Then the local distance between pixel features and segment features may be much smaller than the distance between segment features and the overall mean feature.

Uniform distribution

Another desired property of superpixels is a uniform distribution over the given supporting structure. In the following we build a segment density function over a graph by placing Gaussian kernels at the centre of each segment. This density function is then compared against the uniform distribution over the vertices.

Definition 22. Let $Q \subset \Omega$ be a finite set of points and k kernel function. The **density** with respect to Q is defined as

$$\rho_Q(v) := \sum_{q\in Q} k(\|v-q\|)\,. \tag{A.10}$$

A typical two-dimensional density kernel is given by the multivariate normal distribution

$$k_\sigma(x) := \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} \tag{A.11}$$

The optimal uniform distribution would result in a constant density of $\rho_0 := \frac{|Q|}{|\Omega|}$ for all vertices. Assuming that G is a lattice graph, the optimal kernel parameter σ^* is chosen such that $k_{\sigma^*}(0) = \rho_0$. This gives $\sigma^* = \sqrt{\frac{|V|}{2\pi|\Omega|}}$ for the standard deviation of the Gaussian kernel.

The centres of segments of a partition define a density over the graph. The centre of a segment is simply the mean position of all points in the segment. The closer this density is to the constant density, the more uniform segments are distributed.

Definition 23. Let C be the set of centre points of the partition $\mathfrak{P}$. The **uniform distribution error (UDE)** of a partition is defined as

$$\mathbf{UDE}(P) := \sqrt{\sum_{v\in\mathfrak{P}} \left(\rho_C(v) - \frac{|C|}{|\mathfrak{P}|}\right)^2} \tag{A.12}$$

A.2 Quality metrics for superpoints

To measure the quality of 3D superpixels some additional superpixel properties are introduced. The 3D isoperimetric quotient, which is similar to the normal isoperimetric quotient defined in image space, measures superpixel area and perimeter in 3D. The 3D isoperimetric quotient for a segment is computed as in def. 18:

$$\mathbf{IPQ}_{3\mathrm{D}}(S) := \frac{4\pi\,\mathbf{Area}_{3\mathrm{D}}(S)}{\mathbf{Perimeter}_{3\mathrm{D}}(S)^2} \tag{A.13}$$

with the 3D pixel area computes as

$$\mathbf{Area}_{3\mathrm{D}}(S) := \frac{1}{f^2}\sum_{u\in S}\frac{D(u)^2}{n(u)_z} \tag{A.14}$$

and the 3D pixel boundary computed as

$$\mathbf{Perimeter}_{3\mathrm{D}}(S) := \frac{1}{f}\sum_{u\in\partial S}\frac{D(u)}{n(u)_z}. \tag{A.15}$$

with f the camera focal length, $D(u)$ point depth and $n(u)$ point normal. Using the z component of the point normal one can approximate surface projection distortion, as seen in eq. 3.8 and eq. 3.14.

B Supplementary Results

B.1 *Depth-Adaptive Superpixels*

The following figures show more evaluation results for the *Depth-Adaptive Superpixels* algorithm from section §3.

- Fig. B.1 and fig. B.1 show superpixels for a set of input colour images (depth images are not shown)
- Fig. B.3 compares *DASP* against reference methods using the boundary recall and undersegmentation error metrics. In these and the following figures the diagrams in the left column display the measure for a varying number of superpixels and a the number of iterations for *DALIC* fixed to 5. The diagrams in the right column display the measure for a varying number of *DALIC* iterations and a fixed number of superpixels of 1000. Values are the mean over the whole dataset.
- Fig. B.4: evaluation of isoperimetric quotient
- Fig. B.5: evaluation of properties from eigenvalues
- Fig. B.6: evaluation of explained variation
- Fig. B.7: evaluation of compression error

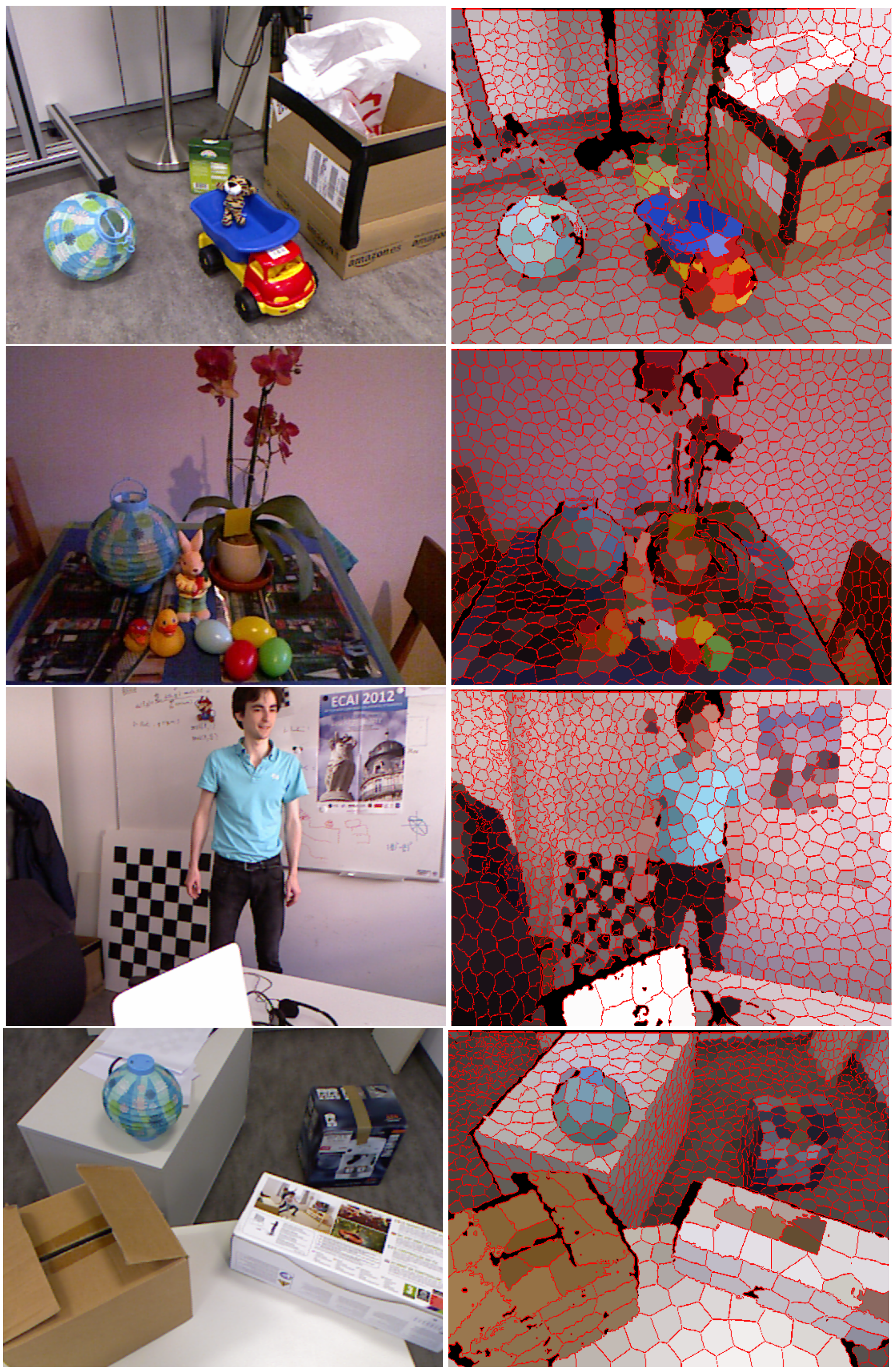


Figure B.1: Examples for *Depth-Adaptive Superpixels*

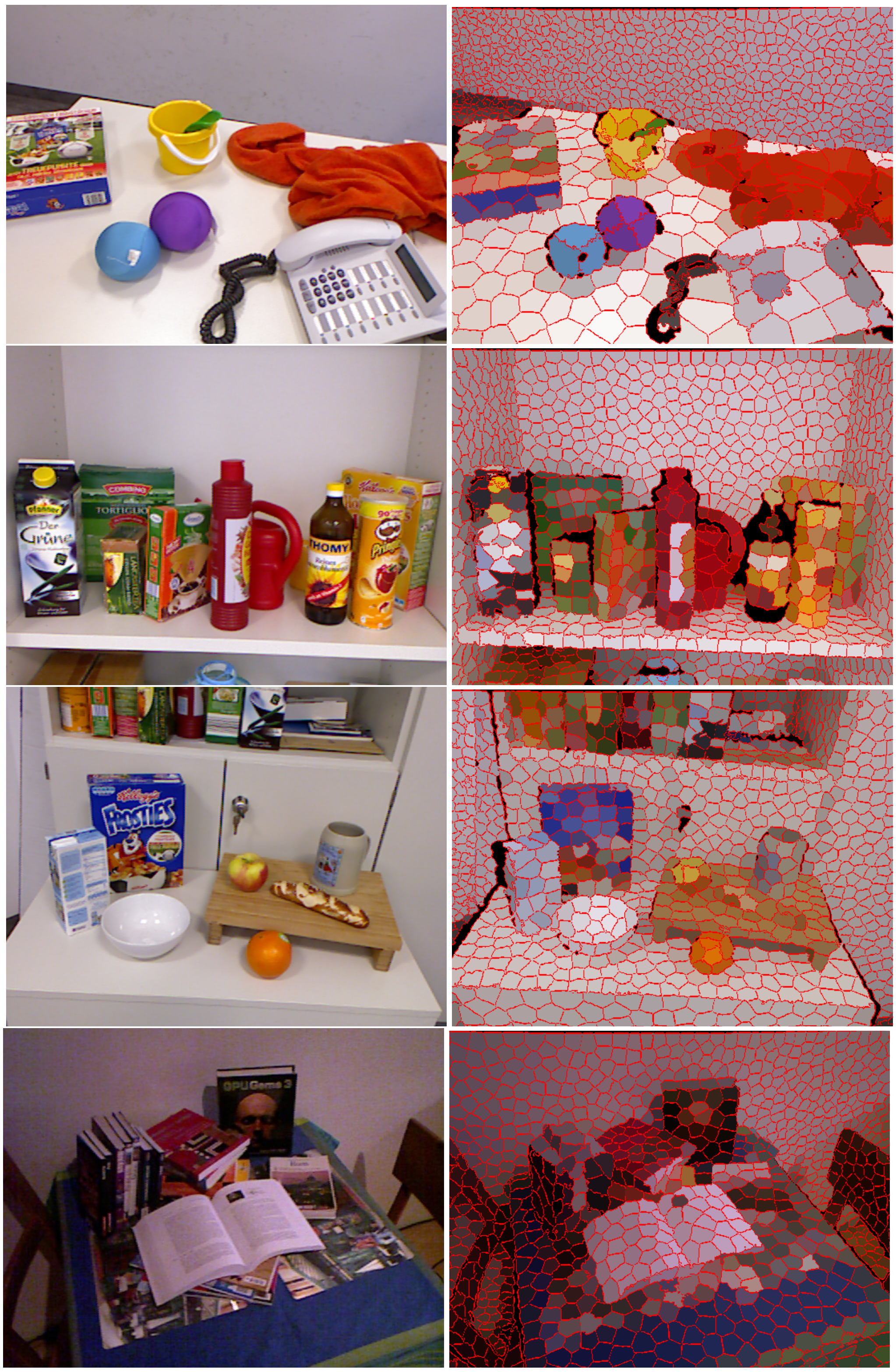


Figure B.2: Examples for *Depth-Adaptive Superpixels* (cont.)

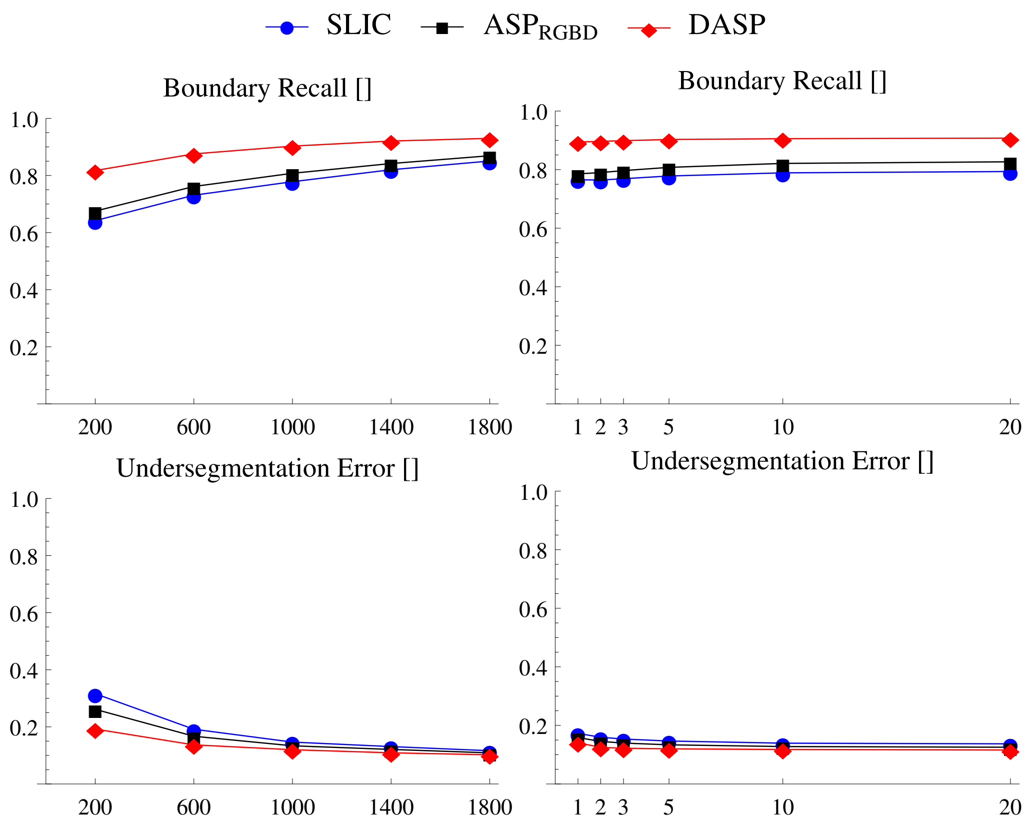


Figure B.3: **Top to bottom**: Boundary recall and undersegmentation error. **Left**: Results for varying number of superpixels. **Right**: Results for varying number of iterations for *DALIC*.

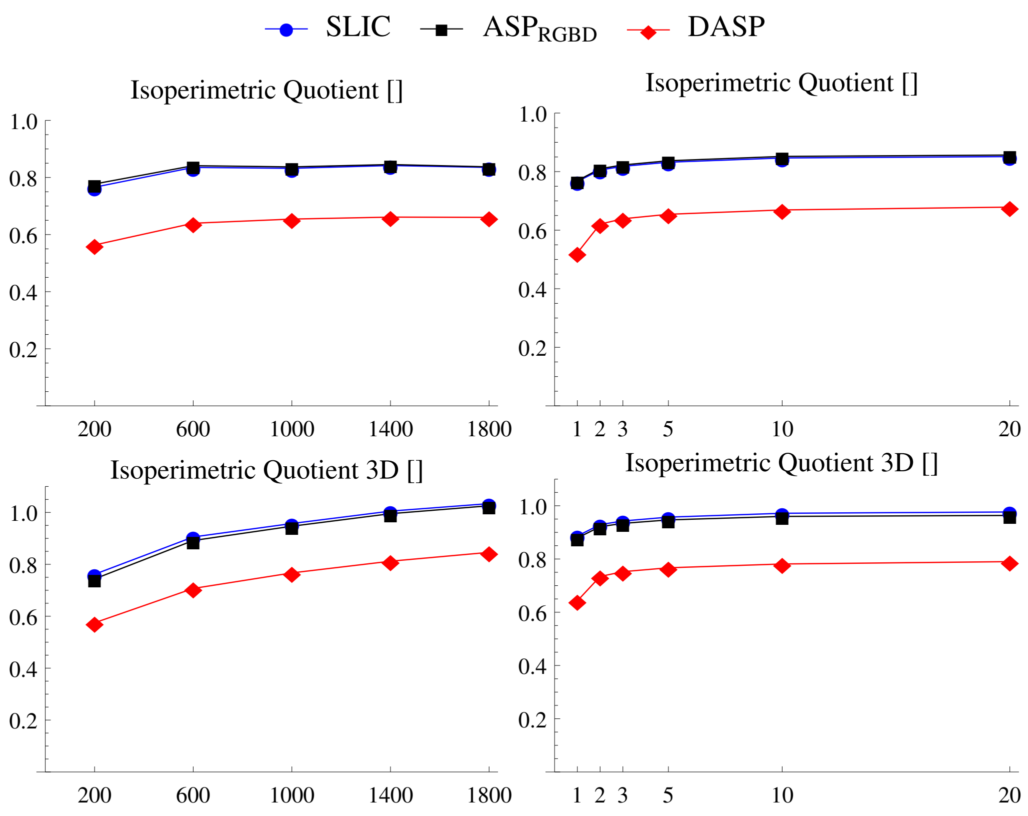


Figure B.4: **Top to bottom**: Isoperimetric quotient (2D) and isoperimetric quotient (3D). **Left**: Results for varying number of superpixels. **Right**: Results for varying number of iterations for *DALIC*.

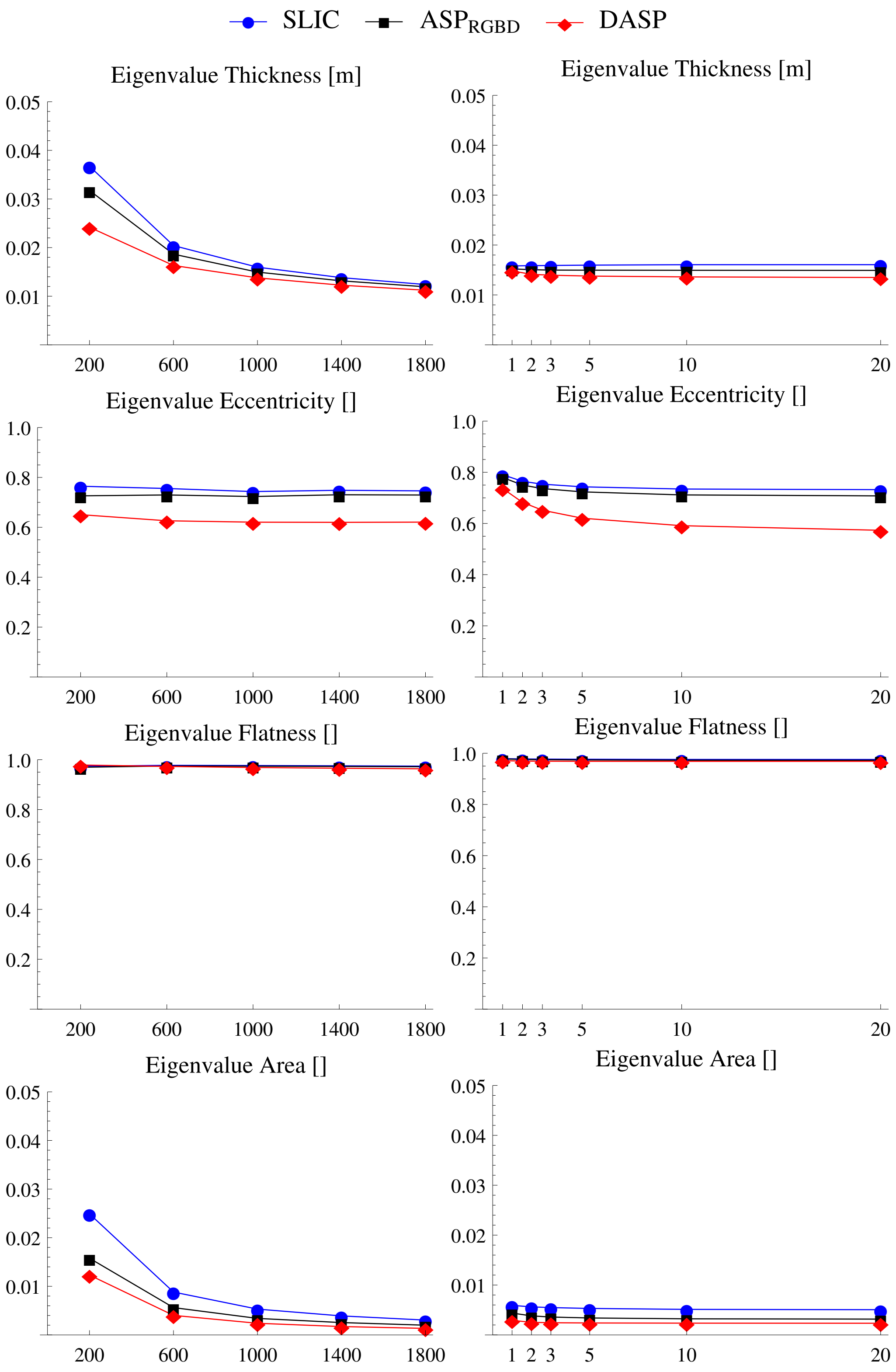


Figure B.5: **Top to bottom**: metrics computed from superpixel eigenvalue analysis: Thickness, Eccentricity, Flatness and Area. **Left**: Results for varying number of superpixels. **Right**: Results for varying number of iterations for *DALIC*.

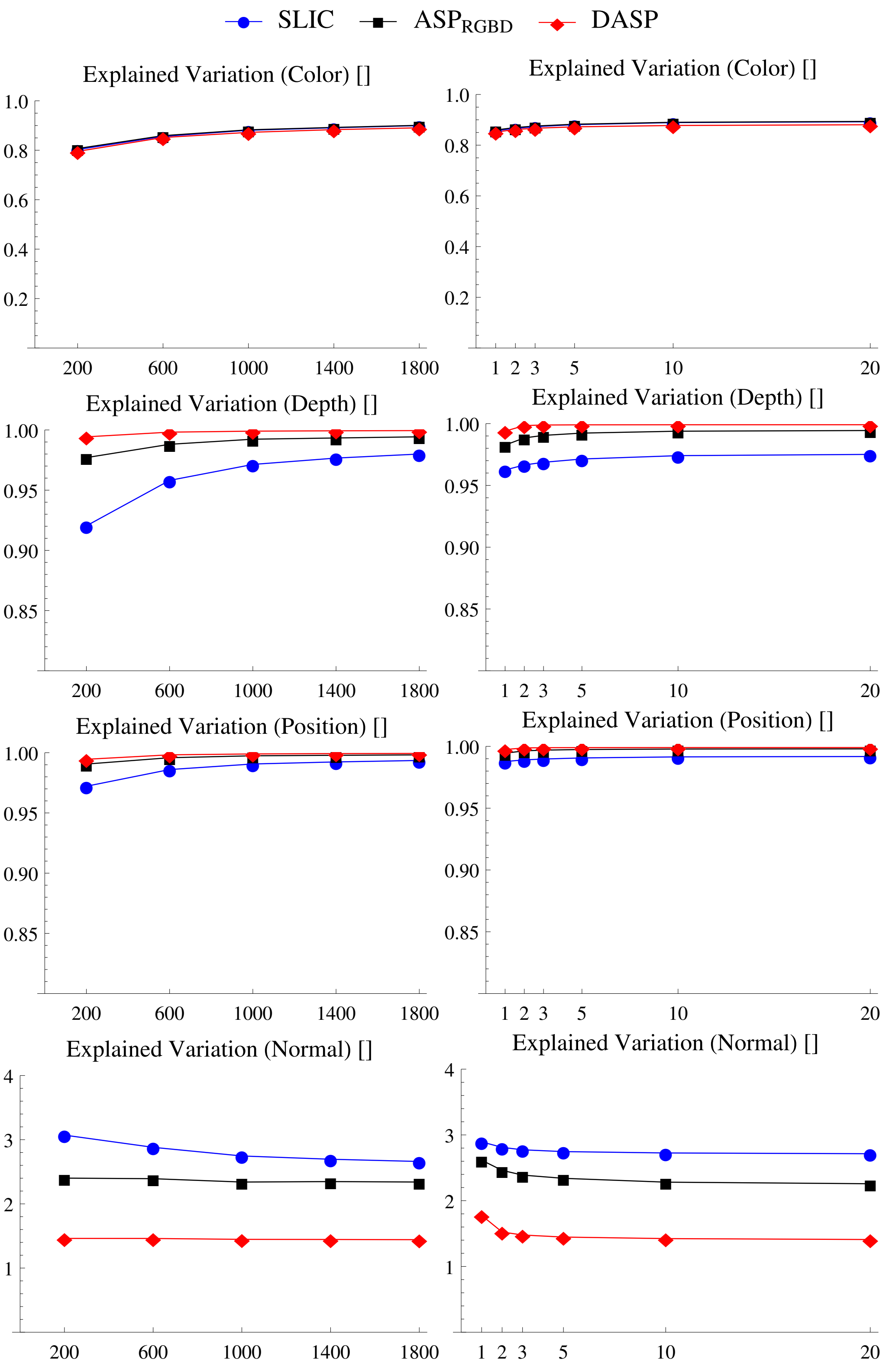


Figure B.6: **Top to bottom**: Expected variation for colour, depth and 3D position. **Left**: Results for varying number of superpixels. **Right**: Results for varying number of iterations for *DALIC*.

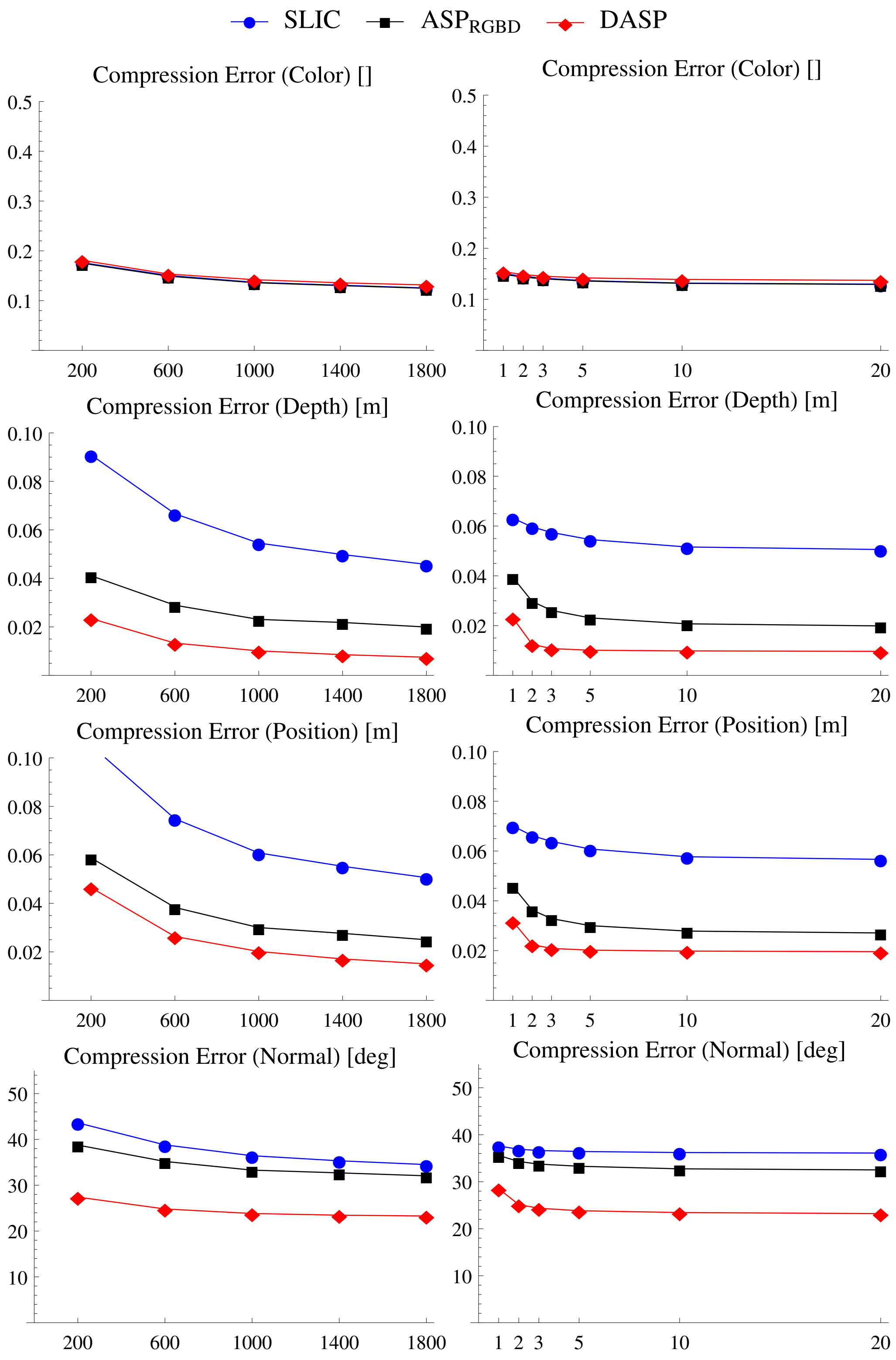


Figure B.7: **Top to bottom**: Compression error for colour, depth, 3D position and 3D normal. **Left**: Results for varying number of superpixels. **Right**: Results for varying number of iterations for *DALIC*.

B.2 *Event-based Particle Filter*

The following figures show more results for the *Event-based Particle Filter* algorithm from section §6.

- Fig. B.8 shows a comparison of tracked path an actual path in the simulated robot self-localization scenario from §6.5
- Fig. B.9 shows a comparison of tracked path an ground truth in the experimental robot self-localization scenario from §6.5

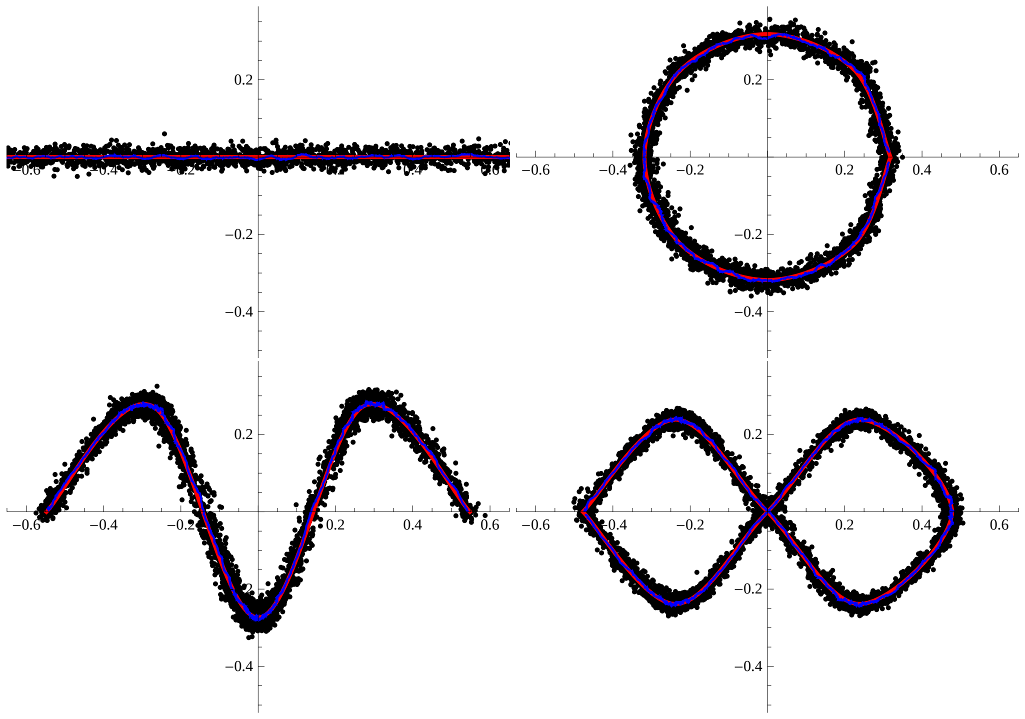


Figure B.8: Tracking results from simulation for four scenarios. Depicted are raw tracking results (black), smoothed tracking results using a mean filter (blue) and ground truth (red). Axes units are in meters.

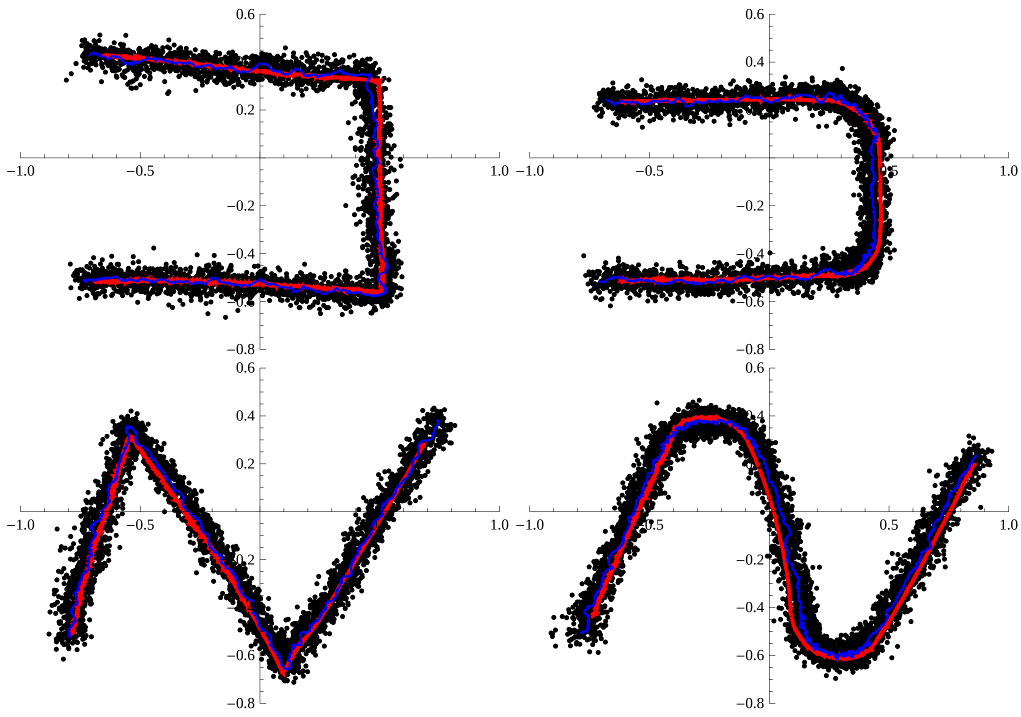


Figure B.9: Experimental tracking results for four scenarios compared to ground truth from an overhead tracking system. Depicted are raw tracking results (black), smoothed tracking results using a mean filter (blue) and ground truth (red). Axes units are in meters.

B.3 *Event-Based SLAM*

The following figures show more results for the *Event-Based SLAM* algorithm from section §7.

- Fig. B.10 and fig. B.11 compare the tracked path against ground truth
- Fig. B.12 and fig. B.13 show the development of path and map over time.

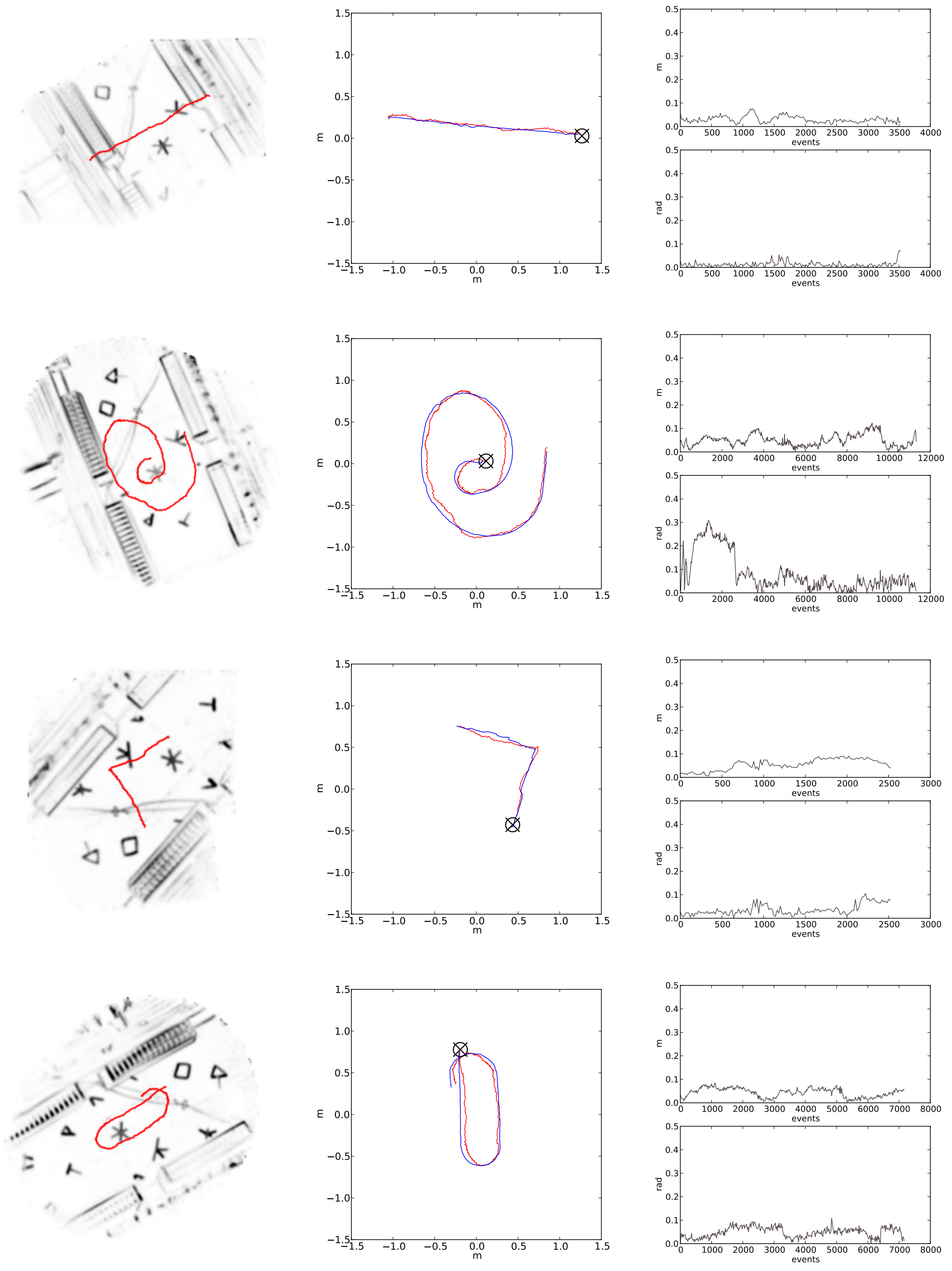


Figure B.10: **Top to bottom:** Three examples out of a total of 40 from the dataset. **Left:** Map and path as created by our method. **Middle:** Trajectories resulting from our method (red) and the external tracking system (blue). The trajectory starting point is marked with X. **Right:** Positional and rotational error over event time.

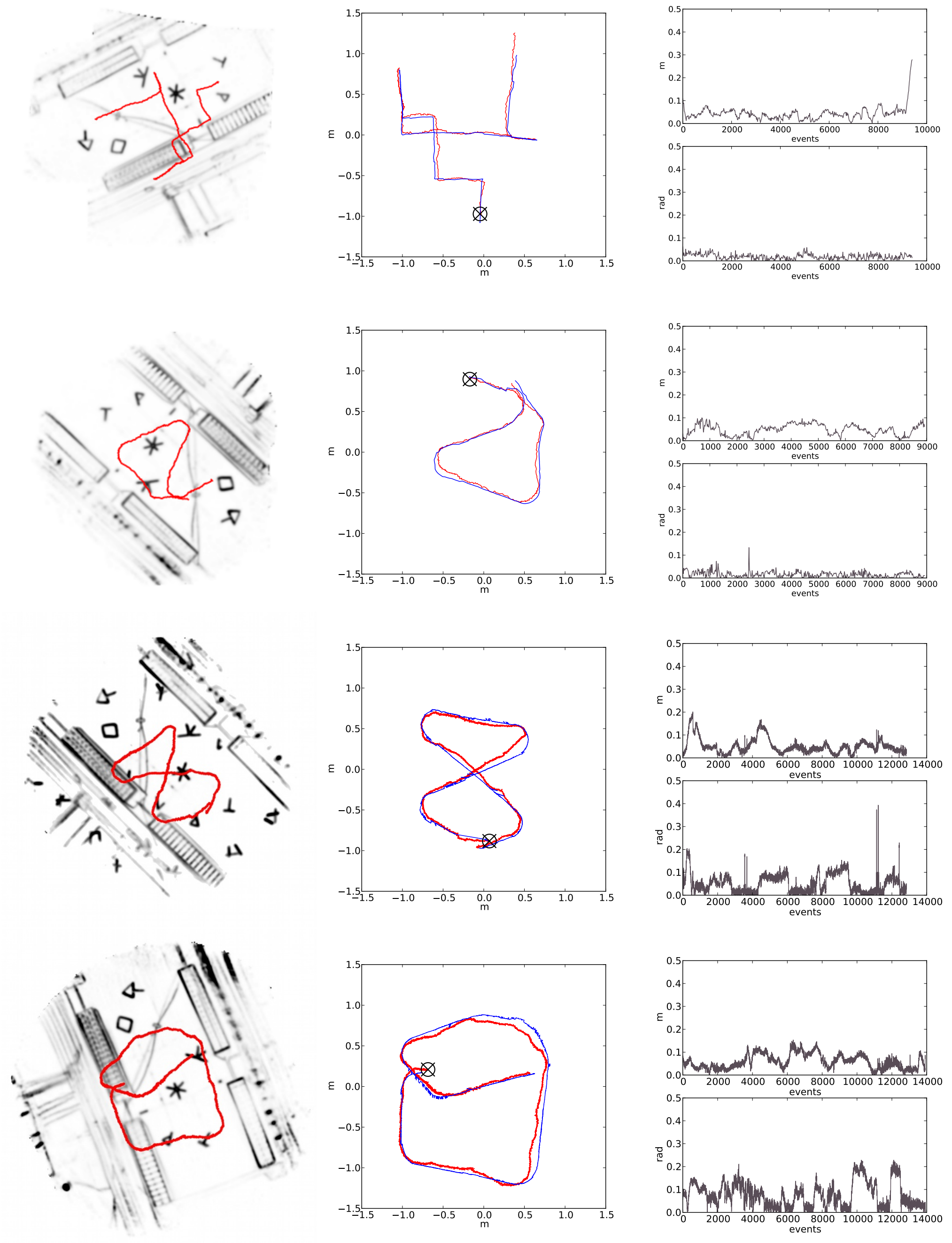


Figure B.11: **Top to bottom:** Three more examples out of a total of 40 from the dataset. **Left:** Map and path as created by our method. **Middle:** Trajectories resulting from our method (red) and the external tracking system (blue). The trajectory starting point is marked with a X. **Right:** Positional and rotational error over event time.

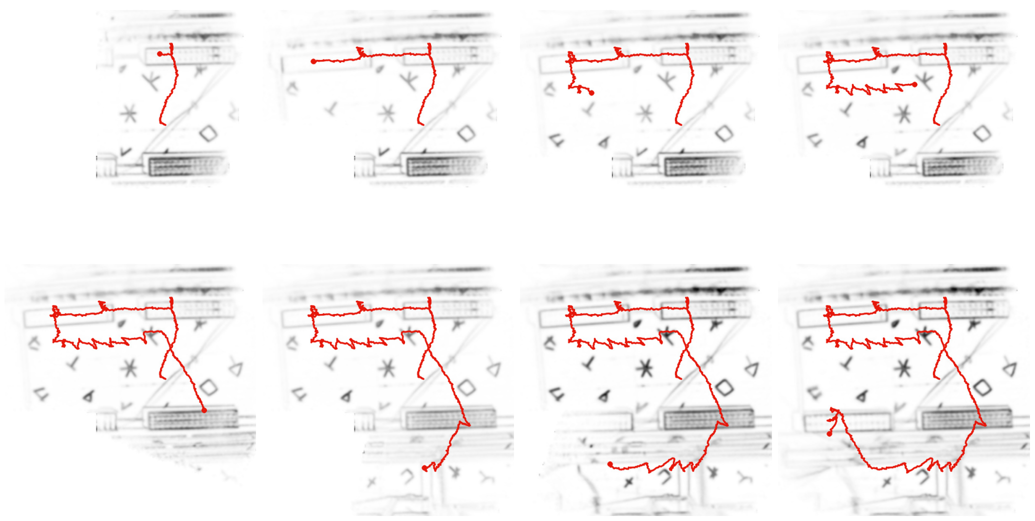


Figure B.12: Example for map and path generation over time. Displayed is a time series of map and path at fixed time intervals. In this scenario the robot moved on its own while exploring the environment. The wiggly parts of the trajectory indicate that the robot hit an environment obstacle on the ground.

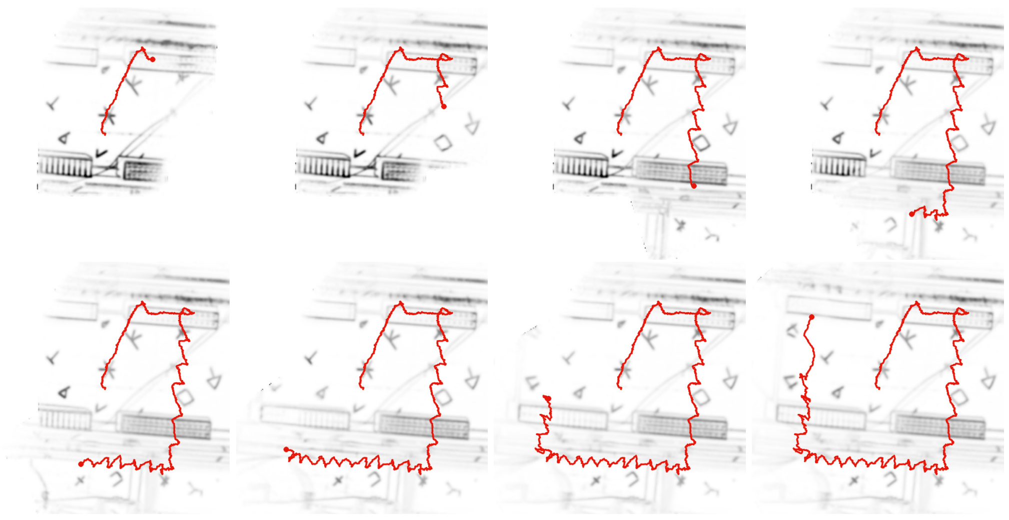


Figure B.13: A second example like in fig. B.12.

B.4 *Event-Based 3D SLAM*

The following figures show more results for the *Event-Based 3D SLAM* algorithm from section §8.

- Fig. B.14 compares tracked paths against ground truth
- Fig. B.15 shows parameter sweeps and results for the positional root-mean-square error and the runtime performance
- Fig. B.16 shows more parameter sweeps and results for the positional root-mean-square error and the runtime performance

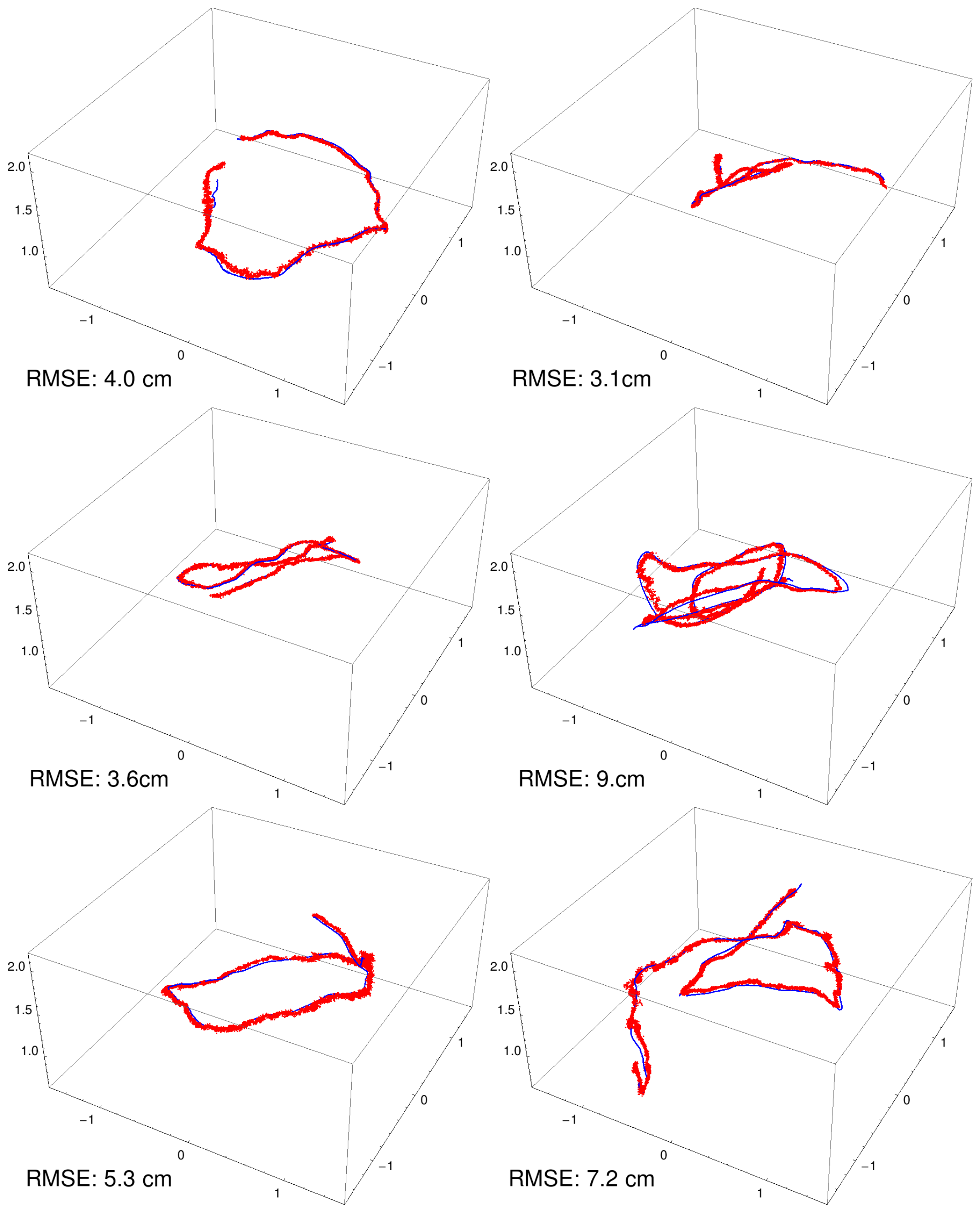


Figure B.14: 6 examples for trajectories tracked with *Event-Based 3D SLAM* (red) compared against grount truth (blue) from an overhead tracking system.

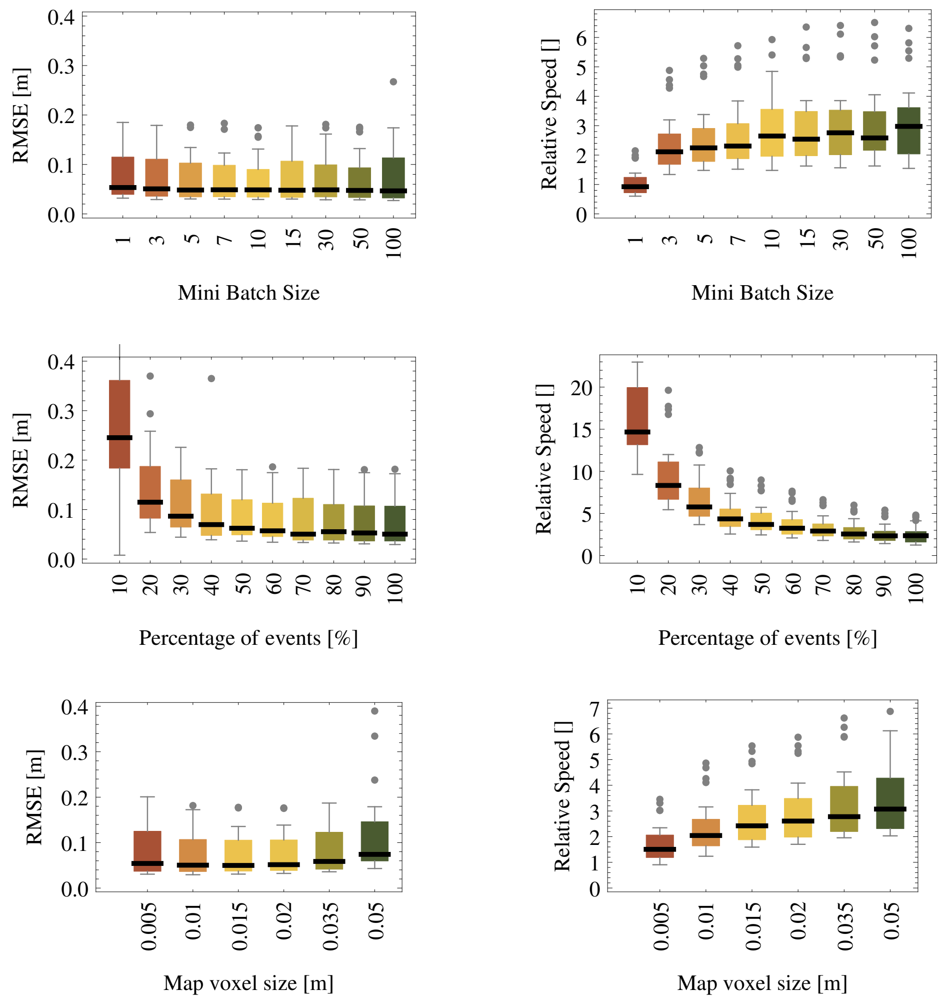


Figure B.15: RMSE and execution time relative to take duration plotted for the following parameters: Mini-batch size *B* (top), percentage of processed events (middle) and voxel size (bottom).

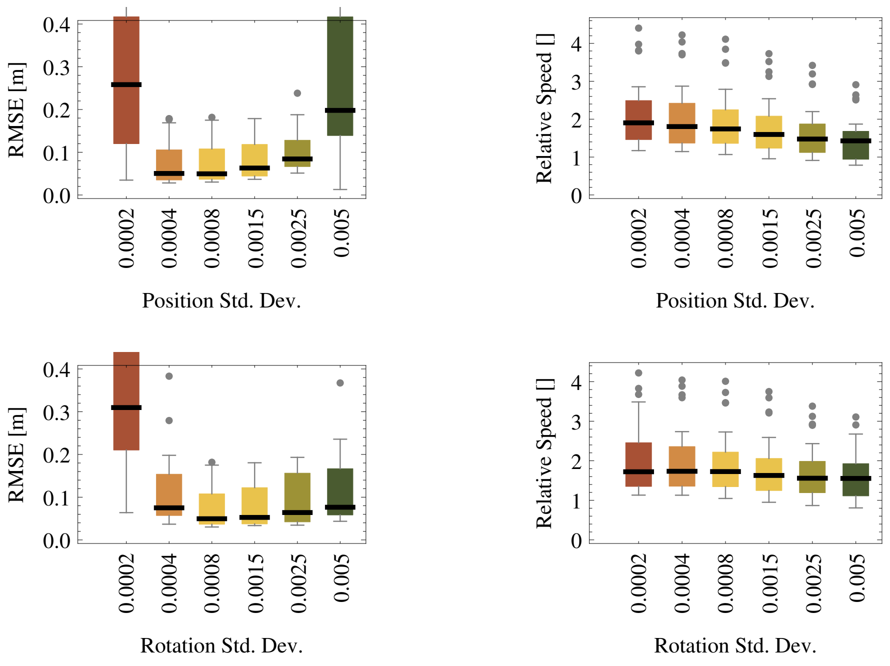


Figure B.16: RMSE and execution time relative to take duration plotted for the following parameters: Standard deviation for position diffusion (top) and angular standard deviation for rotation diffusion (bottom).

Acknowledgements

I would like to thank Prof. Jörg Conradt for introducing me to the world of dynamic vision sensors, for the keen eye he had over my research, and especially for his support during the last year of my PhD. It is an honor for me to have Prof. Daniel Cremers as an supervisor and I would like to thank him for his time and his valuable scientific advice. Additionally, I would like to thank Prof. Michael Beetz for his financial support and guidance during my time in his research group.

I am indebted to my many colleagues to support me and I would like to thank especially David Gossow for helping me with my first scientific publication. I would like to thank Magnus Eberle, Nicolai Waniek, Daniel Nyga, Ulrich Klank, Karinne Ramirez and Alexander Schick for many fruitful scientific discussions and the successfull collaboration. Additionally, I would like to show my gratitude to my students Raoul Hoffmann and David Adrian, thanks to them the recording of evaluation and ground truth data was much less exasperating.

Last but not least my special thanks go to Franziska Löhrer for the many hours she had to listen to stories about all the strange bugs living in my computer.

Bibliography

[1] https://commons.wikimedia.org/wiki/File:New_Bond_Street_1_db.jpg, Authors of the Wikimedia Commons. This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[2] https://en.wikipedia.org/wiki/File:6n-graf.svg, Wikimedia Commons.

[3] *Openvdb*. http://www.openvdb.org/.

[4] *The spirit of '43*. https://commons.wikimedia.org/wiki/File:The_Spirit_of_43-Donald_Duck,_cropped_version.jpg, Public domain due the fact that it was created for the US Government.

[5] Achanta, R., A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk: *Slic superpixels*. Technical report, EPFL, 2010.

[6] Alon, Noga: *Eigenvalues and expanders*. Combinatorica, 6(2):83–96, 1986.

[7] Arbeláez, P., M. Maire, C. Fowlkes, and J. Malik: *Contour detection and hierarchical image segmentation*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011, ISSN 1939-3539.

[8] Arbeláez, Pablo: *Boundary extraction in natural images using ultrametric contour maps*. IEEE Computer Society Workshop on Perceptual Organization in Computer Vision (POCV), 2006.

[9] Balzer, M., T. Schlömer, and O. Deussen: *Capacity-constrained point distributions: a variant of Lloyd's method*, volume 28. ACM, 2009.

[10] Besl, Paul J and Neil D McKay: *Method for registration of 3-d shapes*. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[11] Boppana, Ravi B: *Eigenvalues and graph bisection: An average-case analysis*. In *28th Annual Symposium on Foundations of Computer Science*, pages 280–285, 1987.

[12] Bylow, E., J. Sturm, C. Kerl, F. Kahl, and D. Cremers: *Real-time camera tracking and 3d reconstruction using signed distance functions.* In *Robotics: Science and Systems Conference (RSS)*, June 2013.

[13] Cézanne, Paul: *Les joueurs de cartes.* `https://commons.wikimedia.org/wiki/File:Cezanne_The_Card_Players_Barnes.jpg`, This is a faithful photographic reproduction of an original two-dimensional work of art which is in the public domain as the author is dead for more than 107 years.

[14] Chan, Antoni B, Nuno Vasconcelos, and Gert RG Lanckriet: *Direct convex relaxations of sparse svm.* In *24th international conference on Machine learning*, pages 145–153, 2007.

[15] Cheeger, Jeff: *A lower bound for the smallest eigenvalue of the laplacian.* Problems in analysis, 625:195–199, 1970.

[16] Cheng, Ming Ming, Guo Xin Zhang, Niloy J Mitra, Xiaolei Huang, and Shi Min Hu: *Global contrast based salient region detection.* In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 409–416, 2011.

[17] Chung, F. R. K.: *Spectral Graph Theory. Providence, RI:.* American Mathematical Society, 1997.

[18] Comaniciu, D and P Meer: *Mean shift: a robust approach toward feature space analysis.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5):603–619, 2002, ISSN 01628828.

[19] Conradt, Jorg, Raphael Berner, Matthew Cook, and Tobi Delbruck: *An embedded aer dynamic vision sensor for low-latency pole balancing.* In *IEEE Workshop on Embedded Computer Vision*, 2009.

[20] Cotton, Frank Albert, Geoffrey Wilkinson, Carlos A Murillo, and Manfred Bochmann: *Advanced inorganic chemistry*, volume 5. Wiley New York, 1988.

[21] Donath, William E and Alan J Hoffman: *Lower bounds for the partitioning of graphs.* IBM Journal of Research and Development, 17(5):420–425, 1973.

[22] Doucet, A., J.F.G. de Freitas, K. Murphy, and S. Russel: *Rao-blackwellized partcile filtering for dynamic bayesian networks.* In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 176–183, 2000.

[23] Dunbar, Daniel and Greg Humphreys: *Using scalloped sectors to generate poisson-disk sampling patterns.* Technical report, Tech. Rep. CS-2006-08, University of Virginia, 2006.

[24] Endres, F., J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard: *An evaluation of the RGB-D SLAM system.* In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MA, USA, May 2012.

[25] Fattal, R.: *Blue-noise point sampling using kernel density model.* ACM Transactions on Graphics (SIGGRAPH), 2011, ISSN 07300301.

[26] Felzenszwalb, Pedro F. and Daniel P. Huttenlocher: *Efficient graph-based image segmentation.* International Journal of Computer Vision, 59(2):167–181, 2004, ISSN 0920-5691.

[27] Fiedler, Miroslav: *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory.* Czechoslovak Mathematical Journal, 25(4):619–633, 1975.

[28] Forster, Otto: *Analysis 2.* vieweg, 1977.

[29] Grisetti, G., C. Stachniss, and Burgard Wolfgang: *Improved techniques for grid mapping with rao-blackwellized particle filters.* IEEE Transactions on Robotics, 23:34–46, 2007.

[30] Hartley, Richard and Andrew Zisserman: *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000.

[31] Heyde, Manfred. `https://commons.wikimedia.org/wiki/File:UsseSchloss.jpg`, This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[32] Hoffmann, Raoul, David Weikersdorfer, and Jörg Conradt: *Autonomous indoor exploration with an event-based visual slam system.* European Conference on Mobile Robots, 2013.

[33] Hong, Byung Woo and Michael Brady: *A topographic representation for mammogram segmentation.* In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 730–737. 2003.

[34] Huhle, Benjamin, Timo Schairer, Sebastian Herholz, Andreas Schilling, and Wolfgang Straßer: *Sparse registration-3d reconstruction from pairs of 2d line scans.* 2013.

[35] Hunt, Robert William Gainer: *The reproduction of colour.* Wiley, 2005.

[36] Hunter, Richard S: *Photoelectric color difference meter.* Josa, 48(12):985–993, 1958.

[37] Isard, M and A Blake: *Condensation conditional density propagation for visual tracking.* International journal of computer vision, 29(1):5–28, 1998.

[38] Itti, Laurent: *Automatic foveation for video compression using a neurobiological model of visual attention.* Image Processing, IEEE Transactions on, 13(10):1304–1318, 2004.

[39] Itti, Laurent and Christof Koch: *A saliency-based search mechanism for overt and covert shifts of visual attention.* Vision research, 40(10-12):1489–1506, 2000.

[40] Kato, H., M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana: *Virtual object manipulation on a table-top ar environment.* In *ACM International Symposium on Augmented Reality (ISAR 2000)*, pages 111–119. Ieee, 2000, ISBN 0-7695-0846-4.

[41] Kennedy, J. and R. Eberhart: *Particle swarm optimization.* In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[42] Kerl, Christian, Jürgen Sturm, and Daniel Cremers: *Dense visual slam for rgb-d cameras.* In *International Conference on Intelligent Robot Systems (IROS)*, 2013.

[43] Kerl, Christian, Jürgen Sturm, and Daniel Cremers: *Robust odometry estimation for rgb-d cameras.* In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[44] Kim, K, T H Chalidabhongse, D Harwood, and L Davis: *Background modeling and subtraction by codebook construction.* In *International Conference on Image Processing*, pages 3061–3064, 2004.

[45] Knossow, D., A. Sharma, D. Mateus, and R. Horaud: *Inexact matching of large and sparse graphs using laplacian eigenvectors.* Graph-Based Representations in Pattern Recognition, pages 144–153, 2009.

[46] Krzemiński, Michal. and Justyna Signersak: *Properties of graphs in relation to their spectra.*

[47] Lagae, Ares and Philip Dutré: *A comparison of methods for generating poisson disk distributions.* Computer Graphics Forum, 2008, ISSN 0167-7055.

[48] Levinshtein, A., A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi: *Turbopixels: Fast superpixels using geometric flows.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009, ISSN 1939-3539.

[49] Li, Yuanqing, Andrzej Cichocki, Shun ichi Amari, Sergei Shishkin, Jianting Cao, and Fanji Gu: *Sparse representation and its applications in blind source separation.* Advances in neural information processing systems, 16:241, 2004.

[50] Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck: *A 128x128 120db 15us latency asynchronous temporal contrast vision sensor.* IEEE Journal of Solid State Circuits, 43(2):566–576, 2007.

[51] Lloyd, Stuart: *Least squares quantization in pcm.* IEEE Transactions on Information Theory, 28(2):129–137, 1982.

[52] MacQueen, James *et al.*: *Some methods for classification and analysis of multivariate observations.* In *5th Berkeley symposium on mathematical statistics and probability*, volume 1, page 14, 1967.

[53] MeilPa, Marina and Jianbo Shi: *Learning segmentation by random walks.* 2001.

[54] Montemerlo, Michael, Sebastian Thrun, Daphne Koller, and Ben Wegbreit: *Fastslam: A factored solution to the simultaneous localization and mapping problem.* In *AAAI/IAAI*, pages 593–598, 2002.

[55] Müller, Georg and Jörg Conradt: *A miniature low-power sensor system for real time 2d visual tracking of led markers.* In *IEEE International Conference on Robotics and Biomimetics*, 2011.

[56] Newcombe, Richard A, David Molyneaux, David Kim, Andrew J Davison, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon: *Kinectfusion: Real-time dense surface mapping and tracking.* In *10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[57] Nikolayev, Dmitry and Tatjana Savyolova: *Normal distribution on the rotation group so (3).* Textures and Microstructures, 29(3):201–234, 1997.

[58] Oikonomidis, I., N. Kyriazis, and A. Argyros: *Efficient model-based 3d tracking of hand articulations using kinect.* 2011.

[59] Oikonomidis, Iasonas, Nikolaos Kyriazis, and Antonis A Argyros: *Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints.* In *IEEE International Conference on Computer Vision*, pages 2088–2095, 2011.

[60] Oikonomidis, Iasonas, Nikolaos Kyriazis, and Antonis A Argyros: *Tracking the articulated motion of two strongly interacting hands.* In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1862–1869, 2012.

[61] Park, Mee Young and Trevor Hastie: *L1-regularization path algorithm for generalized linear models.* Journal of the Royal Statistical Society: Series B (Statistical Methodology), 69(4):659–677, 2007.

[62] Pennec, Xavier: *Computing the mean of geometric features application to the mean rotation.* 1998.

[63] Perbet, F. and A. Maki: *Homogeneous superpixels from random walks.* In *IAPR Conference on Machine Vision Applications*, 2011.

[64] Ren, Xiaofeng and Jitendra Malik: *Learning a classification model for segmentation.* IEEE International Conference on Computer Vision, 2003.

[65] Rogister, Paul, Ryad Benosman, Sio Hoi Ieng, Patrick Lichtsteiner, and Tobi Delbruck: *Asynchronous event-based binocular stereo matching.* IEEE Transactions on Neural Networks and Learning Systems, 23(2):347–353, 2012.

[66] Rusu, Radu Bogdan: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.* PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[67] Sakaki, Takeshi: *Earthquake shakes twitter users : Real-time event detection by social sensors.* pages 851–860, 2010.

[68] Schick, A., M. Fischer, and R. Stiefelhagen: *Measuring and evaluating the compactness of superpixels.* In *International Conference on Pattern Recognition*, 2012.

[69] Schmaltz, C., P. Gwosdek, A. Bruhn, and J. Weickert: *Electrostatic halftoning.* In *Computer Graphics Forum*, volume 29, pages 2313–2327, 2010.

[70] Schmidt, F. and H. G. Schaible: *Neuro- und Sinnesphysiologie.* Springer, 1993.

[71] Schraml, S., A. N. Belbachir, N. Milosevic, and P. Schön: *Dynamic stereo vision system for real-time tracking.* In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1409–1412, 2010.

[72] Shi, J. and J. Malik: *Normalized cuts and image segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.

[73] Shotton, J, A Fitzgibbon, M Cook, T Sharp, M Finocchio, R Moore, A Kipman, and A Blake: *Real-time human pose recognition in parts from single depth images.* IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[74] Snyder, John Parr and Philip M Voxland: *An album of map projections.* 1989.

[75] Umeyama, S.: *An eigendecomposition approach to weighted graph matching problems.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(5):695–703, 1988.

[76] Vedaldi, Andrea and Stefano Soatto: *Quick shift and kernel methods for mode seeking.* In *European Conference on Computer Vision*, pages 705–718. Springer, 2008.

[77] Vincent, Luc and Pierre Soille: *Watersheds in digital spaces: an efficient algorithm based on immersion simulations.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6):583–598, 1991.

[78] Wang, Shu, Huchuan Lu, Fan Yang, and Ming hsuan Yang: *Superpixel tracking.* Electrical Engineering, page 2011, 2011.

[79] Wang, Xianfu: *Volumes of generalized unit balls.* Mathematics Magazine, 78/5:390–395, 2005.

[80] Weikersdorfer, David and Jörg Conradt: *Event-based particle filtering for robot self-localization.* In *IEEE International Conference on Robotics and Biomimetics*, pages 866 – 870, 2012.

[81] Weikersdorfer, David, David Gossow, and Michael Beetz: *Depth-adaptive superpixels.* In *21st International Conference on Pattern Recognition*, pages 2087 – 2090, 2012.

[82] Weikersdorfer, David, Raoul Hoffmann, and Jörg Conradt: *Simultaneous localization and mapping for event-based vision systems.* In *International Conference on Computer Vision Systems*, 2013.

[83] Weikersdorfer, David, Alexander Schick, and Daniel Cremers: *Depth-adaptive supervoxel for efficient rgb-d video analysis.* In *IEEE International Conference on Image Processing*, 2013.

[84] Wu, Zhenyu and Richard Leahy: *An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(11):1101–1113, 1993.

[85] Xu, C., C. Xiong, and J.J. Corso: *Streaming hierarchical video segmentation.* In *European Conference on Computer Vision*, 2012.

[86] Yang, X.S., S. Deb, and S. Fong: *Accelerated particle swarm optimization and support vector machine for business optimization and applications.* Networked Digital Technologies, pages 53–66, 2011.

[87] Yu, Stella X and Jianbo Shi: *Multiclass spectral clustering.* In *9th IEEE International Conference on Computer Vision*, pages 313–319, 2003.

[88] Zeng, G., P. Wang, J. Wang, R. Gan, and H. Zha: *Structure-sensitive superpixels via geodesic distance.* In *IEEE International Conference on Computer Vision*, pages 447–454, 2011.

[89] Zhang, Zhengyou: *A flexible new technique for camera calibration.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

[90] Zhu, Ji, Saharon Rosset, Trevor Hastie, and Rob Tibshirani: *1-norm support vector machines.* Advances in neural information processing systems, 16(1):49–56, 2004.

[91] Zivkovic, Z: *Improved adaptive gaussian mixture model for background subtraction.* In *17th International Conference on Pattern Recognition*, volume 2, pages 28–31, 2004.

[92] Zou, Hui, Trevor Hastie, and Robert Tibshirani: *Sparse principal component analysis.* Journal of computational and graphical statistics, 15(2):265–286, 2006.

List of Figures

List of Tables

List of Notations and Abbreviations

L	Logistic function $L_\alpha(x) := \frac{1}{1+e^{-\alpha x}}$
$\mathscr{P}(X)$	Probability of X
$\mathscr{N}$	Random sample from a Gaussian normal distribution
$\mathscr{U}$	Random sample from a uniform distribution
$\mathbb{N}$	Natural numbers (integers)
$\mathbb{Z}$	Positive integers (including 0)
$\mathbb{R}$	Real numbers
$\mathbb{R}_+$	Positive real numbers (including 0)
$\mathrm{SE}(n)$	Special Euclidean Group which describes orientation preserving isometries of an n-dimensional rigid bodies
$\mathrm{SO}(n)$	Special Orthogonal Group which describes orientation preserving symmetries of an n-dimensional rigid bodies
Area	Area of a 2D segment
$\mathfrak{D}$	Set of superpixels
$\mathscr{F}$	Image feature space
$\mathfrak{f}$	Feature of a pixel / element
$\mathfrak{s}$	Mean feature for a superpixel or segment
$\mathscr{G}$	General graph
$\mathscr{G}_I$	Two-dimensional finite lattice graph (for images or video frames)
$\mathscr{G}_S$	Superpixel graph
$\mathscr{G}_V$	Three-dimensional finite lattice graph (for videos)

$\mathbf{I}$	Mapping $\mathscr{G}_I \to \mathscr{F}$
$\mathfrak{P}$	Partition of a set
$\mathfrak{X}$	Ground truth / reference partition of a set
$\mathfrak{H}$	Hierarchical tree of partitions
Ω	A domain in general
$\mathbf{V}$	Mapping $\mathscr{G}_V \to \mathscr{F}$
$\partial\mathfrak{P}$	Boundary of a partition
∂S	Boundary of a segment
ρ	Density function
Vol	Volume of a 3D segment
$A_\rho(\cdot, U)$	Kernel density approximation of a set of points U
E	Set of edges of a graph
$E_\rho(U)$	Error of kernel density approximation of a set of points U
S	Segment of a partition
V	Set of nodes of a graph
$W_\mathfrak{s}$	Edge weights of superpixel graph
$W_\mathbf{T}$	Edge weights of superpixel strand graph
$\mathscr{G}_\mathbf{T}$	Graph of spatio-temporal superpixel strands
$\mathbf{T}$	Spatio-temporal superpixel strand
MM	Motion model for particle filter algorithm
MM_B	Motion model for *Event-based Particle Filter* which processes B events at once
$\mathscr{R}$	Pixel coordinate space for event-based sensors, i.e. $\mathscr{R} = [0, 127]^2$
Ω	Base domain for the system state used in *EB-PF*.
$\mathscr{G}$	Gain map used in autonomous exploration with *EB-SLAM*
Γ	Base domain over which to build the map used in *EB-SLAM*.
$\mathscr{E}$	Exploration map used in autonomous exploration with *EB-SLAM*

$\mathscr{M}$	A map used for mapping the environment in a SLAM method.
$\mathscr{O}$	Occurrence map used in *EB-SLAM*
$\mathscr{Z}$	Normalization map used in *EB-SLAM*
D-eDVS	Combination of *eDVS* and PrimeSense depth sensor
$\mathscr{R}_D$	Space for events from a *D-eDVS* sensor
ASP	(Density-)Adaptive Superpixel
DALIC	Density-Adaptive Local Iterative Clustering
DASP	Depth-Adaptive Superpixels
DDS	Delta Density Sampling
PDS	Poisson Disc Sampling
RGB-D	Refers to an RGB colour space in combination with depth information
RGB	Refers to an RGB colour space
S-DASP	Depth-Adaptive Superpixel Segmentation
SPDS	Simplified Poisson Disc Sampling
T-DASP	Temporal Depth-Adaptive Superpixel
TS-ASP	Temporal-Stable (Density-)Adaptive Superpixel
UCG	Ultrametric contour graph
UCM	Ultrametric contour map