Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Datenverarbeitung

Swarm Intelligence Based Image Analysis for 2D to 3D Conversion

Ulrich Josef Kirchmaier

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzende: Prof. Dongheui Lee, Ph. D.

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Klaus Diepold
2. Prof. Paolo Remagnino, Ph. D.

Die Dissertation wurde am 3. November 2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 28. Mai 2017 angenommen.

Ulrich Josef Kirchmaier. *Swarm Intelligence Based Image Analysis for 2D to 3D Conversion.*
Dissertation, Technische Universität München, Munich, Germany, 2017.

Abstract

With the growing success of 3D cinema, also 3D TV devices for home cinema have been on the rise for several years. Yet, the majority of films and TV content historically is shot in regular 2D, thus providers of e.g. 3D TV channels are lacking content for continuous 3D broadcasting.

A solution to this issue is to transform the existing 2D films into 3D movies by adding depth information. This procedure is called *2D to 3D conversion* and consists of image feature extraction, depth map generation, and depth based image rendering, thus creating a second view from a slightly different viewpoint.

These two views lead to a 3D experience at a human viewer, due to an effect called *binocular disparity*. The small horizontal distance between the left and right eye lets us see two images of a scene that are slightly different, as coming from different angles. The displacements of the perceived objects is directly related to the distance.

Yet, humans can also estimate the depth of a scene when viewing a single 2D image, by utilizing various so called *monocular depth cues*, relations of objects and semantic features in the image, from which humans can extrapolate the depth relying on their experience.

A number of these cues can also be analyzed computationally, and a depth estimation can be created, yet with considerable variance in terms of quality and computational efficiency. Existing state of the art 2D to 3D conversion systems often aim at very different goals, varying from complex and costly human-interaction based systems for ultra high quality solutions for 3D cinema to real-time conversion systems built in current 3D TV devices, which can cause headache and sickness at the watcher after a short period of time.

Both variants are not perfect for providers, that wish to convert 2D films to 3D for broadcasting, etc., as the first variant is too expensive, and the second suffers from too low quality. Hence, the challenge is to create a fully automatic conversion system, which delivers considerable quality at proper computational speed.

In this thesis, I tackle three subtasks of the task of the problem field of 2D to 3D conversion. First I provide a concept for exhaustive image analysis. Its root consists of a novel

image segmentation algorithm named *SISeg*, that is based on the principles of Swarm Intelligence, making use of analogies from the biological phenomenon of self-assembly.

Second, I present a new contour tracking algorithm, which is called *I-BRIEF*. Therefore the very efficient BRIEF interest point descriptor and matching algorithm is extended to describe and match contour features.

Third I introduce a lightweight as proof of example for 2D to 3D conversion, which aims on delivering results of proper quality fully automatic, with high computational efficiency.

All three algorithms are evaluated on using corresponding benchmarks and compared to approaches from the literature, and their quality and computational performance are discussed. All three approaches deliver benchmark results of firm quality, at a considerably high computational performance.

The results show, that utilizing Swarm Intelligence, and especially the principle of self-assembly is well suited to the problem of image segmentation and analysis. Further, the thesis illustrates, that an efficient image and image sequence analysis is a key factor to allow for an efficient depth map generation and 2D to 3D conversion.

Contents

List of Publications

Journal Publications

U. Kirchmaier, S. Hawe, and K. Diepold. Dynamical information fusion of heterogeneous sensors for 3D tracking using particle swarm optimization. In *Information Fusion*, 12(4), pp. 275–283, 2011.

U. Kirchmaier, S. Hawe, and K. Diepold. A swarm intelligence inspired algorithm for contour detection in images. In *Applied Soft Computing*, 13(6), pp. 3118–3129, 2012.

Conference Publications

F. Keyrouz, U. Kirchmaier, and K. Diepold, Three dimensional object tracking based on audiovisual fusion using particle swarm optimization. In *International Conference on Information Fusion*, pp. 1–5, 2008.

U. Kirchmaier, S. Hawe. and K. Diepold. A line detection and description algorithm based on swarm intelligence. In *Proc. SPIE*, pp. 753-–757, 2010.

Technical Reports

M. Lück, and U. Kirchmaier. Geometrically based depth assignment using a single image. Technical report, Lehrstuhl für Datenverarbeitung, Technische Universität München, 2016.

List of Figures

List of Tables

List of Abbreviations

1NN First Nearest Neighbor
ACO Ant Colony Optimization
BFO Bacterial Foraging Algorithm
BRIEF Binary Robust Independent Elementary Features
CV Computer Vision
DTW Dynamic Time Warping
GDM Gradient Description Matrix
I-BRIEF Inter-Point-BRIEF
IP Image Processing
LBD Line Band Descriptor
LRC Left-Right Checking
MSCD Mean and Standard Deviation Curve Descriptor
MSLD Mean and Standard Deviation Line Descriptor
NNDR Nearest Neighbor Distance Ratio
PSO Particle Swarm Optimization
PSR Pixel Support Region
RAPID Real-time Attitude and Position Determination
SDS Stochastic Diffusion Search
SIFT Scale-Invariant-Feature Transform
SI Swarm Intelligence
SISeg Swarm Intelligence Segmentation
SLAM Simultaneous localization and mapping
SURF Speeded Up Robust Features
U-BRIEF Upright-BRIEF
U-SURF Upright-Speeded Up Robust Features (SURF)

Chapter 1.

Introduction

Creating three-dimensional impressions in films has been a desire of filmmakers, just almost as old as the invention of film itself. In 1890, the first, yet impracticable system for 3D Films was patented by British film pioneer William Friese-Greene. The earliest confirmed 3D film shown to a public audience was “The Power of Love”, in 1922. However, due to the enormous technical challenges and the accompanying issues in quality of experience, 3D films led rather a niche existence in its first hundred years.

In the beginning of the 21st century, 3D found its way to mainstream productions, as developments in hardware and software made it more and more applicable.

In 2009 a global hype for 3D technology was started by James Cameron’s movie “Avatar”, which is up to now the most successful movie of all time. Highly successful 3D productions succeeded, e.g. “Transformers 3” in 2011, “Gravity” in 2013, the “Avengers” movies, and “Jurassic World” in 2015. It seemingly starts to become almost the norm in Hollywood to release movies, when it comes to blockbuster action productions or also animated movies.

As a coupled development, 3D TV surged into the home cinema market in the last decade and has nowadays become a standard feature in almost any device. In course of this, pay TV providers started to offer 3D channels for these devices.

However, as the vast majority of cinematic content produced in the last 100 years, is available in 2D only, providers of 3D TV channels are lacking content to fill the program slots. Thus there exists a high interest in ways of adding depth to 2D films subsequently, which also allows studios to convert their movies into 3D for theatrical, or DVD-re release. Such a procedure is generally called *2D to 3D conversion*.

1.1. 2D to 3D Conversion

We humans perceive depth via the so-called *binocular disparity*. This means, that there is a small horizontal distance between our left and right eye, which causes the left eye to see the scene from a slightly different position than the right eye. Within these two “images” of the two eyes, all object are slightly translated, relative to each other, the disparity. This effect is depth dependent, the closer objects are to the eye, the more they are shifted to each other between the two views. The human brain can estimate the depth of all objects in the scene out of their shifts.

When creating new 3D movie content, a stereo camera system is regularly used, in which the two cameras horizontally spaced to each other film the scene, thus imitating the human eyes. This effect - also called *stereopsis*, or in case the eyes, or respectively cameras are not parallel but turned towards each other, *convergence* - displays a so-called *binocular depth cue*.

1.1.1. Monocular Depth Cues

Out of our experiences in the 3D world, we humans can also imagine the depth in scenes, which we see as two-dimensional images or image sequences, where the displacement between the two views logically cannot take effect.

In this case, the human brain estimates the depth of a scene via so-called *monocular depth cues*. These cues display knowledge, or respectively patterns, that we humans learned, which allow us to connect a perceived two-dimensional scene with a depth estimation.

Computational 2D to 3D conversion systems seek to utilize one or several of these depth cues to estimate the depth in a 2D image. The most prominent monocular depth cues are shortly introduced in the following. Further examples approaches are given for the cues. For further information on literature, please also refer to section 5.1.

Linear perspective An image is the results of a perspective projection of the 3D world. Perspective projection holds the property, that, depending on the viewpoint, parallel lines converge at infinity, at the so-called *vanishing points*, out of which one can extract the positioning of objects in space. The assumption of parallel lines, the *Vanishing lines* makes this depth cue prevalent in man-made environments, like houses, railroad tracks, etc. That is why this cue is also called *depth from geometry*. Example implementations can be found in e.g. [133, 110].

Size of objects There are several variants of cues concerning the depth ob objects. First, humans know the size of a broad range of objects and from the size of such objects in their retinal image, humans can deduce the distance to the object. This cue is called *absolute size* of objects.

Second, we can recognize two identical or similar objects in a scene, e.g. two trees, and from the size relative to each other, we can conclude their relative depth between them. This cue thus relies on the *relative size*.

These cues strongly depend on reliable recognition of objects, plus knowledge on their sizes. These necessary steps makes the cues very complex to obtain in computational depth estimation systems. See [84, 99] for example approaches.

Atmospheric scattering This cue, also called *Aerial perspective*, relates to the fact, that, due to the light scattering in the atmosphere, caused by small particles in the air, objects in a great distance appear hazy and with low contrast or color saturation. Together with knowledge of color heuristics of certain scenes, this is used to decide on depth estimation, for example in scene of distant mountains. To do so, a certain colors and color saturation is assigned a specific depth. Logically, this cue only holds for scenes, that hold a large amount of depth. This cue is implemented in [80].

Occlusion/overlapping An object, which occludes another object is perceived as being closer to the viewer than the other one. To decide, which object is the closer “occluding” one, the regularity of the two boundaries is considered. This prerequisites that the more distant objects is not fully occluded. For implementations see [47, 5].

Defocus and blur If an image or film is focused on an object of a specific distance, other objects become more and more blurred, depending on how far away they are from the distance of the object in focus. Analyzing the amount of blur on object borders, one can thus establish a relative depth map between the objects. To achieve depth estimates, the assumption that the object in focus is close to the object and thus increasing blur resembles growing distance, is often taken. Yet this assumption does not necessarily hold. Further, the image of film must be shot with a considerably shallow depth of field, in order to have a significant amount of blur in the defocused areas. See [135, 79, 44] for example implementations.

Texture gradient Eyes can recognize the fine details of textured objects easier if they are nearby. The texture gets weaker if the same object is farther away from the viewpoint, which gives a sense of depth. Examples are the perceptibility of bricks in a wall, or paving stones. To a certain degree, this cue is related to depth from geometry. [70] relies on this cue.

Elevation, relative height Humans estimate objects that are close to the bottom of an image as close, and objects which are near the horizon as distant. This corresponds to the majority of perceived scenes, which base on a horizontal ground perceived from a parallel viewpoint, which hits the horizon in some point. Simple computational approaches to 2D to 3D conversion therefore use a so-called ramp, i.e. a horizontal ground plane, as this assumption fits many scenes. Often, this ground plane is then simply adjusted to the detected segments in the scene by adding depth jumps at top borders of objects, thus "erecting" the segments from the ground plane. Nevertheless, the assumption of connecting image height to depth causes misleading depth assignments in most typical film-shots other than outside scenes, like e.g. close-ups. The method introduced in [83] utilizes the relative height cue along with machine learning.

Statistical patterns The simplest approach to depth cues descending from statistical patterns, are certain color heuristics. The best example is the color of the sky, as he sky can always be regarded as of infinite distance. While there are more color based heuristics, that can guide a simple classification, like grass and trees, skin color, roads, water, etc., a connection to a depth statement is not that straight-forward or reliable for those. Also the sky-depth relation can possibly lead to false depth assignments, in cases, where there is only a reflection of the sky in a window.

A further color-based approach is closely linked to atmospheric scattering and bases upon the fact, that via atmospheric scattering objects with "cool" colors, like blue and violet appear to be distant, compared to objects with "warm" colors, like red and yellow. Yet, such a link between colors and depth is very weak, holding mainly in scenes with e.g. far away mountains. For example implementations see [112, 115]

In the recent years, *machine learning* algorithms have evolved to constitute a special variant of obtaining depth maps via statistical patterns. Approaches in this field utilize a database containing images plus ground truth depth. Any image is then analyzed, and visual patterns that optically resemble patterns from the database images, are assigned with a similar depth. These approaches thus do no depend on predefined models, or cues, but learn data-driven models instead. The variety of approaches arises from the choice of input features on the one hand, and learning algorithms on the other hand [90, 53, 52]. Besides that, machine learning algorithms are also used as a smart way to fuse depth cues.

In case of image sequences, there exist further depth cues, which seeks to especially utilize the depth information descending from all appearing differences between consecutive images.

Motion parallax This cue relies on the circumstance, that object closer to the viewpoint appear to move faster in the image, than objects further away. Thus, from the perceived object movement in the scene, but also from the relative displacement of still objects perceived from a moving viewpoint, one can assign a depth to the scene. Of course, this cue inherits the assumption, that all objects in the scene move with a similar speed and it prerequisites movement of either the scene or the camera. The approach in [23] is based on motion parallax.

Depth from motion This cue holds, when a rigid object moves towards - or away from - the viewpoint. Out of the object's "scaling" in the image, its change of size, humans can estimate its depth.

Structure from Motion When rigid objects, still scene and moving camera, Structure from Motion (SfM) algorithms allow to calculate simultaneously the three-dimensional positions of feature points and the three-dimensional movement of the camera. Besides the computational expense, the major drawbacks of SfM are the prerequisite to the scene being static, and the constraints to the camera movement. The movement must inherit some displacement, still cameras, pure rotational movements, or zoom are not sufficient.

Besides utilizing motion based depth cues, the information obtained via the time dimension can also be employed to increase the robustness and accuracy by fusing the consecutive single image depth maps. An overview on depth cues and 3D reconstruction can be found in e.g. [125, 39, 111].

1.1.2. Modules of a 2D to 3D Conversion System

Just as the human brain, any computational system designed to calculate the depth in 2D images or image sequences will rely on a subset of the depth cues introduced in 1.1.1. Fig. 1.1 shows the modules, such a system consists of, which at the same time display the necessary steps to be taken, to obtain stereo images from a single image.

Image and video analysis As stated above, all monocular depth cues require specific input. Examples are, region colors, objects' appearance, or segment shapes. Thus, image segmentation represents the major pre-processing step necessary to utilize these depth cues. Further cues rely on Further straight lines, or texture features, etc., which relies on further image analysis step additionally to image segmentation.

In case of image sequences, or respectively of motion based cues, such pre-processing usually consists estimating the differences, the movement of objects between frames. Such

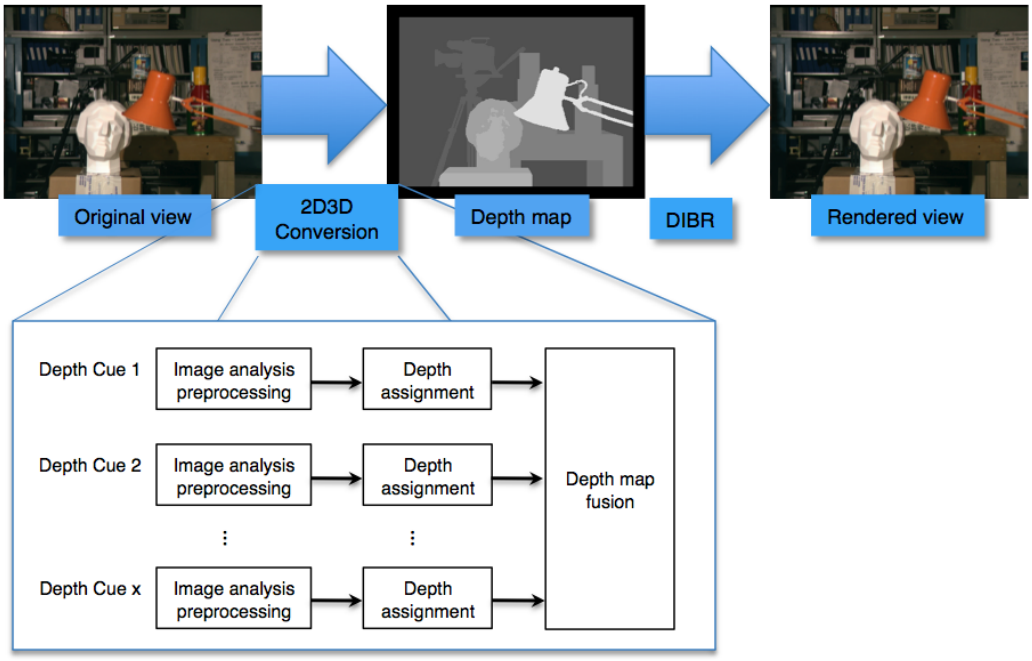


Figure 1.1.: This figure shows the modules and the procedural steps in 2D to 3D conversion. Input images "tsukuba" taken from [101]

estimation techniques consist of e.g. optical flow plus clustering or classification of displacement vectors. Alternatively, tracking techniques can be utilized.

Depth map generation A depth map is typically a 1-channel image of the same size as the input image (unless scaled for processing-specific reasons). The estimated depth of all pixels in the scene is mapped to a greyscale value, usually in 8 bit, i.e. between 0 and 255. The encoding can be arbitrary, such that either dark pixels mean far depth and white pixels are near, or vice versa.

The depth cues do not necessarily deliver an absolute estimation of depth, but a relative estimation of depth distance between objects or regions in the scene. Therefore, some mapping function must transfer the cues results to absolute depths.

As many depth cues deliver only sparse depth estimations, a further step represents calculating dense depths from these sparse estimations.

Each depth cue relies on specific assumptions about the input image, the displayed scene, which shall be interpreted. Because of these assumptions, there is always a range of scenes, where a cue is not applicable and reliable. To overcome these drawback of each

single depth cue, several cues are often combined to a final depth map. Combinations have been realized in various ways, also dynamically, or via learning algorithms.

Depth image based rendering (DIBR) The final step of 2D to 3D conversion displays the creation of the second view. Therefore, a virtual second viewpoint is defined and all objects in the scene are displaced, i.e. horizontally shifted, corresponding to the depth map created before. The appearance of all objects, their color and texture, is extracted from the original image. The shift creates "holes" in the new view, as areas in the scene become visible, which cannot be seen in the original image, and are thus unknown and must be estimated. This task is usually called *hole-filling* or *inpainting*.

While these are the major modules for 2D to 3D conversion, there are usually further steps necessary to obtain convenient 3D experiences in films. One example would be the decision on the final depth assignment in scenes, which again depends on the display used (film theaters or 3D TV). These steps are considered as outside of the scope of this work. Interested readers may refer to [108] for further information.

Also, approaches in literature, often separate the two tasks, the depth map generation and the rendering of a second view, tackling either the one or the other problem. As stated above, both have tasks include a variety of issues to be solved by themselves. Also solutions of both tasks can be arbitrarily combined, which makes a separation reasonable. Also this thesis focuses on the first task, depth map generation, concentrating on the pre-processing step.

1.1.3. The Perfect 2D to 3D Conversion System

A perfect 2D to 3D conversion system therefore must simply *always* calculate the perfect depth map, and render perfect additional virtual views. This perfect system must

- be as *generic*, resp. adaptive, flexible as possible, i.e. be able to tackle any given scenario.
- provide *accurate high-quality* results, in terms of object segmentation, assigned depth, and rendering of the additional views.
- be *robust*, e.g. to weaknesses in image capturing, sub-optimal lightning conditions, blur descending from motion or defocus, etc.
- be *fast*, i.e. have a low computation time. All of the three major tasks necessary for 2D to 3D conversion, image and video pre-processing, depth map generation, rendering

additional views, are are of high complexity. Algorithms to tackle these tasks thus often include computationally expensive steps, like optimization, iterations, numerical solutions etc.

- preferably be cheap - or at least optimizable in *resource consumption*. A lot of time consumption can be minimized by using multi-platform computation e.g. server-farms, resp. cloud based computing, or hardware accelerated, GPU-based computation or parallelization, multi-threading.

1.2. Classificaton of Converson Systems

The perfect conversion system described above would unite all typical quality and efficiency goals, which are always pursued in any software development to a certain extend, but with varying focus. In the field of 2D to 3D conversion, it is interesting to see, that there is a clear focus on the two extremes. In the following, I will describe a rough classification of existing conversion approaches.

1.2.1. High Quality versus High-Speed

Technically, existing conversion systems can be broadly classified into manual, semi-automatic, completely automatic approaches. Likewise, there are two major opposing directions, into which the most approaches in literature can be classified. These directions also imply their main field of application. Fig. 1.2 shows the classification of conversion approaches and how they are connected to the field of application.

Automatic Real-Time Conversion

The first extremum, which many approaches focus on, displays the group of real-time conversion approaches. One can find these directly implemented into 3D TV devices, or as TV-set-top boxes, which allows watch original 2D content directly in 3D. These approaches work logically completely automatic. To achieve real-time conversion including the rendering of a second view, these solutions rely on very simple conversion models and algorithms (including hardware optimization) and deliver only a low quality experience. To achieve such a low computation time, they usually utilize only one or two very simple depth estimation models. In many 3D TV included conversions, a ramp is matched into an image, thus

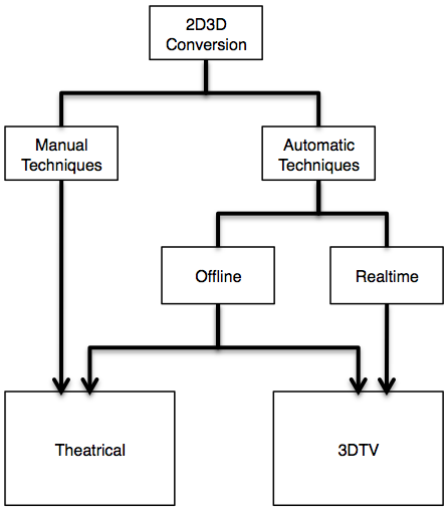


Figure 1.2.: Classification of 2D to 3D conversion approaches and the relation to their application fields. The scheme follows the classification in [108]

utilizing the *relative height-cue* in the way mentioned in section 1.1.1 [115, 129]. Additionally, a simple and rough object segmentation is merely used to enforce discontinuities in the depth map at object borders, leading to such objects being slightly "upstanding" compared their background, and thus to a 3D effect. In some approaches, these depth maps are combined with very simple additional depth maps calculated from *motion parallax* [23]. To increase computational efficiency for the segmentation of motion tracking step, 3D TV conversions often use input existing from e.g. the MPEG [50] or H.264 [91] encoding, like motion vectors. While such concepts can deceit the watcher for the first few minutes, their main drawback is, that they create depth maps, which all base on the same height-depth view, and are thus extremely wrong in a lot of situations.

Summing up, real-time can be regarded as a fun applications, but they do not provide convenient experience for more than few minutes, like watching a complete movie. Actually these very simple real time systems cause headaches or sickness to many viewers after longer consumption [108].

Manual Conversion

The opposing end on the range of conversion approaches depicts high quality conversion, on highest quality used for 3D movies. Actually, even the production of 3D films does not

exclusively shoot all scenes with stereo camera setups, but also relies heavily on 2D to 3D conversion of 2D footage in post-processing for various feasibility reasons.

High end 2D to 3D conversion is still a process with high human involvement, which makes it time-consuming and expensive, further the human operator has to be an expert to obtain high quality results.

Corresponding to the already mentioned modules of conversion, the first step in manual conversion is the rotoscoping. This term denotes the segmentation of all relevant objects, or respectively fragments, that are important for modeling the 3D scene. For image sequences, rotoscoping must be executed frame by frame and all objects and fragments need to be consistent over all frames. Besides the consistency, the exactness of the object borders displays a further quality criterion. This is the reason, why manual rotoscoping being a very time-consuming process [108].

Second, the depth of all the objects in the scene has to be carefully assigned to all objects in the scene, and of course the background. This step is sometimes also called “3D-compositing”. A simple approach create a disparity map is to shift all rotoscoped objects in the scene. Yet this can cause objects in the scene to appear as flat segments. Depending on the desired quality and the complexity of the objects, the objects themselves also have to be modeled in 3D, which is called “internal object depth”. Dealing with transparent and semi-transparent surfaces represents an additional difficult task [108].

The final step is logically the manual DIBR. The most complex task within this represents the hole filling. Compared to a machine, a human operator is better capable of estimating, what an area of a hole will most probably look like. Contrary to that, filling, i.e. in-painting such hole manually is again a very time consuming for humans, especially when high quality is desired.

In the article “2D to 3D Conversions” in his blog “Effects Corner”, the famous visual effects expert Scott Squires gives the following e.g. states, that in the 3D released movie “Transformers 3”, 77 minutes of the overall 156 minutes run-time were created by 2D to 3D conversion techniques, which is almost half of it. The remaining 79 minutes of film were made up of “true” stereo-shot scenes and entirely virtually composed scenes. The overall process of manual 2D to 3D conversion took 4-5 months. This shows the huge amount of effort to be taken in manual high-end editing, which till date only a limited number of specialized companies offer world-wide.

Offline and Semi-Automatic Conversion

Approaches, that position in the wide range between the two extremes mentioned above, can be classified into offline and semi-automatic conversion.

The distinction between real-time and offline automatic conversion techniques is of course floating, as - depending on their complexity - offline techniques might potentially be executed in real-time, if sufficient computational power is invested. However, the major difference lies in the goal of the approaches, that is primarily achieving reasonable quality, with less focus on computational complexity or run-time.

In a high quality range, semi-automatic approaches often aim to assist the manual conversion process, in order to make it more time efficient. They engage in one of the three consecutive tasks in the conversion work flow, either requiring human input as starting points, or delivering an initial result, which can be refined and corrected by humans, or both [124]. For the latter case, logically any automatic algorithm's output can be utilized via post-processing steps. Generally speaking, semi-automatic approaches seek to automatize the time-consuming steps, while the difficult - and quality-boosting - decisions are left to the human operator

There exists a huge number of algorithms for the *segmentation task*. In case of image sequences with none or compensated camera motion, objects can be computationally segmented via background subtraction. Therefore the background must be statistically modeled, via e.g. Gaussian Mixture Models, Kernel Density Estimation, or Optical Flow-based techniques.

Single image segmentation algorithms mostly rely on concepts like clustering, graph theory, edge-, contour-, or texture recognition, or statistical methods, as well as learning-based classification. To achieve high quality, many image segmentation approaches require manual input in terms of coarse marking of the segments. Active contour algorithms are one example, which iteratively fit the contours the exact object boundaries, starting from given starting points. The two segmentation approaches can also be combined to achieve a further quality boost in term of consistency. Section 3.1 provides examples for different segmentation approaches in literature.

To create reliable *3D or depth assignment*, (semi-)automatic algorithms logically utilize the depth cues introduced above, either monocular or binocular cues, or both. Basically, approaches vary in the choice of cue they rely on, the way they create the input into the cues, and also in the way, the create their final dense depth maps out of the cues.

As already mentioned, no cue is universally applicable, but dependent on the scene. Therefore, combining them can deliver a proper quality boost. A large number of solutions in literature concern themselves with the combinations of depth cues to enrich the quality of the final depth map. A variety of fusion algorithms were applied to this topic, from adaptive, weighted sums, via statistic-based fusions, to machine learning approaches. Also they choice of cues to be combined influences the results, which in the end leads to a number of variants for automatic 2D to 3D conversion.

Automatic and semi-automatic approaches can execute the basic part of the DIBR task, i.e. shifting the objects and mapping their texture, leading to significant work speed improvements. Tackling the hole-filling problem represents a way higher barrier for automatic approaches and is thus subject to vivid research, with methods based on image completion, inpainting, occlusion-filling, etc. The main difficulty is, that the hole must be filled with an estimation of the texture of the background only, and the borders of the foreground object need to stay persistent. This is why for example simple interpolation of foreground and background leads to results, which are uncomfortable for the viewer, as the foreground-object's boundaries are blurry, and also not fitting the object's boundaries in the depth dimension. Automatic approaches thus not only need to estimate the appearance of the background, they also need to estimate, which region around a hole belongs to background, which denotes two highly sophisticated subtasks.

All approaches to be found in literature are naturally positioned somewhere between these two extremes. Yet one can categorize all approaches into optimizing into one of these two directions. Therefore, the approaches cluster a lot around these two ends, with rather spare variants, which seek to optimize both aspects, the quality *and* the computational efficiency.

Thus, the key challenge is to establish a conversion, that is both automatic, fast, and of considerable quality.

1.2.2. Problems and Limitations

Estimating the depth information from a 2D image or image sequence computationally represents a complex and challenging task for a number of very general reasons.

Manifold scenarios The variance of scenarios and situations filmed can basically be regarded as infinitely high. Accordingly, estimating the 3D structure in all situations becomes infinitely complex, or even close to impossible. Examples are surrealistic scenes, or also

simply a white wall without any texture or features to be extracted. Naturally, such scenes cannot fulfill the prerequisites of any depth cue.

Image and video pre-processing As explained above, depth cues rely on various pre-processing steps. Some need appearance of regions, others straight lines, in some cases point like features are used, or also flow features. Image processing steps. e.g. segmentation itself is still a task that cannot be regarded as solved perfectly for all possible situations. The difficulty of the segmentation problem becomes evident over the fact, that not even ground truth is uniquely identifiable. Two persons usually have two different opinions on the ground truth segmentation of the same scene.

Limiting assumptions As already stated, any depth cue is based on certain model assumptions. Depending on the scenario, any assumptions can logically fail. For example, the depth from geometry expects the viewed objects to contain surfaces rectangular to each other, which does not need to be the case. Also not every blue color in a scene will be sky, and also the sky is not always blue. Prior knowledge, for example, the known size of objects can also be mislead by the perspective. One can find such limits easily for the assumptions of any cue.

1.3. Formulation of Research Problem

As stated above, approaches to 2D to 3D conversion found in literature usually tend more or less to two opposing directions. They either focus on high-quality but costly and complex solutions, often involving human interactions, or on automatic real time solutions of expendable quality. Or they focus on computation speed, delivering output not satisfactory for 3D experience over a longer time. Establishing a sufficiently fast automatic system that achieves a certain quality level needs the combination of several depth cues, aiming on the goal, that some cues can compensate the weaknesses of other in certain situations, and vice versa.

As already mentioned, the difference between the approaches from literature exists in the way they create the input into the cues, and calculate their models, as well as the way, they combine their results to a final depth map.

To allow for flexible combination of cues, one needs an image and image sequence analysis concept, which is of preferably holistic nature.

As the different cues rely on various pre-processing input, such a module has to deliver

- position, appearance and shape information of segmented regions,
- the shape of edges, or respectively region/object borders,
- position, length, and strength of straight lines,
- feature points,
- movement, or respectively tracking information,

to serve many cues at the same time.

In this thesis I focus on the task of creating an overall image analysis, which can serve the varying input to different depth cues for a 2D to 3D conversion system. Such an image analysis must extract the different types of image information in such way, that the types can connected and also semantically related to each other. In this way, the image analysis would allow for going from sparse to dense complete-pixel results. As many depth cues deliver only sparse estimations, this represents another important step to obtain depth maps.

To establish such an image segmentation, which is at the same time fast and robust while delivering considerable quality, I investigate the principles of *Swarm Intelligence Swarm Intelligence (SI)*. Algorithms based on SI principles have been discovered to be efficient and gracefully degrading alternatives to classical approaches over a wide field of applications. Their distributed nature often greatly simplifies approaches compared to complex centralized concepts. This feature provides the computational efficiency, further it makes SI algorithms well-suited for optimization techniques, such as parallelization, GPU based programming, etc.

To obtain movement information, I investigate a way to extend the matching between consecutive frames from feature point-based to complete edges and segments borders of arbitrary shape. This allows for a denser movement information extraction and easier coupling to the segments extracted in the image analysis step. The additional goal is to create an algorithm, which performs this tasks both fast and robust.

Generally many approaches in literature seek to create depth maps only, omitting the create of the additional views as extra standalone-tasks. Also in this thesis I focus on the depth cues and their input. An example for a light-weight 2D to 3D conversion concept is given in the end, which obtains high computational speed via adaptive weighting of depth cues.

1.4. Contributions

In the course of this thesis, I created novel and promising concepts and solutions to the major tasks of 2D to 3D conversion. These are mentioned below.

1. I developed a unifying image analysis concept, which delivers region-based, edge-based and point-based feature-representations of an image. Utilizing the principles of Swarm Intelligence, this concept efficiently delivers the main input needed for the 2D to 3D conversion step, yet facilitates a variety of other usage.

   To the best of my knowledge, the SI-based image segmentation algorithm introduced in this work is in this form a completely new realization of the SI-principles. It bases mainly on the biological phenomenon of self-assembly. As part of its description, I also relate its concepts to the various SI-principles.

2. Additionally, I present a novel matching concept and a matching strategy, which is specifically tailored to the information extracted by the image analysis tool mentioned above, and allows for finding dense correspondences in consecutive images at low computational time. While being applied to the image analysis and matching concepts introduced here are thus appropriate for a variety of typical tasks in image processing and computer vision, like 2D-Matching and Object Tracking, etc.

3. Furthermore, I present an efficient automatic 2D to 3D conversion approach for the fusion of various depth cues to a reliable depth map. The approach delivers depth maps via an adaptive weighting scheme of the single cue outputs that is based on confidence measures. It exceeds the quality of real-time conversion systems in terms of correctness on a variability of input scenes and quality of details, while maintaining a considerably low computational complexity.

1.5. Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2, describes the general idea and principles of *Swarm Intelligence*, along with some prominent examples of algorithms from this field.

In Chapter 3 I introduce the Swarm Intelligence based algorithm for image segmentation, analysis and feature extraction, which I call *SISeg*.

In Chapter 4 I derive the *IBRIEF* algorithm for describing and matching arbitrary edges, which adapts the *BRIEF* point feature descriptor.

After that, in Chapter 5 I showcase an adaptive scene analysis and depth-cue fusion method as a solution to the task of 2D to 3D conversion, which utilizes the outcomes of the image sequence analysis introduced in the previous chapters.

Chapter 6 evaluates the quality and the computational performance of the algorithms introduced in the chapters 3, 4 and 5, including a comparison to the competing algorithms in its field.

Chapter 7 gives a short summary and concludes this thesis.

Chapter 2.

Background on Swarm Intelligence

In artificial systems, the term *Swarm Intelligence* describes a special variant in the field of artificial intelligence. It can be regarded as a special variant of multi-agent systems [13].

The main inspiration of artificial Swarm Intelligence is the nature itself, like bird flocks, fish school, different types of social insects, like ants, bees, wasps, and termites, or even bacteria. For centuries, scientist have been fascinated by the complex structures of bee and wasps hives or termite nests, as well as the cooperative movement of animals and the coactive foraging of e.g. ants. Scientific observations implied that, while achieving highly organized patterns, these swarm phenomena are not led by a centralized control. Instead, simple agents interact locally with their neighbors, all following a set of rules. The basic underlying assumption of cooperative rule-based behavior is that a single agent observes its nearby environment, reacts on stimuli of the environment according to a simple “plan” which it follows, and thereby changes its environment to a certain degree. This change again induces reactions in other agents nearby - interactions -, which again leads to further environment changes.

A proper design of such interaction schemes can lead to complex problem solving. This phenomenon is called *emergence*. While, as stated above, social animals are the main inspiration of Swarm Intelligence, one can observe swarm behavior even among groups of humans, e.g. when they self-organize their movement into paths in crowded places.

One of the main advantages of this approach, i.e. of modeling complex processes by distributed locally interacting units can significantly simplify these processes of any kind, compared to a centralized control. In his popular science book [37], Len Fisher gives an illustrative example of this statement, using the following fictional scenario. Imagine an alien would visit earth for the first time and hover over a big football stadium with its spaceship. Further imagine, that right in this moment, the audience in the stadium would conduct the ”mexi-

can wave". To an alien watching from above, the stadium might appear like a huge single creature, that coordinatedly moves its 70000 tentacles, which would constitute a very complex procedure for a single centralized control instance. In contrast, the 70000 autonomous agents - the audience - mainly must follow one simple basic rule:

> If the person on your left stands up, rises his arms and sits down again, stand up, raise your arms and sit down again.

This shows, how local rules and interaction can simplify, what would be considered a very complex process from outside.

The first appearance of the term "Swarm Intelligence" in a technical context, was by Beni and Wang [9], who applied SI in context of cellular robotic systems. They utilized several simple agents, self-organized by nearest-neighbor interactions, to build patterns in a one-, two, or three-dimensional environment. Since then researchers utilized the principles on various applications, e.g. motion-planning and cooperate task-solving of multi-robot systems [95], and used them for for optimization, and data analysis problems [130].

The distributed cooperative design of SI algorithms leads to implementations, which are robust and adaptive to changes of the problem space.

In the following section, I present the theoretical background of Swarm Intelligence by illustrating its major basic principles along with examples of their respective verification in animal societies.

Afterwards, I give a rough overview on most popular existing Swarm Intelligence algorithms and then review various applications of these algorithms, with a special focus on image processing- and computer vision-related topics.

2.1. Principles of Swarm Intelligence

In this section an overview over the most prominent characteristics of Swarm Intelligence is given. As Swarm Intelligence is directly inspired by biological phenomena, I will describe each of the principles following below along with examples occurring in nature, i.e. in social animals. For the sake of legibility, I will not specify the exact names of the respective insect species, in which the phenomena described below occur. Instead I will give suggestions of further reading material, in which also the biological names can be found.

2.1.1. Self Organization

The term *Self Organization* originally descends from chemistry and physics and was used to describe the emergence of macroscopic patterns from microscopic processes. In terms of social insects or animals, Self-Organization constitutes “a set of dynamical mechanisms whereby structures appear at global level of a system from interactions among its lower-level components.” [13] and is thus the most essential principle. To achieve a self-organizing emergent system, four basic notions are

- Positive feedback, which forces the establishment of structures, when nearby agents amplify the best solution of a neighbor. It is often observed in recruitment and reinforcement. An illustrative example in nature is the foraging, i.e. the exploration of food sources and the collective transport of the food to the ants' nest. When an ant lies a pheromone trail to the position of some prey or food source, which attracts further ants, which again drop pheromones.
- Negative feedback, which counterbalances the positive feedback. This often comes in form of a certain saturation or exhaustion and helps to establish stable structures after a exploration phase. Continuing the example above, if a the food source decreases over time and is depleted, less agents will be recruited, making way for discovering new sources eventually.
- Fluctuations, which introduce some amount randomness into the behavior of agents. In the ant example, this would at first glance mean, that an ant gets lost on the way to the food source. However, this helps to properly explore the search space, as a lost ant can happen to explore another food source. In terms of optimization algorithms, randomness is one of the key mechanisms to overcome local minima.
- Multiple interactions between the agents, which all examples of Self Organization inhere. While food in the foraging example might theoretically also be gathered by a single ant, the process of iteratively forcing each other mutually to transport the food makes the foraging effective and fast.

2.1.2. Communication and Stigmergy

As mentioned above, interactions between agents are a key feature in Self Organization, or respectively in Swarm Intelligence. There are two basic types of interactions: *direct* and *indirect*.

In nature, *direct interactions* between agents happen via e.g. visual, sonic, chemical, by mandibular contact, trophallaxis, or antennation. Contrary to that, *indirect interaction* happens via the environment, i.e. when one agent changes it, and the next agent reacts to the change. This mechanism is called *Stigmergy*. In terms of the foraging example from the section above, the ants communicate indirectly via the environment by dropping pheromone trails to describe the way between the food source and the nest. Other ants can "read" the pheromone information and follow the trail.

2.1.3. Self-Assembly

Besides the already mentioned principles, which can explain phenomena of collective activities, like nest-construction, cooperative foraging, and collective decision making, *Self-Assembly* denotes a further special behavior in social insect societies. In some situations, insects build physical structures by grouping together and linking to each other.

These structures, called "self-assemblages", are stated to be "intermediate", as they do neither occur on the level of a colony, nor on that of a single individual, but in between, i.e. in subgroups of insects [3]. It follows, that within a colony or swarm, several self-assemblages can occur simultaneously, and not necessarily all individuals of a colony must participate within.

Insects like ants, bees and wasps connect themselves in structures for various purposes. For example, ants can build bridges and ladders to overcome gaps, or plugs to block entries to their nests in defense situations. Other species build living walls of defense. For survival purposes, when a nest is flooded by heavy rain, insects link together forming a living raft used to carry their brood and their queen.

As an example of a utilization of self-assemblies in technical applications, [126] published an implementation of swarm-robots, which cooperatively build three-dimensional structures.

2.2. Swarm Intelligence Algorithms

After presenting the general principles along with examples from nature, this section shall give a insight on how these mechanisms can be successfully applied in algorithms and applications in the field of Computer Science. Therefore, I will introduce some of the most well-known state-of-the-art swarm-intelligence algorithms, along with remarkable areas, where

they have been applied, like optimization, communication networks, simulations, cooperative robotics.

The common ground of SI algorithms is, that they always follow the principles introduced above to some degree i.e. being decentralized and interacting autonomous agents guided by simple local rules. The big difference between them lies in the realization of these principles. This again is closely connected to the biological phenomenon they are based on. Yet, it has to be mentioned here, that not all SI algorithms are modeled after a distinct natural prototype. The following listing gives an overview on the most known optimization SI algorithms, with a short view on the different realizations.

While the first of the example algorithms represents Swarm Behavior rather than Swarm Intelligence and descends from the field of multi-agent-modeling or multi-agent-simulation (e.g. for computer graphics), the rest of the given examples are optimization algorithms.

- The *BOIDS* algorithm [94] realistically simulates the movement of a bird swarm, where each virtual bird follows three simple movement rules in each step, that align each agent's position and velocity to the swarm. The swarm behavior emerges naturally from this, without centralized control. Methods of this kind were applied to crowd simulation experiments, computer games and movies, like Batman Returns or the Lord of the Rings-Trilogy to create realistic mass scenes.
- In the *Particle Swarm Optimization (PSO)* [54] particles move through a search space, and check and compare their candidate solutions, which in turn influences their movement. They are thus guided by solutions of others (broadcast-type communication) and their own best solution (memory). With no explicit function derivation of the search space needed, PSO has proven to be an efficient solution to a variety of problems.
- The *Ant Colony Optimization (ACO)*, introduced by Marco Dorigo in 1992, [32] simulate ants finding the optimal way to a food, or likewise the optimal distances in a graph, utilizing a stigmergic communication mechanism via a pheromone evaporation model. These algorithms thus directly rely on the foraging mechanisms of real ants, which were described in section 2.1.1.
- The *Stochastic Diffusion Search (SDS)* [12] represents an example of following the SI principles without having a concrete archetype, just like PSO. It uses direct one-to-one direct communication instead stigmergy. Agents create and evaluate hypotheses in simple partial solutions and share information on hypotheses (diffusion). Complex overall solutions then emerge from clusters of agents.

- The *Bacterial Foraging Algorithm (BFO)* [86] follows mechanisms of e-choli bacteria, movement mixes randomness and local hill climbing in order to effectively explore a search space. Also the bacterias' reproduction phases are simulated, randomly re-initializing agents with low fitness values to reach a more exploratory search.
- *Bees Algorithm* The Bees Algorithm [88] is a search algorithm inspired by bees, where the swarm's movement is guided by scout bees, communication mimics the bees' waggle dance.

This listing is only an excerpt and by no means complete. There are further interesting implementations and methods to be found in the literature, like e.g. the *Glowworm Swarm Optimization*, the *Intelligent Waterdrops Algorithm*, the *Swarm Contours*, *Firefly*, the *Swarm Organized Projection*, and many more.

Yet, it is to mention, that the huge variety of "metaphor-inspired" approaches based on metaheuristics in recent years also caused criticism in the research community [109], claiming that the novelty and intention of some approaches can overshadow a lack of efficiency. Nevertheless, there is a subset of approaches, like PSO and ACO, which proved high efficiency in a number of problems.

2.3. Related work on SI Algorithms in IP and CV Applications

In recent years, algorithms descending from the field of Evolutionary Computing, Genetic Algorithms, and also Swarm Intelligence are increasingly applied to a variety of tasks in the fields of Image Processing IP and Computer Vision CV.

When utilized in image segmentation, these optimization tools have mostly been used to improve the performance of important steps of segmentation algorithms. In [42, 28], evolutionary and genetic algorithms were used to enhance the clustering process in segmentation. In [61], the authors combined high-level features generated with a visual attention model with low-level features to guide region growing algorithm, where the optimal thresholds of the region growing process were detected using the Particle Swarm Optimization algorithm.

In [65], the PSO was applied to find the optimal fuzzy entropy thresholds to segment images into foreground and background. A related approach was introduced in [36], where PSO was used to tune thresholds in 2D-histograms, maximizing the entropy to segment infrared images.

SI was also utilized to improve Segmentation based on clustering approaches, which can get stuck in local optima, depending on their initialization. In [96], the authors developed a hybrid combination of the Ant Colony Optimization algorithm (ACO) and PSO to make K-Means clustering and Single Competitive Learning more independent from their initial cluster centers, and learning rate, respectively.

Besides deploying SI techniques to optimize specific steps in existing segmentation approaches, SI algorithms were also applied to the tasks of edge detection and contour detection, akin to the proposed algorithm. The works in [57] and [58], which cope with contour and straight edge detection, can actually be considered preliminary work of the SI algorithm proposed in this thesis.

The authors of [34] proposed ant-based correlation for edge detection. Their method is capable of performing feature extraction for edge detection and segmentation, generating less distortion in the presence of noise, as compared to classical edge detectors, like Sobel, Prewitt, and Canny.

In [105], a variant of the PSO algorithm is utilized for finding a proper edge detection filter size in noisy images. While combining edge filters of different sizes can improve the edge detection in noisy images, it is generally computationally expensive when applied to the entire image. Thus, the heuristic capabilities of PSO were used to adaptively decide, where on the image to use filters of various sizes. The evaluation of edge intensity and the proposed movement principle in this approach have analogies to the algorithm presented in this thesis, as they divide the edge regions into sets and use averaged intensities. However, the major difference is the usage of discrete PSO instead of the communication scheme used in this theses, where the agents share their local information.

Besides image processing problems e.g. like edge detection [7], SI was also successfully applied to other Computer Vision tasks, like feature extraction and object recognition [78, 77, 2, 59], and 3D tracking and sensor data fusion [56, 55].

Chapter 3.

Swarm Intelligence-Inspired Image Analysis

This chapter is dedicated to first crucial step for the 2D to 3D conversion procedure, which constitutes a tool for image analysis.

As stated in the introduction, all available depth cues prerequisite pre-analyzed various image input, like e.g. color information of regions, the position and appearance of straight lines, etc. One of the pivotal keys to depth map generation is the assumption, that image boundaries and object boundaries most often coincide.

Hence the need of an image analysis, that is based on image segmentation - we want to segment image content in order to be able to assign each segment, region, object a certain depth, to achieve a dense depth map. Additionally, we want to be able to get all kinds of image features, area based, edge based, or point-based, out of each image.

This chapter is organized as follows. After a review of the state of the art in image segmentation, I introduce a Swarm Intelligence based boundary detection and image segmentation framework, which I call *SISeg*. The successive section then details, how the boundary detection is expanded to an image analysis tool to describe the shape and appearance of regions, contours, lines, and points, coevally. This chapter is completed with a discussion on how the SISeg-algorithm relates to the Swarm Intelligence principles and how the final concept evolved.

3.1. Related Work on Image Segmentation

Image Segmentation is a central problem in computer vision that has been studied for years. It constitutes an important initial step for high-level tasks in computer vision, like object recognition, image analysis, and scene understanding. It pursues the goal of dividing an

image into parts, regions, or objects, which are preferably self-consistent and of meaningful content. Application examples include separating objects considered as foreground from the background, or separating multiple objects from each other.

However, one of the main difficulties in evaluating boundary detection and image segmentation arises from the fact, that not even human beings are able to uniquely and unambiguously decide and agree over a correct segmentation [72]. Different persons are likely to have different opinions on the correct solution to the segmentation problem of any image, especially involving natural images. This lack of an unique ground-truth complicates the comparison of results descending form different segmentation methods. Nevertheless, efforts were taken to establish a framework for objective evaluation of segmentation algorithms. A prominent example is the Berkeley Segmentation Dataset (BSDS) [72], which provides a huge collection of natural images along with human-marked proposals of ground-truth, as well as a precision-recall based measurement for comparison [73]. Figure 3.1 shows three example images along with three different human ground-truth suggestions.

Figure 3.1.: Three sample images from the Berkeley Image Segmentation Dataset (BSDS 500) [6], each along with three different human-marked ground-truth images.

The plenty of solutions to the segmentation problem proposed in the literature descend from versatile fields, such as clustering [113], graph theory [35, 25], region growing [27], optimization of an evaluation function like energy minimization [20], as well as image feature-based methods [24, 21]. The majority of approaches found in the literature can be categorized into

one or a subset of these fields, with each adding improved mechanisms to solve specific tasks.

Many segmentation approaches focus on boundary or contour detection as an initial step, as boundaries naturally correspond to borders between objects. Identifying and locating sharp discontinuities in image regions closely relates this to the task of edge detection. Mere edge detection, however, registers on the one hand any type of abrupt changes in image brightness, and misses smooth or subtle transitions of image brightness on the other hand. Regarding the goal of detecting boundaries, this causes false positives like textured regions, or single highlighted spots and true negatives like region boundaries of low contrast.

One recipe to overcome these drawbacks is to combine edge detection techniques with diverse soft computing approaches like Genetic Algorithms, or Fuzzy Logic, etc. [104, 93]. Other methods combine oriented local image cues, like brightness, color, and texture descriptors with edge detection to get a global estimation. Such combination methods include multi-scale cue combination [6], energy and probability weighting [69], or likelihood-methods [120].

In recent years, methods including learning steps have taken the lead, and provide the best results, when human segmentation displays the ground truth and a training dataset is available.

In [66] the authors introduce a contour detection approach with contour patterns for local image patches, that are clusters of hand-drawn sketches. Their input feature vector consists of CIE-LUV color information along with oriented gradient information plus a self-similarity measure computed over the pixels within the patch to tackle texture. For classification a random forest approach is used, which turns out to represent an efficient solution to this problem. The authors report a considerably efficient computational speed, using about 1 second per image. Most of the computation time is spent on the generation of the input feature vector.

Similar to this, in [31] local image patches are learned to extract edge information by mapping all patches to a set of patterns, like lines, T-junctions, or Y-junctions. They also utilize Structured Forests for classification combined with structured prediction. While delivering high quality results, their approach makes the edge classification extremely fast compared to the majority of other learning based edge detection or segmentation approaches. However, they make use of parallelization on four cores and optimization, to tackle the complex input feature vector efficiently.

The authors of [127] use sparse coding and oriented gradients to learn dictionaries of contour patches, achieving excellent results at high computational costs.

In [51], the authors extract the point-wise mutual information from learned affinity functions to decide, if pixels are within the same or different segments. The approach works extremely well, yet the extraction of the affine functions is computationally expensive.

3.2. SI inspired Image Segmentation

In the subsequent sections, I will introduce a procedure for image analysis, which I developed in course of this thesis. It consists of a computationally efficient, low-cost image segmentation algorithm called *SISeg*, which allows the extraction of regions by detecting and cutting region borders of considerably large scales.

Additionally, the procedure provides the extraction of salient lines, as well as feature points, along with the regions, and thus allows for establishing an appearance-based scene description.

As the image segmentation is conducted by a set of agents, which utilize the principles of Swarm Intelligence, following a scheme that draws inspiration from “Self-Assembly”, the acronym *SISeg* stands for “Swarm Intelligence based Segmentation”. The relation of the mechanisms realized in the *SISeg* concept to the principles of SI and Self-Assembly are discussed in the end of this chapter.

Generally speaking, a set of agents, which utilize a local view on the image and on their neighbored agents, is distributed over the image under analysis. They examine the image differences of their position to their neighbors’ positions to establish an estimate, or respectively a model for a boundary in their respective regions. In an agent’s view, such a local boundary model consists of the *boundary direction*, i.e. where it cuts through a segments, and the *boundary connection*, i.e. the course, the boundary takes in the agent’s neighborhood. The interaction with the local neighborhood, the mutual reacting and adapting to models in the local surrounding causes the establishment of global boundaries. Likewise, any segment as well as any boundary consists of a set of agents, which are distributed over the image space and connected to each other. Fig. 3.2 shows an example of a progression of a boundary between two regions established by the agents.

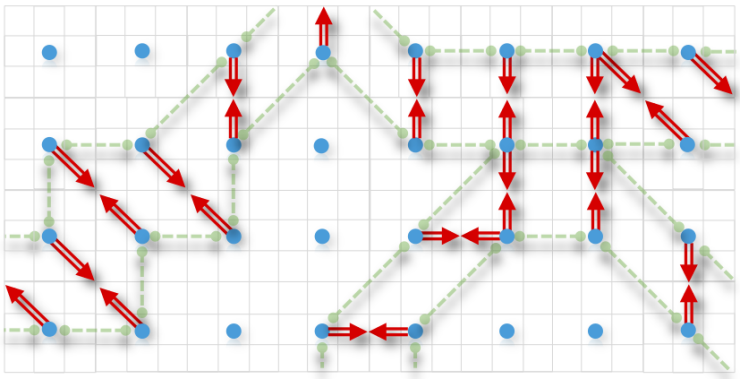


Figure 3.2.: Example of SISeg-agents (blue) establishing a boundary chain on a pixel grid, with boundary directions (red) and symmetric connection directions (green).

The following sections depict the SI-based approach of image segmentation in detail, followed by a discussion on the realization of the SI-principles.

3.2.1. General Agent Configurations

This section describes the general setting of agents in this algorithm, their positioning, and the local view they have on the image environment and to other agents. Further, the local geometry they utilize and the appearance based measures they extract from the environment is explained. These principles are a prerequisite and the basics to the procedure of the algorithm, which will be explained in the consecutive section.

Positioning and Environment View

To separate regions in an image from each other, *SISeg* utilizes a set of $\mathcal{N}$ agents, which are equidistantly distributed over the input image $\mathcal{I}$ in a fixed distance of 3 pixels in each direction. Therefore $\mathcal{N} = \lceil h/3 \rceil \times \lceil w/3 \rceil$, with h, w being the image height and width.

This means, that the agents form a grid in which each agent occupies a window $\mathcal{W}_n^{px}$ of 3×3 pixels centered at its position, with $n = 1 \ldots \mathcal{N}$ serving as the agents' IDs. In order to make sure, that all agents possess a window of equal size, an image is enlarged by either one or two columns, or respectively rows on its right and bottom side if necessary. After the algorithm execution, all results are transformed back to the original image size by omitting, or respectively cutting the image columns and rows that were added before.

The window $\mathcal{W}_n^{px}$ represents both the agent's directly accessible input as well as the image space, an agent is "responsible" for. This means, that the agent decides for each pixel inside the window, to which region it belongs, and if a boundary intersects the window, which pixel-wise course it takes. For the distance metric introduced below, each agent also decides, which pixels within the window are it considers as inside or outside its region. This means, an agent sets a binary mask within $\mathcal{W}_n^{px}$, as will be explained in detail in subection 3.2.2

As stated in the introduction of this section, the algorithm establishes boundaries by local interactions between the agents. Therefore, each agent possesses a communication window $\mathcal{W}_n^{co}$, allowing each agent to collect information from its surrounding eight neighbors, thus forming a *Moore neighborhood*, as shown in the middle of Fig. 3.3.

Within $\mathcal{W}_n^{co}$, an agent can access indirect information about their environment and also see the current models of its eight neighbors, which will - in combination with its own model and environmental state - influence its own actions. Such an indirect way of communication is called *stigmergic* (described in subsection 2.1.2). Thus, an agent can view, or *sense* the environment within the $\mathcal{W}_n^{co}$ window, but it can change, or *act on* the environment only within its $\mathcal{W}_n^{px}$ window.

Local Geometry and Communication

To interpret the local environment, the agents need a local coordinate system and a number geometrical operators to describe their *local geometry*, which shall be introduced in the subsequent paragraphs. Directly resulting from the *Moore neighborhood* the agents' basic geometry consists of a set of $\mathbb{D} = \{1, 2, \ldots, 8\}$ directions. In this chapter the variable $d \in \mathbb{D}$ will denote a direction $1 \ldots 8$. As shown in Fig. 3.3, $d = 1$ is defined to point into negative x direction and the numbers increase in clockwise direction. As stated above, an agent n can access all model information of its surrounding eight neighbors, which is notated by the subscript d added to the respective model information. In this case, d represents the 'relative' agent ID, i.e. the neighbor's ID relative to agent n.

In addition to the absolute direction d, the agents need to describe relative directions, which depend on an initial direction, as they interpret their environment depending on the direction, in which they hold a boundary. Following the definition of d, the *clockwise* relative direction is denoted as '+' and the *counter-clockwise* direction as '-'. Going x steps from a direction d into a relative direction $\pm$ is thus notated as $d^{\pm x}$, so if e.g. $d = 1$, then $d^{+3} = 4$, and $d^{-2} = 7$.

The relative direction *opposing d* is notated as o, such that $d^o = d^{\pm 4}$, so for example, if $d = 2$, then $d^o = 6$.

To sum up the notation principles of the agents' geometry, a relative direction is marked as superscript on an absolute direction, while any direction as subscript of a variable allows an agent n to access the variable of its neighbor of in that corresponding direction.

In order to analyze its local environment, each agent calculates the distances, or respectively differences of its governed image patch $\mathcal{W}_n^{px}$ to the patches of its eight neighbors, which is denoted $\mathcal{D}_n(d)$. There are two important aspects concerning the image input, which are tightly connected to each other. The first one is the choice of the *color space*, and the second is the choice of a proper metric or distance measure $Dist(agent, neighbor)$ in order to describe the relations of image values. In case of a standard *Euclidean distance measure* on a *RGB-color image*, the distance denotes to

$$\mathcal{D}_n(d) = Dist(\mathcal{W}_n^{px}, \mathcal{W}_d^{px}) = \frac{\sqrt{\sum_{ch}(\mu(\mathcal{I}_{ch}(\mathcal{W}_n^{px})) - \mu(\mathcal{I}_{ch}(\mathcal{W}_d^{px})))^2}}{|ch|}, \tag{3.1}$$

with $\mathcal{I}_{ch}$ being the color values of channel $ch = \mathrm{R}, \mathrm{G}, \mathrm{B}$, and thus $\mu(\mathcal{I}_{ch}(\mathcal{W}_n^{px}))$ being the mean pixel values of an agent n's input window $\mathcal{W}_n^{px}$, and $|ch|$ the number of image color channels.

The general concept of SISeg does not limit the usage of spaces and distance metrics as input for $\mathcal{D}_n(d)$ in any way. However, during the remains of the description of the concept, $\mathcal{D}_n(d)$ denotes a distance measure between agents and their eight neighbors, with a high value indicating a potential segment boundary, and a low value imposing the likeliness of a connection between the agents, or respectively, the absence of a boundary.

As an additional metric, $\mathcal{D}_n^o(d)$ denotes the difference or distance between the agents' window values of two opposing neighbors of an agent, ie. $Dist(\mathcal{W}_d^{px}, \mathcal{W}_{d^o}^{px})$, as shown in Fig. 3.3b.

Via the eight neighbors, the communication window $\mathcal{W}_n^{co}$ thus enables an agent to indirectly view its surroundings within a 9×9 pixel field, which together with the 3×3 input patch allows for both accurate localization as well as robustness against local image disturbances caused by aberrations or noise.

The fixed-grid *Moore neighborhood* further facilitates the agents to cooperatively model ar-

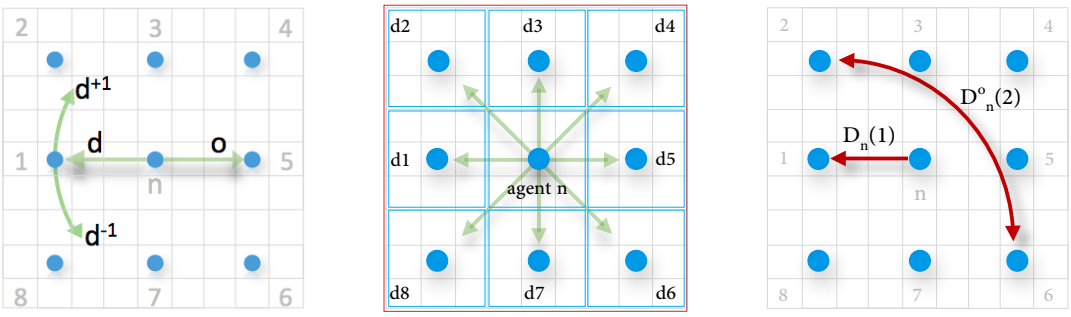


Figure 3.3.: Left figure shows the definition of relative directions in clockwise, counter-clockwise, and opposing directions for $d = 1$. The middle figure shows *Moore neighborhood* $d = 1 \ldots 8$, the input window $\mathcal{W}_n^{px}$ (blue) and the communication window $\mathcal{W}_n^{co}$ (red) of an agent n. The right figure shows, between which agents the measures $\mathcal{D}_n(d)$ and $\mathcal{D}_n^o(d)$ are calculated.

bitrary boundary structures while maintaining manageable complexity regarding the variety of basic models and the rule-set, that defines the agents' behavior.

Memory: Agent Environment and Boundary Model

Besides communication, a further typical feature of SI algorithms is the *memory* of each agent. In the *SISeg* algorithm, the memory of an agent n consists of two models, *Agent Environment Model* $\mathbb{M}_n^E$ and the *Agent Boundary Model* $\mathbb{M}_n^B$. The two memory models are updated in each iteration and influence all decisions, each agent takes.

Every agent keeps all of the input it extracts from its environment in its *Agent Environment Model* $\mathbb{M}_n^E$, which consists of the agent's ID, distance metrics, and window information described above, plus the the *line condition* $\mathbb{L}_n$ defined in 3, i.e. $\mathbb{M}_n^E = \{n, \mathcal{W}_n^{px}, \mathcal{W}_n^{co}, \mathcal{D}_n, \mathcal{D}_n^o, \mathbb{L}_n\}$.

The result of the *SISeg*-algorithm, from the perspective of an agent, will be its *Agent Boundary Model* $\mathbb{M}_n^B$, which describes its estimate on the course and strength of the boundary segment the agent governs in its image patch.

During the procedure, each agent establishes its boundary model, which formally consists of $\mathbb{M}_n^B = \{\mathbb{B}_n, \mathbb{B}_n^{ack}, \mathbb{C}_n, \mathbb{B}_n^{pot}\}$, and comprises of sets introduced in the following.

The set $\mathbb{B}_n$ holds all *boundary estimation directions*, i.e., the directions d, in which an agent estimates a boundary between itself and its neighbor in that direction. The set $\mathbb{B}_n^{ack}$ holds all *acknowledged boundary directions*, i.e. those directions d, which are boundary estimates of an agent and also of the neighbor in this direction. $\mathbb{B}_n^{ack}$ is thus a subset of $\mathbb{B}_n$.

The set $\mathbb{C}_n$ stores *boundary connections* to neighbors. For any boundary direction guess in $\mathbb{B}_n$, an agent searches in clockwise and in counter-clockwise direction for the neighbors that are connect to for evolving the boundary. Thus the set $\mathbb{C}_n$ holds a set of relative IDs.To some degree, this set holds redundant information, which could also be extracted from the acknowledged boundary set. Yet, storing it explicitly is more efficient in terms of computational speed, as the extraction would need a geometrical analysis in each iteration, for each neighbor.

Agents that are not part of a boundary thus have an empty set $\mathbb{C}_n = \emptyset$, agents within regular boundaries (around large regions) possess two connected neighbors, and agents in special scenarios, like line-alike boundaries, or can have one to four connections. As the set $\mathbb{C}_n$ stores only connections that share a boundary, all neighbors to an agent, which are not within its $\mathbb{C}_n$ set or within $\mathbb{B}_n^{ack}$ set, lie in the same region and are thus connected.

Having defined the agents view, geometry, and models, I now want to explain the procedural steps, that each agent takes to decide on its model and to successively obtain the final segmentation result. The geometrical operators defined above allow an agent to examine the $\mathbb{B}_n, \mathbb{C}_n$ sets of itself, as well of its neighbors, via a set of behavioral rules, which will be defined in the different phases, each agent is going through.

3.2.2. Initialization-Phase

The algorithm starts with the *initialization phase*, in which the agents first utilize the initial distances to their neighbors to estimate an *initial boundary direction*. This phase sets the initial seed for all boundaries, and at the same time offers a mechanism to suppress local suboptimal boundary guesses.

Initial Boundary Direction Guess

In order to make the initial boundary guess more robust, the agents do not simply choose the direction of maximum distance. Instead, each agent searches its initial boundary direction based on a *weighted sum of distances*. For all directions, the direct distance is weighted with the two distances of the directly neighbored distances. The sum is thus calculated by

$$\mathcal{D}_n^s(d) = \frac{1}{w_1^s + 2 \cdot w_2^s} \times (w_1^s \times \mathcal{D}_n(d) + w_2^s \times \mathcal{D}_n(d^{-1}) + w_2^s \times \mathcal{D}_n(d^{+1})), \tag{3.2}$$

with w_1^s, w_2^s being arbitrary weighting factors. Empirical evaluation indicates, that a general weighting fulfilling $w_1^s > w_2^s$, i.e. the direct direction, provides both robustness and a proper localization.

Each agent searches its *initial boundary direction* $\mathbb{B}_{n,0}$, which is marked by the subscript index 0. The initial boundary direction is chosen to be the direction d with the highest summed distance $\mathcal{D}_n^{s,max}(d)$ to the respective neighbor, thus

$$\mathcal{D}_n^{s,max}(d) = \max_{d \in \mathbb{D}} \mathcal{D}_n^s(d) \tag{3.3}$$

and

$$\mathbb{B}_{n,0} = \arg\max_{d \in \mathbb{D}} \mathcal{D}_n^s(d). \tag{3.4}$$

Non-maximum suppression (NMS)

Equation 3.4 applies to all agents on an image. However, to reduce the effects of oversegmentation, some boundary segments describing boundaries of low saliency should be suppressed. To achieve this, it is straight forward to examine the agents' maximum summed distance values $\mathcal{D}_n^{s,max}$.

For natural images of arbitrary complexity, it is not easy to find one single absolute threshold for suppressing the boundary building, even if setting this value would be done adaptive. Analyzing natural images, one can instead see, that there is a certain minimum distance value, below which boundaries do not make sense, as well as a maximum distance value, above which a boundary should definitely be accepted. But there is also a considerable gap between those values, in which a thresholding must be realized, which takes the neighborhood into account. An agent should possess a maximum distance, which is considerably higher than its neighbors - yet, as the agents search elongated boundaries, not single maximum points, the threshold must not be set too high. Comparing an agent's detected maximum distance to maximum distances of its neighbors allows adapt to the local image content. In a highly textured image area, an also agent needs higher distance values than in smooth regions.

Thus, I apply a scheme, which divides the range of distances into three subareas, establishing a hysteresis-like threshold function. Agents with distance values below τ^{min} are suppressed, while agents with values higher than τ^{max} always pass. For the range in between, the threshold τ^{rel} constitutes the factor, by which an agents maximum distance must be bigger than the *mean* of all maximum distances of its eight neighbors. Thus, the agent's value

$\mathbb{B}^{pot}$ describes, if an agent is allowed to set a potential boundary, it's condition is calculated by

$$\mathbb{B}_n^{pot} = \mathcal{D}_n^{s,max} \geq \tau^{max} \vee (\mathcal{D}_n^{s,max} \geq \tau^{min} \wedge \frac{\mathcal{D}_n^{s,max}}{\mu(\mathcal{D}_{\mathbb{D}}^{s,max})} \geq \tau^{rel}), \tag{3.5}$$

with $\mu(\mathcal{D}_{\mathbb{D}}^{s,max})$ being the mean of the maximum distances of all neighbors in $\mathbb{D}$.

Experiments showed, that values between 1.1 ... 1.5 for τ^{rel} deliver reasonable results.

Then equation 3.4 adapts to

$$\mathbb{B}_n = \underset{d \in \mathbb{D} \wedge \mathbb{B}_n^{pot}}{\arg\max}\ \mathcal{D}_n^s(d). \tag{3.6}$$

Figure 3.4 shows an example for acknowledged boundaries of agents in the end of the initialization phase, when using (from (a) to (d)) no NMS, only absolute NMS with a considerably high (green) and a low (red) threshold, only the relative part of the condition in 3.5, and the complete condition of 3.5.

Once initialized, all agents, that form a single boundary segment, will in the consecutive phase try to connect to greater segments, inducing growth of boundaries, if necessary. From this, it is plain to see in this example, that omitting NMS will lead to over-segmentation.

Acknowledge Boundaries

A key step to establish boundaries in this algorithm is the interaction between the agents. The root of this is, that any agent wants any of its boundary estimations acknowledged. This means, that the neighbor lying in the boundary direction of an agent needs to have a boundary direction estimation pointing either towards or 'nearby' the agent, with 'nearby' meaning towards one of the direct neighbors. This condition is mathematically given by the rule

Ruleset 1. *Acknowledge boundaries update*

$$\begin{aligned} &\forall\, d \in \mathbb{D} : \\ &\{d \in \mathbb{B}_n^{ack} \mid d \in \mathbb{B}_n \wedge (d^o \in \mathbb{B}_d \vee d^{o+1} \in \mathbb{B}_d \vee d^{o-1} \in \mathbb{B}_d)\} \end{aligned} \tag{R 1.1}$$

where $\mathbb{B}_n^{ack}$ is the subset of $\mathbb{B}_n$ holding all acknowledged boundary directions.

In order to synchronize their views, agents then check if they have caused acknowledged

(a) Agents initialized without NMS

(b) Absolute threshold based NMS, with τ^{min} = 6 (red), and τ^{min} = 20 (green)

(c) Relative threshold based NMS, with τ^{rel} = 1.1

(d) Relative and absolute thresholds combined, τ^{rel} = 1.1, τ^{min} = 6, and τ^{max} = 20

Figure 3.4.: Sub-figures (a) to (d) show the all initialized agents holding acknowledged boundaries after applying various approaches for non-maxmimum suppression (NMS).

boundaries at neighbors via this 'nearby' scenario, and if so, they adapt to them by adding the directions to those neighbors in their boundary list. This is expressed by the rule

Ruleset 2. *Adapt to nearby boundaries*

$$\forall\, d \in \mathbb{D}:$$
$$\{d \in \mathbb{B}_n^{ack} \mid d^o \in \mathbb{B}_d \wedge (d^{+1} \in \mathbb{B}_n \vee d^{-1} \in \mathbb{B}_n)\} \tag{R 2.1}$$

which means, the agents check the direct neighbors to their own boundary directions, and if these neighbors hold an acknowledged boundary towards the agent, it also sets it.

Set Line-alike Agents

The most regular boundaries belong to regions that are considerably big compared to the agents' window size. Thus, agents, which describe such boundaries usually have the boundary estimations lying mainly on one side of their window.

However, in order to make the algorithm capable of describing small, fine-grained regions, whose extent is similar to the agents' $\mathcal{W}_n^{px}$-window size, the agents must be able to establish two different boundaries on opposing sides. Such boundaries are called *line-alike*, contrary to the *regular* boundaries. The agents must be able to distinguish such line scenarios from e.g. smooth boundary transitions. To prevent agents from setting false line-alike estimations, each agent evaluates all directions in its neighborhood via a *line condition*, which compares an agent's distance to each direction $\mathcal{D}_n(d)$ to the distance in the opposing direction $\mathcal{D}_n(d^o)$, as well as to the direct distance between the two opposing neighbors $\mathcal{D}_n^o(d)$. The *line condition*, which is stored in the set $\mathbb{L}_n$, is denoted as

Ruleset 3. *Line condition*

$$\forall d \in \mathbb{D}:$$
$$\{d \in \mathbb{L}_n \mid \frac{\mathcal{D}_n^o(d)}{min(\mathcal{D}_n(d), \mathcal{D}_n(d^o))} < \tau^O \wedge \frac{\mathcal{D}_n(d^o)}{\mathcal{D}_n(d)} > \tau^o\} \tag{R 3.1}$$

with τ^O and τ^o being thresholds in a range between [0 ... 1].

As the final step of the *initialization phase*, all agents, whose initial boundary estimation $\mathbb{B}_{n,0}$ got acknowledged by Ruleset 1, set themselves as line-alike agents, if the direction opposing their initial boundary direction is part of the set $\mathbb{L}_n$, as described in

Ruleset 4. *Set line-alike agents*

$$\forall\, d \in \mathbb{D}:$$
$$\{d \in \mathbb{B}_n \mid d^o \in \mathbb{B}_n^{ack} \wedge d \in \mathbb{L}_n.\} \tag{R 4.1}$$

After adding these additional boundary estimations, the agents update the boundary acknowledgement using Ruleset 1. This is the starting point for the *Boundary Grow-Phase* described in the next section, in which all agents decide on their further neighborhood, if they are connected or not.

3.2.3. Boundary Grow-Phase

Having obtained initial estimates for the boundary directions, the *SISeg*-algorithm now executes the *boundary grow-phase*. In this phase the agents with acknowledged initial boundaries, that fulfilled the non-maximum suppression condition, decide on the exact course of the boundaries between them and their neighbors.

Therefore, they check the neighborhood next to a boundary, which are the three neighbors on each side rectangular to the boundary direction.

Basically, within this area, an agent can either find *at least one neighbor* with acknowledged boundaries, that are in a direction aligned to the agents' own, or *no neighbor* with a fitting boundary segment. In the first case, an agent can decide on the connection of its own segment to the neighbored segment, taking the best suited one in case of multiple choices, and thus step iteratively from loose segments to a globally connected boundary. In the second case agents examine, which neighbors would be most suited to their segment and thus induce boundary growth, by communicating the wish of a boundary direction to those neighbors. Those neighbors can then decide on growing such a boundary, and in the next iteration, case one will be fulfilled.

As already stated, an acknowledged boundary direction prerequisites, that an agent holds a boundary estimate in a certain direction, and the neighbored agent in this direction holds an estimate in the opposite direction, i.e. towards the agent. Thus any boundary segment is formed by a *pair* of agents. This again leads to a symmetry, which allows agents to search the continuity of their boundary in the neighborhood of only one side. Thus, agents have an active and a passive area for growing the boundaries. Their passive side is again the

active side of the neighbors in that area, or in case of growing boundaries, it lies in the responsibility of the opposing neighbor, i.e. the partner of the boundary segment *pair*.

In terms of efficiency, exploiting this symmetry leads to halving the computational efforts.

Figure 3.5 displays examples for the three neighbors in clockwise direction, which an agent checks for boundary growth or respectively boundary connection, depending on its boundary direction.

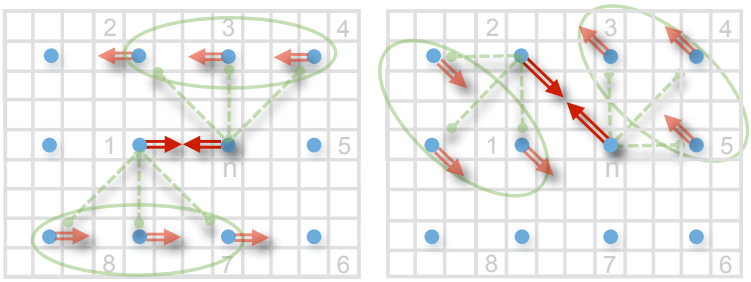


Figure 3.5.: Example of the three neighbors in clockwise direction, an agent checks for boundary growth, depending on the direction of its acknowledged boundary. The left image shows an example of a straight direction (d = 1), the right image for a diagonal direction (d = 2).

The agent's means for all the cooperate decisions stated above consist of a set of behavioral patterns, realized as condition-action rulesets. These patterns force the agents to react specifically on various scenarios they sense in their local environment. The agents' view on the local environment is composed of the distances to the neighbors, but also of the *boundary models* of the neighbors. This is what the agents sense and what - combined with their own current boundary model - determines their behavior, leading to cooperate decisions on a basis of an indirect, *stigmergic* communication. From the Swarm Intelligence perspective, the agents' local environmental view also represents their *search space* or *solution space*.

In the following, these patterns, which are conducted sequentially over a number of iterations, are described and explained in detail.

Alignment Check

As a first step to get from the single estimates to closed boundaries, agents with acknowledged boundaries examine the boundary models of the three neighbors in clockwise direction, as shown in figure 3.5 above. They therefore check if one or more of the three neighbors hold acknowledged boundaries in a direction, that is *aligned* to the agents' boundary.

For regular, i.e. not line-alike agents, *alignment* basically means, that one of the three neighbors hold an acknowledged boundary in the same or similar direction as the agent itself, which is shown in figure 3.6.

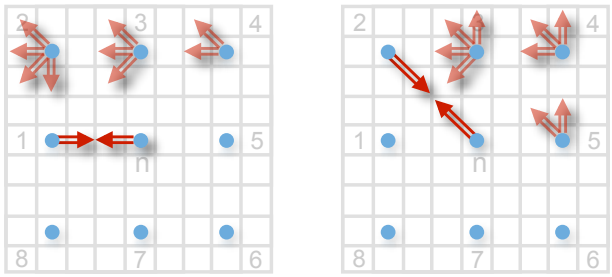


Figure 3.6.: Required boundary directions at three neighbors, in case of a not line-alike agent. At least one of the displayed direction must be an acknowledged boundary, such that the neighbor is considered to be aligned to the agent. The left image shows an example of a straight direction (d = 1), the right image for a diagonal direction (d = 2).

As described in Ruleset 4, agents can also establish a line-alike boundary model, i.e. with two boundaries in opposing directions. For any agent investigated in the alignment check, the logical operator I implies, that an agent holds a line-alike model that is *aligned* to the direction d under investigation. This alignment-condition slightly differs from the regular case, as can be seen in image 3.7.

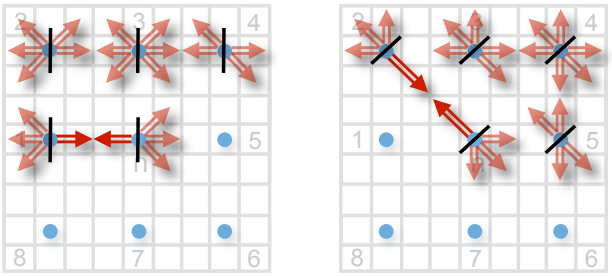


Figure 3.7.: Required boundary directions for an agent, its boundary neighbor and the three neighbors for potential connection, to be considered a line-alike agent. At least one of the displayed direction must be an acknowledged boundary on each side of the border shown in black. The left image shows an example of a straight direction (d = 1), the right image for a diagonal direction (d = 2).

Mathematically, the alignment-condition for each neighbor $d^{+1,2,3}$, denoted as $a^{d^{+1,2,3}}$, is evaluated by checking if the neighbors acknowledged boundary sets $\mathbb{B}^{ack}_{d^{+1,2,3}}$ hold boundaries in the alignment directions d, d^{-1} and so on, as described above. This is expressed by the following ruleset

Ruleset 5. *Check alignment of neighbors*

$$\forall\, d \in \mathbb{D}:$$

$$a^n = d \in \mathbb{B}^{ack}_n \wedge d^{+1} \notin \mathbb{B}^{ack}_n \tag{R 5.1}$$

$$a^{d^{+1}} = a^n \wedge (d \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{+1} \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{-1} \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{-2} \in \mathbb{B}^{ack}_{d^{+1}}) \tag{R 5.2}$$

$$a^{d^{+2}} = a^n \wedge (d \in \mathbb{B}^{ack}_{d^{+2}} \vee d^{+1} \in \mathbb{B}^{ack}_{d^{+2}} \vee d^{-1} \in \mathbb{B}^{ack}_{d^{+2}}) \tag{R 5.3}$$

$$a^{d^{+3}} = a^n \wedge (d \in \mathbb{B}^{ack}_{d^{+3}} \vee d^{+1} \in \mathbb{B}^{ack}_{d^{+3}}) \tag{R 5.4}$$

$$l^{d^n} = a^n \wedge (d^o \in \mathbb{B}^{ack}_{d^n} \vee d^{o+1} \in \mathbb{B}^{ack}_{d^n} \vee d^{o-1} \in \mathbb{B}^{ack}_{d^n}) \tag{R 5.5}$$

$$l^{d^{+1}} = (d \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{+1} \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{-1} \in \mathbb{B}^{ack}_{d^{+1}}) \wedge (d^o \in \mathbb{B}^{ack}_{d^{+1}} \vee d^{o+1} \in \mathbb{B}^{ack}_{d^{+1}}) \tag{R 5.6}$$

$$l^{d^{+3}} = (d \in \mathbb{B}^{ack}_{d^{+3}} \vee d^{+1} \in \mathbb{B}^{ack}_{d^{+3}}) \wedge (d^o \in \mathbb{B}^{ack}_{d^{+3}} \vee d^{o-1} \in \mathbb{B}^{ack}_{d^{+3}}) \tag{R 5.7}$$

$$a^{d^{+1}} = \neg(a^{d^{+1}} \wedge a^n \wedge l^{d^{+1}} \wedge \neg l^{d^n}) \tag{R 5.8}$$

$$a^{d^{+3}} = \neg(a^{d^{+3}} \wedge a^n \wedge l^{d^{+3}} \wedge \neg l^{d^n}) \tag{R 5.9}$$

where the rule R 5.1 represents a precondition to obtain all agents, that need to be checked, depending on the direction d. Rules R 5.2 to R 5.4 evaluate the alignment condition for regular boundaries, and rules R 5.5 to R 5.7 for line-alike boundaries. Further, if neighbors d^{+1} and d^{+3} are line-alike, while the agent n is not, rules R 5.8 and R 5.9 block the alignment condition to those.

Connecton Decision

The logical operator a defined in the paragraph above establishes the alignment information for each neighbor $d^{+1,2,3}$ individually. In the next step, an agent has to decide, to which potential neighbor it then eventually connects, which is stored in the operator $c^{+1,2,3}$.

In the simplest case, an agent has detected only *one* aligned neighbor. In this case, it logically connects to this neighbor, which is shown in the following ruleset,

Ruleset 6. *Connection decision - single neighbor*

$$\forall\, d \in \mathbb{D}:$$

$$c^{+1} = a^{n} \wedge\ a^{d^{+1}} \wedge \neg a^{d^{+2}} \wedge \neg a^{d^{+3}} \tag{R 6.1}$$

$$c^{+2} = a^{n} \wedge \neg a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge \neg a^{d^{+3}} \tag{R 6.2}$$

$$c^{+1} = a^{n} \wedge \neg a^{d^{+1}} \wedge \neg a^{d^{+2}} \wedge\ a^{d^{+3}} \tag{R 6.3}$$

If more than one of the three neighbors appears to fulfills the alignment condition, an agent has to decide, which one of them fits best to its own boundary. While agents seek for the highest distance $\mathcal{D}^s(d)$ to determine the initial (and later the growing) direction of the boundaries is, relying on the highest $\mathcal{D}^s(d)$ of the aligned neighbors is actually a sub-optimal choice for seeking the optimal boundary connection. In scenarios, where two different boundaries join, the higher $\mathcal{D}^s$ of one aligned neighbor would bring an agent to establish the connection to it, while the 'true' boundary might actually continue towards another aligned agent. In a nutshell, the strongest boundary segment an agent can see, must not necessarily be part of the same boundary it describes.

Deciding on the course of the boundary based on the distances $\mathcal{D}$ towards the aligned neighbors turns out to be a better choice to follow the correct boundary course. This ensures, that it tracks its own boundary. The prerequisite, that a boundary should occur in the direction of the highest distances $\mathcal{D}^s(d)$ of an agent is still given implicitly, as the aligned neighbors already set their aligned boundaries according to that rule.

Due to the symmetry mentioned above, all agents seek the boundary connection in clockwise direction, which fits overall best to themselves *and* their respective boundary neighbors d. Therefore, the agents do not simply connect to the aligned neighbor with minimum distance towards themselves. Instead they also take the distance between their boundary neighbor d and the aligned neighbor's boundary neighbor into account and seek for the lowest sum out of these two distances, called $\mathcal{D}_n^{pair}$. This paired distance additionally increases the robustness, as it becomes harder for an agent to choose a local minimum based on the distance.

The condition to choose the best connection c from multiple aligned neighbors is expressed by the following ruleset

Ruleset 7. *Connection decision - choose best neighbor*

$$\forall\, d \in \mathbb{D}:$$

$$\mathcal{D}_n^{pair+1} = \mathcal{D}_n(d^{+1}) + \mathcal{D}_d(d^{+1}) \tag{R 7.1}$$

$$\mathcal{D}_n^{pair+2} = \mathcal{D}_n(d^{+2}) + \mathcal{D}_d(d^{+2}) \tag{R 7.2}$$

$$\mathcal{D}_n^{pair+3} = \mathcal{D}_n(d^{+3}) + \mathcal{D}_d(d^{+2}) \tag{R 7.3}$$

$$\begin{aligned} c^{+1} = c^{+1} \vee\ & \\ & (a^n \wedge\ a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge \neg a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} < \mathcal{D}_n^{pair+2}) \vee \\ & (a^n \wedge\ a^{d^{+1}} \wedge \neg a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} \leq \mathcal{D}_n^{pair+3}) \vee \\ & (a^n \wedge\ a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} < \mathcal{D}_n^{pair+2} \wedge \mathcal{D}_n^{pair+1} \leq \mathcal{D}_n^{pair+3}) \end{aligned} \tag{R 7.4}$$

$$\begin{aligned} c^{+2} = c^{+2} \vee\ & \\ & (a^n \wedge\ a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge \neg a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} \geq \mathcal{D}_n^{pair+2}) \vee \\ & (a^n \wedge \neg a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+2} \leq \mathcal{D}_n^{pair+3}) \vee \\ & (a^n \wedge\ a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} \geq \mathcal{D}_n^{pair+2} \wedge \mathcal{D}_n^{pair+2} \leq \mathcal{D}_n^{pair+3}) \end{aligned} \tag{R 7.5}$$

$$\begin{aligned} c^{+3} = c^{+3} \vee\ & \\ & (a^n \wedge\ a^{d^{+1}} \wedge \neg a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} > \mathcal{D}_n^{pair+3}) \vee \\ & (a^n \wedge \neg a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+2} > \mathcal{D}_n^{pair+3}) \vee \\ & (a^n \wedge\ a^{d^{+1}} \wedge\ a^{d^{+2}} \wedge\ a^{d^{+3}} \wedge \mathcal{D}_n^{pair+1} > \mathcal{D}_n^{pair+3} \wedge \mathcal{D}_n^{pair+2} > \mathcal{D}_n^{pair+3}) \end{aligned} \tag{R 7.6}$$

which is executed complementary to ruleset 6.

While rulesets 6 and 7 cover a broad range of scenarios, there are two cases, which require a special treatment by the agents, including additional rulesets.

Both cases display situations, which mark the transition between the boundary connection scenarios, and the boundary grow scenarios. This means, that an agent can detect some neighbored boundary segments, but the boundary model of the neighbors is likely to be incomplete.

In the first scenario, shown in figure 3.8, the agent's neighbor in boundary direction d is a line-alike agent and neighbor d^{+1} is aligned. Following ruleset 6, the agent would connect to d^{+1}. However, as the boundary neighbor is a line-alike agent, it could potentially also be

connected to d^{+1}. This would imply agent n being connected to either d^{+2} or d^{+3}, which then would need to grow the corresponding boundaries. The decision, whether the neighbor d^{+1} is connected to n or its neighbor d again depends on a comparison of the paired distances $\mathcal{D}_n^{pair}$, and is displayed in the following ruleset

Ruleset 8. *Connection decision - line-alike boundary neighbor*

$$\forall\, d \in \mathbb{D}^{\oplus}:$$

$$c^{lnb} = (c^{+1} \wedge l^{d} \wedge \mathcal{D}_n^{pair+1} > \mathcal{D}_n^{pair+2} \wedge \mathcal{D}_n^{pair+1} > \mathcal{D}_n^{pair+3}) \tag{R 8.1}$$

$$\{c^{+1} = \text{FALSE} \mid c^{lnb}\} \tag{R 8.2}$$

which negates the connection c^{+1}, thus inducing boundary growth, if the paired distance towards d^{+1} is not the minimum one.

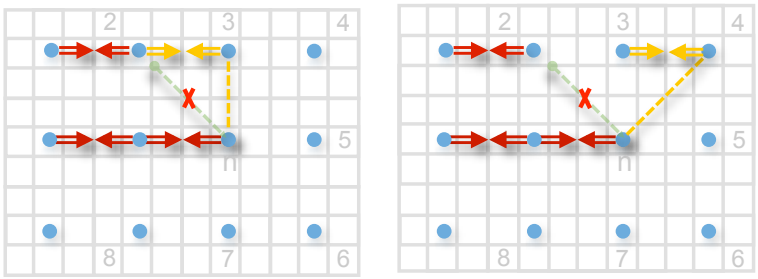


Figure 3.8.: This figure shows an example of a line alike neighbor in direction ($d = 1$), with the agent n deleting the connection to d^{+1} and inducing boundary growth instead. The left figure shows growing between d^{+1} and d^{+2}, the right figure between d^{+2} and d^{+3}.

The second scenario, which needs a special treatment is shown in figure 3.9 (left). It can occur, when an agent holds only one single acknowledged boundary in diagonal direction (i.e. $d \in \mathbb{D}^{\otimes}$) and is thus connected to its two neighbors d^{+1} and d^{+1}. Yet, if these two neighbors hold an acknowledged boundary between them, a *connection conflict* occurs, as they are at the same time indirectly connected via the agent n. The agent n solves this conflict by setting a boundary grow request to one of the two neighbors, and the agent logically again chooses the one with the higher distance $\mathcal{D}_n$, as shown on the right of figure 3.9.

The symmetry in all rulesets again allows an agent to only take care of a boundary request towards its neighbor d^{+1} in clockwise direction. If the comparison between the distances

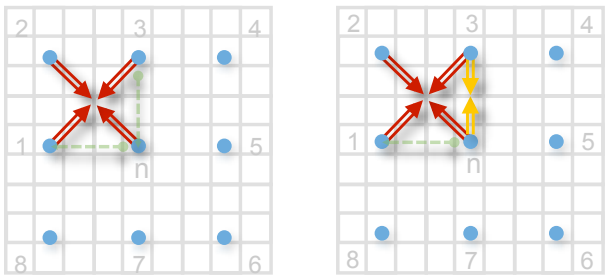


Figure 3.9.: The left figure shows an example of the connection conflict for direction ($d = 2$). The right figure shows the solution of this conflict by adding grow-boundaries to d^{+1}.

induces a boundary towards neighbor d^{-1}, it will be this neighbor taking care of it, forced by the same rule.

Mathematically, the detection and solution of this connection conflict is described in the following ruleset:

Ruleset 9. *Connection decision - connection conflict*

$$\forall\, d \in \mathbb{D}^{\otimes}:$$

$$c^{conf} = c^{+1} \wedge d^{-1} \notin \mathbb{B}_n^{ack} \wedge d^{-2} \in \mathbb{B}_{d^{+1}}^{ack} \wedge \mathcal{D}_n(d^{+1}) > \mathcal{D}_n(d^{-1}) \tag{R 9.1}$$

$$\{c^{+1} = \text{FALSE} \mid c^{conf}\} \tag{R 9.2}$$

$$\{d^{+1} \in \mathbb{B}_n^{grw}, d^{o+1} \in \mathbb{B}_{d^{+1}}^{grw} \mid c^{conf}\} \tag{R 9.3}$$

All agents that successfully detected connections via the rules stated above, have to update their models, i.e. the sets for boundary and boundary connections.

Ruleset 10. *Connection decision - update $\mathbb{B}_n$ and $\mathbb{C}_n$ sets*

$$\forall\, d \in \mathbb{D}:$$

$$\{d \in \mathbb{B}_n,\ d^{+1} \in \mathbb{C}_n \mid c^{+1}\} \tag{R 10.1}$$

$$\{d \in \mathbb{B}_n,\ d^{+1} \in \mathbb{B}_n,\ d^{+2} \in \mathbb{C}_n \mid c^{+2}\} \tag{R 10.2}$$

$$\{d \in \mathbb{B}_n,\ d^{+1} \in \mathbb{B}_n,\ d^{+2}, \in \mathbb{B}_n,\ d^{+3} \in \mathbb{C}_n \mid c^{+3}\} \tag{R 10.3}$$

The boundaries are afterwards acknowledged by repeating the corresponding rule 1.

Boundary Grow Inducement

As already mentioned, in some scenarios agents force their neighbors to grow boundaries in certain directions, so they can connect to it afterwards. I call this procedure *boundary inducement*.

Two such scenarios were already stated above in rulesets 8 and 10. The general case takes place, if an agent cannot connect to any neighbor by executing the already mentioned rulesets.

To select the best suited neighbor for inducing boundary growing, the agents search for the highest 'paired´ sum of distances $\mathcal{D}^{s,pair}(d^{+1,2,3})$. Once detected, they place a request or suggestion for the corresponding boundary direction in the neighbors' set $\mathbb{B}^{grw}_{d^{+1,2,3}}$. As shown in figure 3.10, agents set their growing boundary requests in their corresponding direction the best suited of their 3 potential connection neigbhors plus the neighbor opposing in this direction.

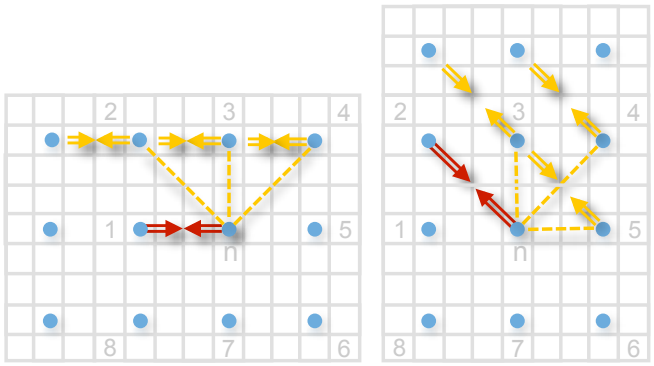


Figure 3.10.: The figure shows all potential boundary grows an agent can set to its connection neighbors and their respective potential boundary neighbors. The left image shows an example of a straight direction (d = 1), the right image for a diagonal direction (d = 2).

Using $\mathcal{D}^{s,pair}$ instead of $\mathcal{D}^{s}$ again happens for the reasons of symmetry. For the same reason, an agent sets the boundary suggestion to both its neighbor and the respective neighbor opposing it.

Mathematically, the following ruleset expresses this procedure:

Ruleset 11. *Boundary grow - induce best pair*

$$\forall\, d \in \mathbb{D}:$$

$$g^{n} = a^{n} \wedge \neg c^{+1} \wedge \neg c^{+2} \wedge \neg c^{+3} \tag{R 11.1}$$

$$\mathcal{D}^{s,pair}_{d^{+1}} = \mathcal{D}^{s}_{d^{+1}}(d) + \mathcal{D}^{s}_{d^{d+1}}(d^{o}) \tag{R 11.2}$$

$$\mathcal{D}^{s,pair}_{d^{+2}} = \mathcal{D}^{s}_{d^{+2}}(d) + \mathcal{D}^{s}_{d^{+1}}(d^{o}) \tag{R 11.3}$$

$$\mathcal{D}^{s,pair}_{d^{+3}} = \mathcal{D}^{s}_{d^{+3}}(d) + \mathcal{D}^{s}_{d^{+2}}(d^{o}) \tag{R 11.4}$$

$$\{\mathcal{D}^{s,pair}_{d^{+1}} = 0 \mid c^{lnb}\} \tag{R 11.5}$$

$$g^{+1} = g^{n} \wedge \mathcal{D}^{s,pair}_{d^{+1}} > \mathcal{D}^{s,pair}_{d^{+2}} \wedge \mathcal{D}^{s,pair}_{d^{+1}} \geq \mathcal{D}^{s,pair}_{d^{+3}} \tag{R 11.6}$$

$$g^{+2} = g^{n} \wedge \mathcal{D}^{s,pair}_{d^{+1}} \leq \mathcal{D}^{s,pair}_{d^{+2}} \wedge \mathcal{D}^{s,pair}_{d^{+2}} \geq \mathcal{D}^{s,pair}_{d^{+3}} \tag{R 11.7}$$

$$g^{+3} = g^{n} \wedge \mathcal{D}^{s,pair}_{d^{+1}} < \mathcal{D}^{s,pair}_{d^{+3}} \wedge \mathcal{D}^{s,pair}_{d^{+2}} < \mathcal{D}^{s,pair}_{d^{+3}} \tag{R 11.8}$$

$$\{d \in \mathbb{B}^{grw}_{d^{+1}}, d^{o} \in \mathbb{B}^{grw}_{d^{d+1}} \mid g^{+1}\} \tag{R 11.9}$$

$$\{d \in \mathbb{B}^{grw}_{d^{+2}}, d^{o} \in \mathbb{B}^{grw}_{d^{+1}} \mid g^{+2}\} \tag{R 11.10}$$

$$\{d \in \mathbb{B}^{grw}_{d^{+3}}, d^{o} \in \mathbb{B}^{grw}_{d^{+2}} \mid g^{+3}\} \tag{R 11.11}$$

with d^{d+1} marking the neighbor in direction d of the agent's neighbor d^{d+1}. By setting the distance value to 0, rule R 11.5 respects the special decision ruleset 8, which assigns the neighbor d^{+1} to the boundary neighbor if the condition c^{lnb} is met.

Boundary Grow Decision

In the last mentioned rulesets, agents possibly communicated boundary suggestions in certain directions to their surrounding neighbors. Consequently, all agents now have to check their $\mathbb{B}^{grw}$ sets to see if they received one or multiple boundary requests from their neighbors. If so, they choose the best suggestion and add the direction in the set $\mathbb{B}_{n}$. To find the best boundary, the agents take the highest summed distances $\mathcal{D}^{s}$, which is shown in the following ruleset

Ruleset 12. *Boundary decision -choose best*

$$b = \underset{d \in \mathbb{D} \ \wedge \ d \in \mathbb{B}_n^{grw}}{\arg\max} \ \mathcal{D}_n^s(d). \tag{R 12.1}$$

$$\forall\, d \in \mathbb{D} :$$

$$\{d \in \mathbb{B}_n \mid b\} \tag{R 12.2}$$

Once the agent have set new boundaries this way, ruleset 1 is again executed to update their acknowledged boundary set $\mathbb{B}_n^{ack}$.

Iteration stop break condition

During the growing phase consists of rulesets 1 to 12 are successively executed for a number of iterations. The stop condition for the growing phase needs one of the following two conditions fulfilled. Either no more growing takes place, i.e. no boundary suggestions are set in $\mathbb{B}^{grw}$ at any agent. Or a maximum number τ^t of iterations t is reached.

Mathematically, the stop condition q is expressed by
Ruleset 13. *Iteration break condition*

$$q = \mathbb{B}^{grw} = \varnothing \vee t > \tau^t \tag{R 13.1}$$

Boundary Model Finalization

To prepare the consecutive image information extraction phase, all boundary agents execute a final update of their model. To this point, an agent is either entirely within a segment, or it is part of a boundary, and then holding at least one boundary direction and two connection directions in the corresponding model sets $\mathbb{B}_n^{ack}$ and $\mathbb{C}_n$. For the majority of agents, their model can already be considered finalized.

The remaining scenario missing is the one of boundary intersections. While the agents usually solve most of such intersections already in the growing phase, at some points, it can occur, that an agent has already fixed its model and does not react on a further adaption request in the growing phase.

To detect and adapt to such unfinished boundaries, each agent first has to check, if any relevant neighbor holds a boundary connection estimate pointing in direction of the agent, i.e., $d^o \in \mathbb{C}_d$. A neighbor is considered relevant, when it lies in the connection region of the agent, i.e. when the agent does not hold a boundary or a boundary connection in direction of the neighbor, thus $d \notin \mathbb{B}_n \wedge d \notin \mathbb{C}_n$.

Once an agent detects such a neighbor, it has to adapt its own boundary model to it, by setting its boundary connection model in the neighbor's direction in $\mathbb{C}_n$ and filling all open directions towards it with boundaries in $\mathbb{B}_n$. Hence, an agent also has to find out, whether to enlarge the boundary model in clockwise, or in counter-clockwise direction. It does so by checking the boundaries in the neighbor's model in a clockwise and counter-clockwise direction. Figure 3.11 shows an example for this step.

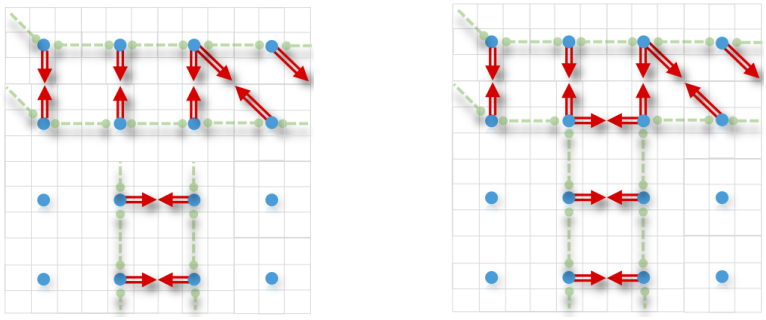


Figure 3.11.: Left figure shows an example of an intersection scenario, that is treated in the model finalization step. The right figure shows the adaption of the two agents in the middle.

3.2.4. Image Information Extraction - Phase

Once the boundary growing phase has finished, all boundary agents have encountered models holding a boundary description which is aligned and embedded in global boundaries.

In the subsequent *image information extraction phase*, the algorithm now is to extract image information out of the agents' set, i.e. the appearance, location, and shape of image segments, their boundaries and the positions of distinguished image points. To achieve this, agents group themselves into "swarms" according to the segments and their boundaries. This procedure will be detailed in the following.

Region Swarms

To extract such region- or respectively segment-based information, agents that lie inside the same region must connect to each other and build a group, or respectively *swarm*. They do so by sharing the same id, which is called the region-swarm ID sw^{reg}.

To achieve this, they utilize the following procedure consisting of sequential connecting and a region-fusion step:

1. First, all agents connect to each other, which have at least one neighbor, that neither lies in direction of a boundary, nor of a boundary connection, i.e. $d \notin \mathbb{B}_n^{ack} \wedge d \notin \mathbb{C}_n$. This step includes all non-boundary agents. All now connected agents then share a swarm ID sw^{reg}, which computationally can be achieved by an 8-neighbor connected component algorithm.
2. All agents, that did not obtain an ID from the previous step, use their boundary connection information to connect and spread a sw^{reg}. This way, all agents in line-alike scenarios receive an ID.
3. Now the sw^{reg} and the agents pixel window information is used to calculate the mean region image appearance $\mathcal{I}^{reg} = \mu(\mathcal{I}(sw^{reg}))$ of each region.
4. Neighboring regions are then fused, if the difference between their mean region appearance is smaller than a threshold τ^{reg}. The region IDs of agents in fused regions are then adapted accordingly and the mean region appearance $\mathcal{I}^{reg}$ is re-calculated.

This procedure first leads to a slightly over-segmented result, which is then compensated by the fusion step. This has an advantage over the naive straight-forward approach of simply connecting all agents in directions that are no boundaries: in the fusion step, the potentially over-segmented regions are compared using the overall mean appearance of the regions considered, not only the local window information of the corresponding neighbored agents. This makes it more robust against blurry region borders. Thus it can prevent from under-segmentation.

Once all agents have allocated themselves to a swarms in this way, extracting region-based image information is straight forward.

- The most relevant segment information is the segment's *appearance*. It can be extracted as mean of the color information by calculating the mean over all windows $\mathcal{W}_n^{px}$ of the agents belonging to that segment, like already stated above as $\mathcal{I}^{reg}$. This can of course happen in any image color space. Alternatively also histogram based information can be extracted, based on all pixels inside the agents' windows.

- Analog, as position information, the *centroid* of each region calculates as mean from the positions of all agents belonging to the swarm.
- Accordingly, any other geometrical information, the segment's shape or orientation, i.e. the *image or Hu moments* [48], can easily be obtained using the positions of the grouped agents.
- Further, the exact shape of a segment is described by the course of its boundary. Extracting this information is subject-matter of the following section.

Thus, the region swarms allow for a description of regions, which is both effective and comprehensive.

Boundary Swarms

Additionally to the region swarm ID, the boundary agents share a boundary swarm ID sw^{bnd}. This ID allows to describe all segment boundaries detected by the agents.

With the establishing of sw^{reg}, building boundary warms is relatively simple: Each agent that holds an acknowledged boundary, i.e. $\mathbb{B}_n^{ack} \neq \varnothing$ connects to all agents in its neighborhood, which themselves are boundary-agents and possess the same sw^{reg} as the agent.

Then one can again utilize each swarm of agents to extract information from the region boundaries, or respectively the edges in an image. The boundary or edge information can be *appearance based*, i.e. referring to the strength of a boundary, or *geometry based*, i.e. regarding an edges shape, length, and localization.

To obtain an effective measure of the boundary appearance or strength, the mean Distance of a boundary to the neighbored boundary is calculated, in terms of the agent distances as well as in terms of the mean region differences.

Similarly, the length of a boundary is simply the number of agents' pixels that form the boundary.

Curvature

To describe the course or shape of a boundary, each agent estimates the *local curvature* of its location by taking into account the relative positions of its connected neighbors. The approach follows [4], which also delivers dominant points on the boundaries, based on high curvature values. These dominant points can then also used to split a boundary, which

entails an entire region of arbitrary size, into interconnected boundary sub-segments, thus describing meaningful edges. The subsequent subsection will detail, how to attain and use these dominant points, as well as the boundary splitting. Yet before this, I explain, how to calculate the curvature, as well as how to utilize it in the context of boundaries.

The curvature $\kappa(ag)$ of an agent at point (x, y) on the swarm is defined as the rate of change of the slope angle of the tangent at point (x, y), with respect to the arc length. The method in [4] applies prior smoothing to the planar curve, represented by *sw*, by convolving a one dimensional Gaussian kernel with x and y coordinates along t independently, such that

$$X(t) = x(t) * g(t, \sigma), \tag{3.7}$$

where $g(t, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-t^2}{2\sigma}^2)$ with σ as the kernel size. The same is applied to the y-coordinates to obtain $Y(t)$. The curvature κ of the smoothed curve is now defined as

$$\kappa(t, \sigma) = \frac{\dot{X}\ddot{Y} - \dot{Y}\ddot{X}}{(\dot{X}^2 + \dot{Y}^2)^{3/2}}, \tag{3.8}$$

where t is the path length along the curve, σ is the width of the Gaussian Kernel and $\dot{X}, \ddot{X}, \dot{Y}, \ddot{Y}$ are the first and second derivatives of $X(t)$ and $Y(t)$ with respect to t. The derivatives can be approximated by finite differences as

$$\dot{X} = X(t+1) - X(t-1) \tag{3.9}$$

$$\ddot{X} = X(t+1) + X(t-1) - 2X(t). \tag{3.10}$$

The same counts for Y. Traversing the curve along t a positive curvature means a concavity on the left, and negative curvature means concavity on the right in an image.

Using the curvature as shape descriptor of boundary segments, one can easily obtain all straight edges or lines as a subset, by 'filtering' for curvature values, that are below a threshold, i.e. $|\kappa(t, \sigma)| < \tau^{sl}$. The threshold τ^{sl} allows for a certain tolerance in defining, which edges are considered straight lines.

Anchorpoints

An important goal of the image description concept introduced in this thesis is to coevally enable extraction of region-based, edge-based, and interest point-based information. In the following, the key step in extracting interest points from the edge information introduced

in the previous chapters is displayed. I call the interest points that obtained in this work *anchorpoints*.

To obtain an efficient description and matching strategy, geometrical, or respectively structural anchorpoints are utilized to additionally subdivide an extracted feature swarm, or respectively salient edge, as will be explained in the next chapter. These anchorpoints are located at distinct positions on an edge, describing prominent changes of the curse of an edge.

Finding points with local maxima and minima in the curvature will result in a structural interesting points. To discard anchor points, that are in a almost straight line segment, I therefore set a threshold τ^{ap} for the curvature, thus an anchor point is defined if $\kappa(t, \sigma)$ is a maximum or minimum and $|\kappa(t, \sigma)| > \tau^{ap}$. For example, having a anchor points in a swarm sw_i, the edge-swarm can now be split into sub-swarms, such that a single swarm sw_i is now represented by a set of sub-swarms $sw_i = \{sw_{i1}, sw_{i2}, ..., sw_{ib}\}$, where each swarm $sw_{ij} = \{ag_{ij}^1, ag_{ij}^2, ..., ag_{ij}^\lambda\}$ and $b = a - 1$ as the number of sub-swarms.

Figure 3.12 shows an example result for the image information extraction, the input image, the segmented regions' mean appearance, and the contours, plus detected straight lines and anchorpoints. Figure 3.13 details the concept of edge curvatures and anchorpoints.

3.3. Interpretation of SI Principles in SISeg

In this section, I will introduce the general idea of how the *SISeg* concept follows Swarm Intelligence. Further, I will categorize the concept's main mechanisms in terms of SI-key paradigms and state, how they were applied and realized, and how the principles where weighted to each other in terms of importance and applicability to this problem field.

Self-Assembly

As mentioned in chapter 2, the two major Swarm Intelligence application fields are optimization algorithms and swarm robotics. While the first are mainly driven by foraging and similar concepts, realizations of the concepts of *Self-Assembly* are predominantly found in swarm-robotics. Mostly, optimization algorithms result in a single location in a search space as final solution, with the Ant Colony Optimization depicting a prominent exception, where the pheromone concentrations after a certain number of iterations, often being binarized via a threshold, is regarded as the global solution.

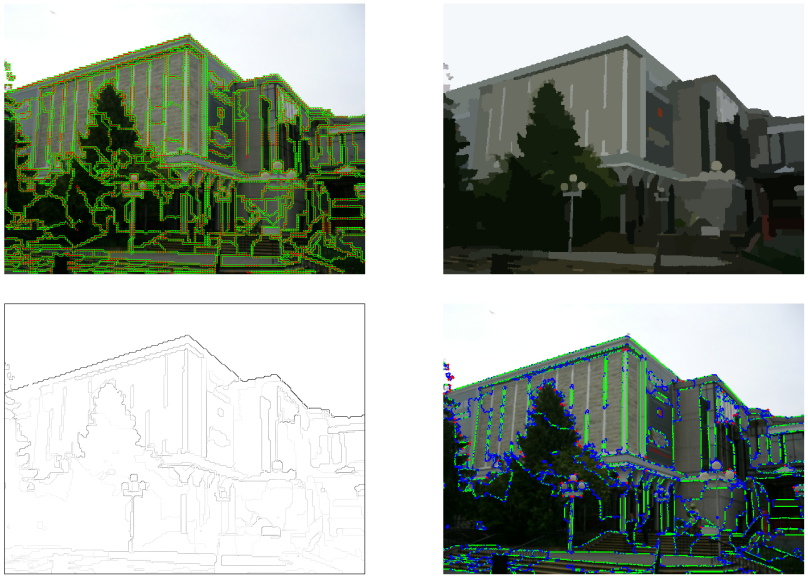


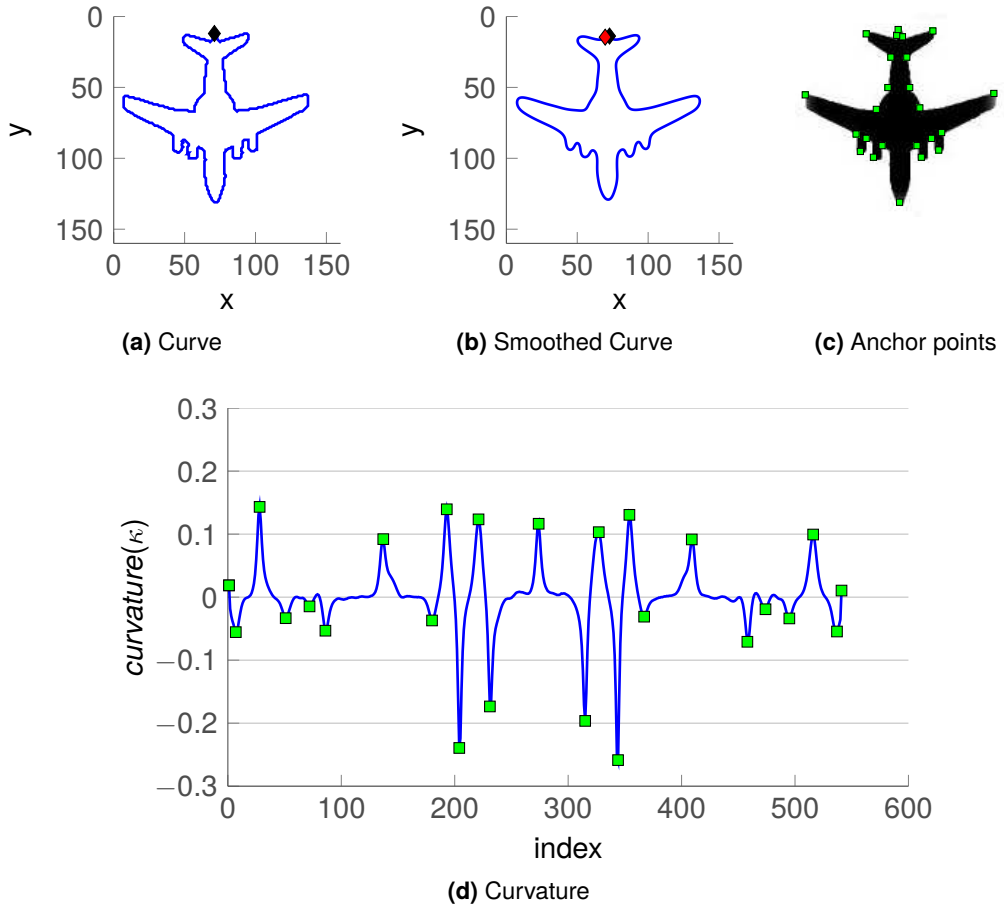
Figure 3.12.: This figure shows an example of SISeg extraction steps. Top left: original image with agent models including boundary directions (red) and boundary connection (green) models. Top right: segmented regions' appearance. Bottom left: extracted contours, bottom right: final boundaries on pixel level (blue), extracted straight lines (green) and anchor points (red). Image taken from [29]

However, the nature of the task of image segmentation inherently demands a sort of distributed solution, where a segment, or respectively its boundary in one location of an image can be completely independent of a different segment in another position, while just as well, segments that span over the entire image can occur.

Thereby, I decided to apply the concept of *Self-Assembly* to the segmentation task, as it allows for agents to locally cooperate to establish a boundary, while coevally being either completely unaffected in their behavior by the actions of agents in more distant locations or indirectly connected in the same boundary. Dependence between all the agents in a swarm must therefore be possible but not compulsory.

This composition relates to the statement introduced in section 2.1.3, that *Self-Assembly* takes place in an intermediate level between the level of a single individual and the level of the whole colony.

Considering examples from the nature, the formation of boundaries can for instance be compared to the phenomenon of ants defending the entries of their hill from intruders by interlacing with each other, building a living wall. Likewise, the agents in the *SIseg* con-



(a) Curve **(b)** Smoothed Curve **(c)** Anchor points

(d) Curvature

Figure 3.13.: (a) shows the detected (single) swarm by SISeg. Start and end point are depicted as black diamond. (b) shows the curve smoothed by a Gaussian kernel with σ = 3.85 and a window size of 21. (c) depicts the image with the detected anchor points. (d) gives the resulting curvature with the found anchor points overlayed as green rectangles with τ^{ap} = 0.01.

cept sort of fold their “arms” with connected neighbors and disconnect to neighbors of the separated regions.

Autonomy and Cooperation

Instead of a centralized control, boundaries establish by the interplay of the influence of the surrounding environment (including the neighboring agents), and each agent’s desire to choose the optimal possible model. While the first aspect represents cooperation, the latter part causes each agent decide its contribution to the structure autonomously.

Communication

The agents operate in a *search space*, which consists of the localized image distances plus the current boundary model estimations of the neighbors. Both the image input and the surrounding estimates trigger reactions in each agent.

Thus, the algorithm does not follow a direct communication scheme, which would mean a request-answer structure between the agents, but a loop of observing, reacting and by reaction influencing the environment again. This clearly makes the communication *stigmergic*.

Memory

In Swarm Intelligence based approaches, each agent usually possesses a kind of limited memory, which often consists of a solution experienced in the past and influences an agent's future decisions. In the *SISeg* algorithm each agent memorizes the state of its boundary models in loop growing phase, which it utilizes to establish a final boundary connection. The memory is thus crucial for decisions and constantly updated.

Iterative Structure

An element which is inherent in all realizations of SI based concepts is the iterative structure, as any agent is to adapt to its environment in course of finding a optimum or equilibrium, or simply convergence. The iterative structure in *SISeg* is found in the growing-loops, i.e., when agents adapt their boundary models to connect to global boundary estimates.

Randomness

In optimization algorithms, randomness is often added into decision processes in form of noise or random weights. This represents a key factor to overcome local minima while exploring the search space. In this approach, randomness does not add a benefit to decisions on models. Instead, a deterministic approach at initialization and growing, which depends merely on comparison of the distances allows for optimal and repeatable results. Therefore, randomness was not integrated in this concept.

3.4. SISegConcept Evolution

When I first considered to utilize Swarm Intelligence on the image segmentation problem, there were logically merely the basic principles, and thereby nearly endless possibilities to realize these concretely in an application. Section 2.2 might give a hint on how diverse the principles can be carried out. In the following I want to give only a few examples on the variants I created in course of creating the SISeg algorithm. This list is by far not complete.

Movement

Besides several movement approaches, a main approach was to initially let agents perform a local steepest descend search, forcing agents to position at local maximum distance values. After this positioning, agents used communication to establish the contour descriptions. This method was part of the method published in [57]. As an opposing approach introduced in this thesis, agents stay on a fixed grid and thus do not apply movement on the image. This significantly simplifies the local geometry and communication structure to a eight-neighborhood method, which then increases efficiency. Yet, it shall be made clear, that movement in Swarm Intelligence based optimization approaches refers to movement in an arbitrary search space. The iterative adaption of the boundary models in *SISeg* corresponds to a movement in a discrete space of possible boundary and connection model combinations.

Connection Strategy

In course of this work, I created diverse concepts for boundary generation and boundary connection strategies, utilizing the differences between agents in various ways. Finding the correct neighbor to connect to at a boundary demands two important aspects. First, the agent to connect to shall have a preferably minimum difference or distance to the agent $\mathcal{D}_n(d)$, and second, it shall have a high boundary fitness $\mathcal{F}_n^{\mathsf{B}}$, i.e. the distance of the neighbor towards the boundary. Basically, these two values are to ensure coevally, that an estimated boundary follows the real region boundary by sticking to high differences between regions, and that it also does not switch over to other regions. Further, the boundary model geometrically have to suit to each other.

To decide the boundary course, I thus tried different combinations of the values of $\mathcal{F}_n^{\mathsf{B}}$ and $\mathcal{D}_n(d)$. My intuitive approach were weighted sums, which I tested with different combina-

tions of weights, including also the two extremes of weighting one term zero, i.e. to rely either on the difference value, or on the boundary fitness exclusively. A notable variation of the weighted sums is an approach, that utilized *Harmonic Mean* of the two values. This gave allowed a “natural” dynamic way to weight the two values to each other. Additionally I introduced a lower boundary, below which the *Harmonic Mean* was replaced by a minimum value threshold, as it turned out to work best within a certain range of boundary strength.

Further variants included combinations with normalizing the values in order to get rid of the influence of absolute values, as local maxima could distract form the correct boundary course. Yet, getting rid of this is preferable in some but not all situations. Another variant, which worked fair enough, consists of a test, which compares the distances in boundary and connection candidate directions simply in a binary manner. This led to both stable estimations of a wide range and high computational efficiency.

The second aspect, on which I implemented and tested a whole variety of approaches, was the connection strategy, which tells an agent, which neighbor(s) to choose for connection. All variations included the aspects of 1) connecting to neighbors with high (maximum) gradients, 2) avoiding degenerate, or respectively obtaining smooth boundary shapes if possible by constraining the local geometry, and 3) meeting the real image segment boundary course as good as possible.

Summarizing, all approaches worked well to estimate the boundary course in most image scenarios. The different variants allowed to push performance in some ambiguous situations. Yet, one can always find scenarios in real world images, which caused the fitness value to deliver false detection.

All variants eventually led to the final strategy of *SISeg*, which separates the two aspects of high boundary distances plus low connection distances simply into separate rules, each taking care on its own.

Chapter 4.

Image Correspondence Matching

In the previous chapter I introduced an image analysis concept, which delivers exhaustive image information of region-, edge-, and point-based type. However, in case of image sequences also temporal information is available. For depth map generation, the temporal information can be utilized for movement-based depth estimation on the one hand, and for increasing robustness by tracking image depth content and checking its consistency over time on the other, thus removing outliers.

4.1. Background on Feature Description and Matching

In this section I give an overview of recent feature description and matching paradigms, concerning Point, Curve, Line and Shape Matching. Let it be noted, that this overview is by no means a complete survey, but rather an excerpt of algorithms rather similar to the problem statement. For exhaustive reviews and evaluations of feature detectors and descriptors take a look at following works: For point-based features [75, 116, 40] and for Shape-based features [119, 131, 76]. In [17] a thorough review of all visual tracking, including line-based features, is conducted. Related work related to narrow- and wide baseline matching is introduced in the next section.

4.1.1. BRIEF Feature Descriptor and Matching

The Inter-Point-BRIEF (I-BRIEF) Matching algorithm, which I describe in Section 4.3, is an extension of the Binary Robust Independent Elementary Features (BRIEF) concept to describe and match line-features. Therefore the original BRIEF, descriptor is introduced in the following.

Point based descriptor and matching strategies have been extensively investigated over decades and logically there exists a broad spectrum of approaches (see e.g. [102]). Since its introduction in 1999, the famous *SIFT* (Scale Invariant Feature Transform)-operator can be regarded as the spearhead in terms of robustness, being able to match points in complex wide baseline scenarios. It uses multi-scale extrema detection by Difference of Gaussian and histogram based extraction of edge orientation. The main - or only - drawback of the SIFT approach is its computational complexity. One of the most prominent alternatives to SIFT is the SURF (Speeded Up Robust Features) approach. It was developed as an approximation to the SIFT descriptor. It reduces computations by utilizing the Hessian matrix and box filters, which significantly speeds up its execution while introducing a considerably small quality loss.

There exists a variety of approaches in literature, which aim to obtain high robustness and quality while restraining the computation time. The concepts usually vary in the ways and parametrizations of calculation steps like scales, feature extractions, handling transformations, and the distance functions for matching.

In 2010, Calonder et al. in [16, 15] presented the BRIEF, which reduces calculations to binary operations, thus obtaining very high efficiency. They represent a image patch directly by a binary string of 128, 256 or 512 bits, while Scale-Invariant-Feature Transform (SIFT) needs 4096 bit and SURF-64 needs 2048 bit. The binary representation can be matched very efficiently using the Hamming Distance, which basically breaks down to a bit-wise XOR followed by a bit count. The hamming distance $hd(a,b)$ between two binary strings a and b of the same length n is mathematically expressed as

$$hd(a,b) = \sum_{a_i \neq b_i} 1, \; i = 1, \ldots, n. \tag{4.1}$$

As an example, the Hamming distance of the binary strings $a = 0010$ and $b = 0110$ results in $hd(a,b) = 1$.

The binary description of a image patch ψ is achieved as follows. The patch is first pre-smoothed by a Gaussian Kernel reducing noise-sensitivity. The smoothing can be achieved efficiently by using integral images without loosing significant accuracy. For the BRIEF descriptor, a normal distributed subset of pixel pairs (128, 256, 512 pairs) in the patch is chosen and pixel intensities are compared between the pixel positions $\mathbf{x}_1$ and $\mathbf{x}_2$, such that

$$\tau(\psi; \mathbf{x}_1, \mathbf{x}_2) = \begin{cases} 1 & \text{if } \mathtt{I}(\psi, \mathbf{x}_1)) < \mathtt{I}(\psi, \mathbf{x}_2) \\ 0 & \text{else.} \end{cases} \tag{4.2}$$

This method is similar to evaluating the sign of the derivatives within a patch. Four different sampling distributions drawn from random sampling (uniform and normal distributed) and a symmetrical and regular distribution are considered for determining the test positions in the patch. The random distributed intensity tests clearly outperform the regularized one. The original BRIEF is also called Upright-BRIEF (U-BRIEF), since no rotation is considered explicitly . While U-BRIEF is of course not rotation invariant, it achieves acceptable results in up to 15° of rotation. Scale changes are only considered by pre-smoothing, but the authors and conducted experiments suggest that U-BRIEF is robust to small and intermediate scale changes. For larger changes in scale a drop in accuracy can be observed. The results are promising and comparable to Upright-SURF (U-SURF) in terms of recognition rate. The achieved speed-up over SURF is tremendous. For descriptor computation a 35- to 41 fold and for matching 12- to 45 fold gain in computation time can be achieved, resulting from the utilization of binary operations.

4.1.2. Line- and Edge-based Features

Matching wide-baseline images is often based on local features. In scenes with low-textures these often fail due to the lack of distinctive information. Low-textured scenes, for example scenes showing human-made environments (houses, cities), most certainly will contain edges and therefore line segments in the form of approximated object boundaries, which can be used as correspondences instead. Matching methods for line- and edge-based methods can be categorized into three different types of approaches occurring in the literature [103, 132]

1. Match individual line segments with
   a) the nearest neighbor in image- or feature-space.
   b) feature Matching using some matching strategy.
2. Match point correspondences with
   a) uniform or arbitrary distributed points.
   b) apriori knowledge of the epipolar geometry.
3. Match groups of line segments with
   a) graph Matching, interpreting lines of an image as a graph and partially match different nodes.
   b) topology, use the spatial relations between feature-pairs in an image.

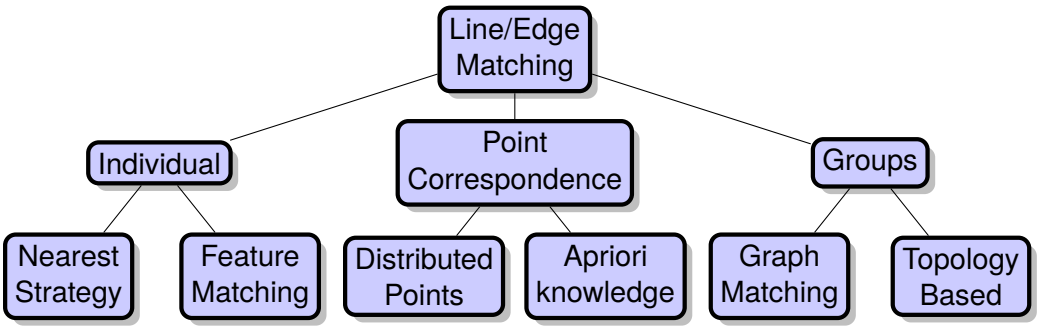


Figure 4.1.: Different matching concepts for line-and edge-based features

This categorization scheme is also depicted in Fig. 4.1. Edges and lines contain structural information of the scene and are especially important in low-textured scenes, since local features need a certain variance in image properties such as color or gradients to be reasonable discriminant. Additionally, edges and lines can be used in conjunction with point-based and region-based features, because they usually contain complementary information. As local features suffer from low contrast environments, edge- and line-based feature detection and description techniques suffer from inherent problems too, such as

1. inaccurate locations of line endpoints,
2. fragmentation of lines,
3. lack of disambiguate geometric constraints,
4. lack of distinctive appearance in low texture scenes.

In the following I will give a short overview of state-of-the-art narrow- and wide-baseline methods utilizing edge- or line-based features.

4.1.3. Matching Strategies for Features

Once features are detected and described, they need to be compared to other features for similarity. The similarity is measured using a distance function or a similarity measure, in most cases the Euclidean distance in the feature space is taken as the measure. Usually a matching strategy is applied to find the best match in one or more images having the best similarity or smallest distance. The matching strategy depends on the application itself. Since certain conditions are assumed, which will often not hold outside of the spe-

cific application. For example, assumptions about the spatial proximity of features between consecutive images in video sequences will not hold in general wide-baseline image pairs.

Often image degradation in different image pairs can vary a descriptor. Even if the same feature is located at the same correct pixel position in a different image the descriptor may not stay exactly the same as in the image it originates. Often these perturbations are caused by noise, image discretization, or because another part of an object is now detailed in the vicinity of the feature. These changes lead to a variation in the descriptor itself. A matching strategy alone is not able to match all features to their corresponding correct features, but rather reduce the number of false matches. These false matches, i.e. outliers, often correspond to image parts which are not shown in both images, e.g. background clutter or occlusion and cannot be matched anyway. To tackle this problem different matching strategies emerged in the literature. In the following I will discuss them shortly [111].

Nearest spatial neighbor Especially in feature tracking applications it can be assumed, that context between images is quasi-constant, as the captured movement is usually low compared to the frame rate of the sequence. Therefore it can be applicable to take the spatially *nearest* feature between two images as correspondence.

Thresholding - Another strategy is to take the feature below some threshold ϵ as a match, but it is difficult to find a threshold, which is generally applicable. Additionally, the useful range of thresholds can vary a lot in different parts of the feature space.

First Nearest Neighbor (1NN) - To overcome the problems of using a threshold directly on the matches, one attempts to find the nearest neighbor in the feature space. A threshold is still used to reduce outliers and the needed threshold can still vary for different parts in the feature space. This strategy is most common in visual tracking applications.

Nearest Neighbor Distance Ratio (NNDR) Another more capable strategy is to take a look at the ratio of the first and the second nearest neighbor in feature space. This is called the NNDR and is defined as $NNDR = \frac{d_1}{d_2}$ with d_1 and d_2 as the first and second nearest neighbor distance in feature space. This usually reduces the number of outliers and has become the de-facto standard for wide-baseline matching.

4.1.4. Discussion

In the following, I will explain and discuss various approaches. Starting with point-based features, which have become the de-facto standard for finding image correspondences. A lot of research has been conducted in this area in the last decades. Point-based concepts,

such as SIFT or SURF, are relatively stable under all possible image transformations, but directly rely on image data for describing patches or regions. Binary point-based features are related to aforementioned approaches, but are more computationally efficient in means of extraction and description while giving similar performance. Point-based features will most likely fail for image sequences, which lack descriptiveness by image properties alone.

Therefore more robust features can be introduced using edge- or line-based features, which contain significant structural information. Line segments can be tracked in image sequences with the nearest neighbor in image or feature space within a search region propagated in a prediction-observation scheme such as a Kalman Filter [26, 30, 81]. Other approaches exist, where control points facilitate the reestablishing of a line [43, 103]. Line segments can also be matched across wide-baseline images with different approaches, such as gradient-based [122], appearance-based [8, 132], topology-based [121], or via geometric constraints [103]. These concepts often rely on feature matching or costly graph matching techniques. The authors in [74] introduce an interesting edge-based description and matching approach for wide-baseline image pairs relying on gradient histograms, but fail to give specifics about computational complexity. These methods are usually capable to describe and match a large number of features across views and can be made robust to most occurring distortions. There already exist methods, which can inherently match edge-based contours like Mean and Standard Deviation Curve Descriptor (MSCD) from [122] and [74].

Approaches in the context of shape matching are rather found in applications related to object recognition, than finding arbitrary correspondences between images. Furthermore shape-based features try to find similar shapes in different images, rather than the same, possibly transformed, feature in different images. Active contours are used to either detect shape-based features with a evolution mechanism or are applied to (multi-) object tracking [128]. Either way, they need additional input for initialization. Due to their iterative nature they have high computational cost and are hardly applicable to the goals.

None of the above listed approaches can achieve fast matching for many edge based features in narrow- and wide-baseline image pairs. Using SISeg as input allows for arbitrary curves to match, instead of restricting to straight lines or line segments. Additionally I do not want to use a priori knowledge in the form of contour-models to stay applicable for the general case. The approach introduced in section 4.3 utilizes the concepts specified in this section.

4.2. Related Work on Narrow and Wide Baseline Maching

This section gives an overview on line and contour matching algorithms in narrow- and wide-baseline applications.

Narrow-baseline

Narrow baseline tracking of line and edge features is a rather well studied subject, as its inherent constraint of locality of image content also applies to video sequences. For the same reason it allows for prediction with Kalman Filter sets. The approach in [26], and an advancement of it in [30], are examples for applying a Kalman Filter, feeding with a special set of parameters of the edges of interest, like their lengths and positioning and orientation angles, in relation to a fixed origin, the movement of midpoints, etc., and matching using the Mahalanobis distance.

In [43] the authors proposed a line tracking algorithm called Real-time Attitude and Position Determination (RAPID). It assigns control points to a line and predicts these with an $\alpha - \beta$ filter or with a Kalman Filter. Afterwards a one dimensional search for gradient maxima, perpendicular to the according contour, is applied. The found control points are used to reestablish the position of a three dimensional model. The authors of [81] use an adapted RAPID tracker with a new way of handling multiple line hypotheses. Lines are extracted via Canny edge detector, the contours get linked and polygonized. A simple line segment is providedfor RAPID as model input and hypothetical lines are proposed in the neighborhood of the line segment.

In [33] the authors use edge landmarks in monocular Simultaneous localization and mapping (SLAM), which enables them to handle edges of arbitrary location and orientation. Therefore they define the edge features as a local portion of an edge, small, locally straight segments, which they call *edgelet*. Description and matching is then done utilizing local windows centered around the edgelet and creating small description vectors.

The authors of [103] utilize prior knowledge of the epipolar geometry to establish the line correspondence for narrow as well as wide-baseline matching. They first project the line i.e. the set of points with the known fundamental matrix F into the second image. Correspondences can be found by utilizing the fact that a point $\mathbf{x}$ on line $\mathbf{l}$ corresponds with a point $\mathbf{x}'$ on line $\mathbf{l}'$ when $\mathbf{x}' = \mathbf{l}' \times (\mathrm{F}\mathbf{x})$. To measure the similarity between line segments the average of the individual correlation score between common line pixels is used. A match

for **l** can then be found by calculating the matching score with all segments in an epipolar constrained search space and taking the best one results in a correspondence.

Wide-baseline

Scenarios, in which the viewpoint, rotation, or camera parameters differ significantly from one image to the other, are generally referred to as *wide-baseline* scenarios. Practical applications to wide-baseline matching is image stitching to create a single image with a panoramic view from multiple images taken from different viewpoints. Recently, line-based feature approaches emerged for wide-baseline images. In the following I will review some of these concepts.

Wang et al. [122] present a descriptor for line matching, which borrows some ideas from SIFT, called Mean and Standard Deviation Line Descriptor (MSLD). Lines are extracted with Canny edge detector and split at high curvature points to acquire line segments. They center a rectangular Pixel Support Region (PSR) at each such segment, which is oriented to the average gradient *dl* to achieve rotation invariance. After Gaussian weighting, and division in subregions, a Gradient Description Matrix (GDM) is formed, which contains all the structural information of the line neighborhood. Since the GDM still depends on the line length, the mean and standard deviation from each column vector are extracted, which are then again normalized for further scale invariance and used to form the descriptor. For a robust matching they apply Left-Right Checking (LRC) and NNDR. Initially, the method is for line segments, but using each pixel gradient instead of the mean gradient of the line to determine the PSR, the authors were able to extend their approach for general curves, called MSCD.

In [8] the authors introduce a combined line and region wide-baseline stereo matching algorithm, which is tailored to poorly textured scenes. After detecting lines using the Canny edge detector, they initially match them between images by the neighboring color profiles on both sides of the line, represented as histograms. For robustness against illumination changes, a quantized HSV color space is used. The dissimilarity of two segments is expressed by the square root of the mean of the histogram dissimilarities for both sides. The authors allow for one to many correspondences and store the best 3 matches. These so called soft matches help to match weakly distinctive line segments. After first correspondences are established, the matches (segments and regions) are iteratively refined with a topological filter, based on the semi-local spatial arrangements in two views. Subsequently,

more matches are found by iteratively reintroducing unmatched line segments, which respect the topological structure of the current set of matches. The algorithm performs better than SIFT when confronted with poorly textured, such as indoor, wide-baseline image pairs.

The authors of [121] cluster detected line segments into local groups, according to spatial proximity and relative saliency between segments. Edge pixels are extracted and linked into connected curves and split into straight lines at dominant points in a multi-scale fashion.

These groupings of line segments is called a Line Signature. The grouping mechanism, is based only on surrounding lines and is therefore scale invariant. The line segment pairs are described by a set of length and angles and the ratio of mean gradient magnitudes between line segment pairs, which allows for measuring affine as well as general similarity. Line signature similarity now remains the sum of the similarity between their corresponding segment pairs, which can be found by maximizing the segment mapping with respect to the similarity measure. Matching is then speeded up by a codebook approach.

In [132] lines are detected in a scale space approach using the EDLine [1] algorithm to each octave producing a set of lines in the scale space. Detected lines are initially grouped by spatial and directional extent. Through a histogram comparison made from the line directions, the global rotation can be approximated, which serves as a correction for perspective transformation in the case of global rotation. For describing the local appearance similarity between the remaining groupings a proprietary Line Band Descriptor (LBD) is used, which is quite similar to MSLD. Further pruning of matches is applied, based on a local appearance dissimilarity tolerance. Nodes in the relational graph consist of potential correspondences and the weighted links represent the pairwise consistencies much like the similarity measure in [121]. To solve the matching problem efficiently, a spectral technique is employed.

In [74] Meltzer et al. proposed a wide baseline correspondence algorithm based on edges, that are extracted via a scale space approach. Edgels are ordered via a linking mechanism in location and scale. Then for each side of the edge a support region is estimated, called the scale envelope. This envelope is determined by evaluating the integral of the Laplace-Operator along each edgel on both sides independently. Anchor points are then selected at extrema points of the envelope on both sides and are therefore depending only on image statistics, rather than edge geometry. Similar to SIFT, gradient histograms are extracted and weighted for each region. Matching is then achieved by a derivate of Dynamic Time Warping (DTW) the so called Smith-Waterman Algorithm, which was developed for protein sequence alignment [107]. To feed the edge descriptor to the Smith-Waterman matching,

the histograms are transformed into *letters* in a fixed alphabet by clustering all histograms with k-means clustering. The authors state, that the algorithm outperforms SIFT in domains where occluding boundaries are dominant, but no measure for computational efficiency is given.

4.3. Inter-Point-BRIEF Descriptor

In this section I introduce an approach for edge-based matching, which relies on inter-point relations. It utilizes Anchorpoint extraction from section 3.2.4 as input, but can cope with other edge detection algorithms as well. In this chapter, I will explicate the general idea of the concept, detail important sub-components and verify certain decisions with experiments on video sequences, taken from the dataset in [40]. A detailed description of the dataset can be found in subsection 6.2.2, along with a comparison with a state-of-the-art approach in the evaluation chapter.

The general pipeline for feature matching contains four steps

1. feature extraction using e.g. *SISeg*, as shown in detail in Section 3.2,
2. feature description,
3. feature matching.

The detection and extraction of salient contours in the *SISeg*-concept, which is described in Section 3.2 serves as input into this algorithm.

4.3.1. Inter-Point-BRIEF Descriptor Concept

I propose a novel description approach, which is based on BRIEF [16]. The point-based BRIEF descriptor is already described in Section 4.1.1. I call this concept the Inter-Point-BRIEF (I-BRIEF). It establishes a relation between *two* anchor points on an edge-based feature. Instead of evaluating intensity parameters inside of an image patch, as in BRIEF, the intensity tests are distributed between *two* patches. For two points on an edge, the I-BRIEF results in a correspondence-based edge descriptor. In the case, that both points coincide this method falls back to a BRIEF-like descriptor.

After stating the general idea, I now derive the approach in a similar manner as BRIEF. Equation Eq. 4.2 is extended with another patch around another point. That is, the test τ

between two patches ψ_1 and ψ_2, both of size $S \times S$, point $\mathbf{x_1}$, $\mathbf{x_2}$ in the coordinate system of ψ_1 and ψ_2 respectively, is now defined by

$$\tau(\psi_1, \psi_2; \mathbf{x}_1, \mathbf{x}_2) = \left\{ \begin{array}{ll} 1 & \text{if } \psi_1(\mathbf{x}_1) < \psi_2(\mathbf{x}_2) \\ 0 & \text{otherwise} \end{array} \right\} \tag{4.3}$$

where $\psi_1(\mathbf{x}_1)$ and $\psi_2(\mathbf{x}_2)$ are the smoothed pixel intensity of patch ψ_1 and ψ_2 at coordinate $\mathbf{x}_1 = (x_1, y_1)^T$ and $\mathbf{x}_2 = (x_2, y_2)^T$ respectively. Next, a n_d-dimensional set of $(\mathbf{x}_1, \mathbf{x}_2)$-location point pairs is chosen. The choice of location point pairs is for now arbitrary and is further defined in Section 4.3.2. To form a n_d-dimensional binary string relating the two patches,

$$f_{n_d}(\psi_1, \psi_2) = \sum_{1 \leq j \leq n_d}^{N} 2^{j-1} \tau(\psi_1, \psi_2; \mathbf{x}_1^j, \mathbf{x}_2^j) \tag{4.4}$$

is evaluated.

This builds a unique descriptor for a set of point pairs. I use the binary string $f_{n_d}(\psi_1, \psi_2)$ as an efficient descriptor between two anchor points and for a feature p_i the descriptor becomes $d_i = f_{n_d}(\psi_1, \psi_2)$. Like in the original paper [16] I use $n_d = 128, 256, 512$ test pairs, which I call I-BRIEF-k with $k = \frac{n_d}{8}$ and therefore name the descriptor by its size in bytes. To create such a descriptor some design choices need to be made

1. choice of smoothing kernel,
2. how to cope with rotation between patches,
3. the geometric ordering of the image patches,
4. concerning endpoint stability.

I will discuss these in detail in the subsequent chapters. To find the best sub-components of the I-BRIEF for the purpose, I test them in video sequences displaying unconstrained motion over 500 frames with different plain textures, related by a homography. The approach is rated with the mean precision, which relates the correct matches with all established matches and gives the probability of a randomly selected match being correct. The dataset and metrics employed here are further shown in section 6.2.2. For stability reasons, the edges that serve as input to I-BRIEF are filtered, to retain only those with a high saliency. The edges are re-projected via a homography for each frame, thus the end points to the I-BRIEF stay the same over the whole sequence, only the image is perturbed by motion blur and projective transformations.

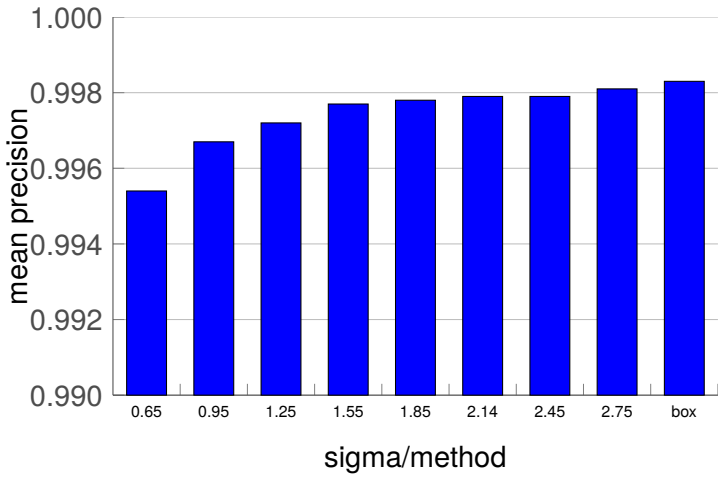


Figure 4.2.: Mean precision of continuous video images for different smoothing kernels ranging from $[0.65, 2.75]$ with a window size of 9 and a box-filter with a window size of 9.

4.3.2. Inter-Point-BRIEF Descriptor Design

Smoothing Kernel

The tests in Eq. 4.3 takes only intensity comparisons between single pixels into account. Since measuring single pixels against each other is very noise sensitive, pre-smoothing is crucial to the system, which increases the stability and repeatability [16]. Since the intensity tests can be seen as evaluating the sign of the intensity difference between two points, it accentuates high frequencies and hence amplifies noise [111]. Fig. 4.2 shows the relation between several Gaussian kernels with different variances ranging from 0.65 to 2.75 and the same using a box-filtering technique. These results reflect the findings in [16, 15]. The box filtering using integral images provides at least the same performance by lower computational cost, therefore I will use it for all following experiments.

Rotation

For point based features rotation needs to be estimated roughly. Contrary to that, edge-based features inherently have a orientation. To align the patch ψ_i to the edge, one can pre-rotate the patch accordingly. This step makes the descriptor rotation invariant. I use the angle between the anchor point and the centroid $\overline{\mathbf{x}}_{AP}$ of the next c points on the curve that lie between the starting point and the end point. Setting $c = \frac{S}{2}$ aligns the patch with the

line segment inside of the patch for any smooth curve. The angles for the rotation of the image patch can now be defined by the vector

$$\mathbf{d}_{AP} = \mathbf{x}_{AP} - \overline{\mathbf{x}}_{AP}, \tag{4.5}$$

where $\mathbf{d}_{AP}$ is the direction of the patch and $\mathbf{x}_{AP}$ is the anchor point. With this directional vector $\mathbf{d}_{AP}$ one can define the rotation of a patch by the angle θ_{AP} between $\mathbf{d}_{AP}$ and the x-axis. A sample point $\mathbf{x}$ of a patch ψ is now rotated, using

$$\mathbf{x}' = \mathrm{R}_{\theta_{\mathrm{AP}}}\mathbf{x}, \tag{4.6}$$

where

$$\mathrm{R}_{\theta_{\mathrm{AP}}} = \begin{pmatrix} \cos(\theta_{AP}) & -\sin(\theta_{AP}) \\ \sin(\theta_{AP}) & \cos(\theta_{AP}) \end{pmatrix} \tag{4.7}$$

is the corresponding rotation matrix. Fig. 4.3 shows the rotation scheme of the image patch, according to the centroid.

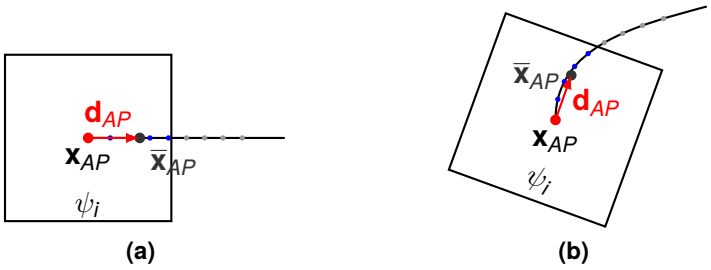


Figure 4.3.: (a) shows the upright I-BRIEF. In (b) a possible rotation of a single patch is displayed.

Spatial Distribution of Test Pairs

To build a string of n_d binary intensity tests between two patches, the set of n_d test pairs has to be selected properly. The point $\mathbf{x_1}$ is defined to be always within the patch ψ_1 and $\mathbf{x_2}$ is taken from ψ_2. The approach used differs slighty from the original BRIEF, as since an edge splits two regions, pixels, that fall within a certain distance of the edge are not considered. In BRIEF the immediate area around the key point is important opposed to edges where the properties of the surroundings are of major interest. Consider the intensities of pixels along an edge, evaluating the sign of the derivative between two edge pixels may not be stable and might result in flips in the descriptor in different images. Therefore I employ a

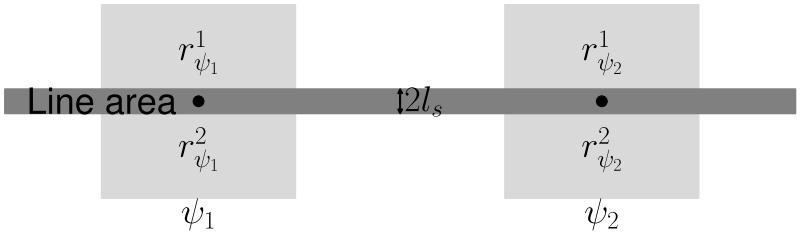


Figure 4.4.: Basic layout of the upright descriptor with two patches, each having two regions, and line area where no intensity tests are conducted.

suppression area with length l_s on each side of the edge, where no tests can be located, much like it is done in [8]. To make sure to not comparing edge pixels with each other, the area delimitation is set to the size of the Gaussian kernel used for edge detection. The distribution scheme is now defined locally for both regions of each patch $r^j_{\psi_i}$ with $i \in \{1,2\}$ and $j \in \{1,2\}$.

The following distributions assume a straight line that lies on the x-axis. Let it be stated that no rotation of the patches is considered here. In this case, one can define region $r^1_{\psi_i}$ to be the region with positive y values and therefore $r^2_{\psi_i}$ with negative y values. In Fig. 4.4 this setup is shown. The geometric ordering of the (x, y) locations is defined by the following distributions.

- ***G I –*** Uniformly distributed $\mathcal{U}(a,b)$, with lower bound a and upper bound b:

$$\begin{aligned}
r^1_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{U}(-\frac{S}{2}, \frac{S}{2}), && i \in \{1,2\} \\
r^1_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{U}(l_s, \frac{S}{2}), && i \in \{1,2\} \\
r^2_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{U}(-\frac{S}{2}, \frac{S}{2}), && i \in \{1,2\} \\
r^2_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{U}(-\frac{S}{2}, -l_s), && i \in \{1,2\}
\end{aligned} \tag{4.8}$$

  where $\sim$ i.i.d. means independent and identically distributed.

- ***G II*** – Normal distributed $\mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2

$$\begin{aligned} r^1_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{N}(\frac{S}{4}, \frac{4S}{25}), && i \in \{1,2\} \\ r^1_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{N}(0, \frac{4S}{25}), && i \in \{1,2\} \\ r^2_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{N}(-\frac{S}{4}, \frac{4S}{25}), && i \in \{1,2\} \\ r^2_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{N}(0, \frac{4S}{25}), && i \in \{1,2\} \end{aligned} \tag{4.9}$$

  If a value lies outside of the patch or within the line area, it is redistributed, choosing $\sigma = \frac{2S}{5}$ to provide a more decentralized distribution.

- ***G III*** – Combination of $\mathcal{U}(a,b)$ and $\mathcal{N}(\mu, \sigma^2)$. y values are normal distributed, whereas x values are uniformly distributed:

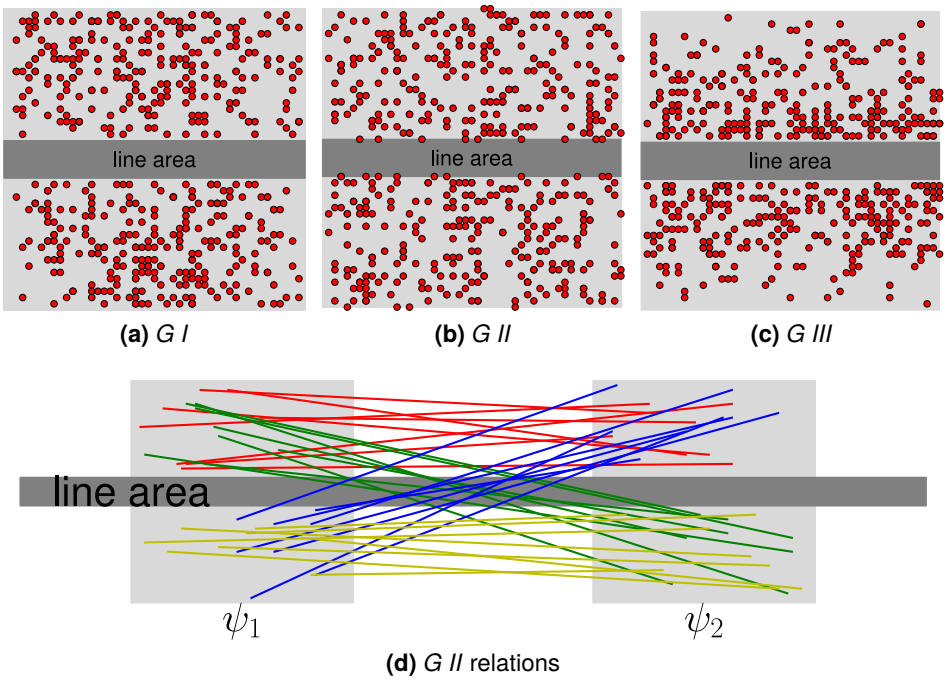
$$\begin{aligned} r^1_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{U}(-\frac{S}{2}, \frac{S}{2}), && i \in \{1,2\} \\ r^1_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{N}(l_s, \frac{S}{100}), && i \in \{1,2\} \\ r^2_{\psi_i}(x_i) &\sim \text{i.i.d. } \mathcal{U}(-\frac{S}{2}, \frac{S}{2}), && i \in \{1,2\} \\ r^2_{\psi_i}(y_i) &\sim \text{i.i.d. } \mathcal{N}(-l_s, \frac{S}{100}), && i \in \{1,2\} \end{aligned} \tag{4.10}$$

  Again, values lying outside the patch or within the line area are redistributed, choosing σ to be $\frac{S}{10}$ to examine how intensity tests close to the line behave.

In Fig. 4.5 the different distributions and the relation between two patches are depicted. To find the best geometric ordering, I compute the mean precision over all five different video sequences for I-BRIEF-$\{16, 32, 64\}$ and plot the results in Fig. 4.6. Distribution *G II* provides slight advantages over both remaining distributions and therefore the distribution *G II* was used for everything introduced in the following.

Endpoint Stability

Since it is well known, that endpoints are not a stable feature for reliable correspondence estimation, the stability of the approach is evaluated in an artificial imperfect situation. The location of endpoints is varied by truncating the curve by a new Gaussian modeled start- and endpoint. A feature p_i^k with the point set $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ is truncated by discarding all



(a) *G I* **(b)** *G II* **(c)** *G III*

(d) *G II* relations

Figure 4.5.: (a), (b) and (c) show the different distributions of the ψ_1 patch with 512 tests (64 byte). (d) depicts the relations between two patches with *G II* distribution, showing 32 (4 byte) test pairs for the sake of clarity. Colors differentiate the regions in which the binary test fall. Region-pair 1-1(red), 1-2(green), 2-1(blue) and 2-2(yellow).

points with a lower index for start point alteration and a higher index for endpoint alteration. Thus the new point set becomes $\{\mathbf{x_a}, \mathbf{x_{a+1}}, \ldots, \mathbf{x_{b-1}}, \mathbf{x_b}\}$, with $a, b \sim i.i.d.\ \mathcal{N}(\mu, \sigma_{ep})$. Fig. 4.7a shows the results for this experiment. It is evident that alteration of the endpoints perturbs the descriptor and especially in non-trivial situations this can lead to disastrous performance, indicated by the high variance for strong variations. Although I-BRIEF seems stable against small endpoint perturbations, one cannot rely on endpoints per se, instead more robust points for inter-point relations are needed.

Therefore anchor points, as discussed in Section 3.2.4 are used. To measure robustness of points I use the ϵ-repeatability criterion [102, 40]. Here, the repeatability r compares the geometric stability of an endpoint or an anchor point in different images. It can be computed by projecting a single point with the according homography into the new frame and measuring the distance between the original point and the projected point. If the Euclidean distance between those points is smaller than some threshold ϵ_{ep}, the point is considered to be successfully ”repeated”. The repeatability is the rate of the total observed endpoints

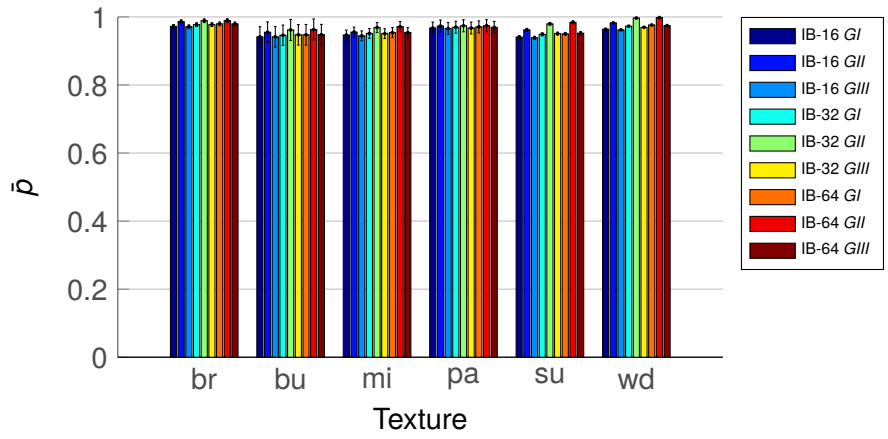


Figure 4.6.: Mean precision and variance in several videos with unconstrained motion, using I-BRIEF-$\{16, 32, 64\}$ descriptor each with distribution $\{GI, GII, GIII\}$ and nearest neighbor matching.

or anchor points in both images, thus $r = 1$ means a perfectly repeated point. In Fig. 4.7b an experiment was conducted, which evaluates the mean repeatability $\bar{r}$ of endpoints and anchor points averaged over all frames in five different videos. I employ the SISeg edge extractor in every frame of a sequence and measure the ϵ_{ep} repeatability of endpoints and anchor points separately. The results suggest that anchor points are more stable than endpoints by means of repeatability.

One can now define for a arbitrary curve s_i with N_{AP} anchor points, in addition to both endpoints, $N_{AP}+1$ sub features as $p_i = \{p_{i_1}, p_{i_2}, \ldots, p_{i_{N_{AP}+1}}\}$, with each subset having its own curve representation s_{i_j} and descriptor d_{i_j}. In the next chapter I will discuss how one can match these sub-features for arbitrary edges or respectively contours.

4.4. Inter-Point-BRIEF Matching

In this chapter I will further explain the matching process for features between two frames. To cope with split and merged edges, I propose an alternative method, in addition to those in section 4.1.3. One can still apply a 1NN matching strategy for video sequences or a NNDR based matching strategy for wide-baseline matching, as will be described here.

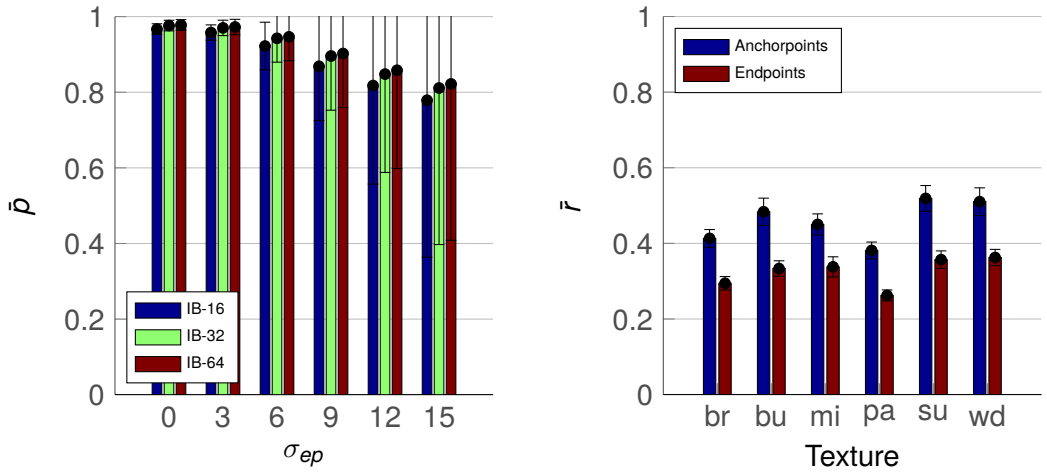


Figure 4.7.: (a) shows I-BRIEF mean precision $\bar{p}$ vs. endpoint stability varying endpoints with a Gaussian distribution $\mathcal{N}(0, \sigma_{ep})$ for I-BRIEF-$\{16, 32, 64\}$. (b) displays the mean repeatability and variance in several videos with unconstrained motion

1NN Matching Strategy

One can use the 1NN matching strategy based on entire edges by finding matches, that are only described by the relation between the endpoints. This matching thus happens at *feature level*. For each feature in $\mathcal{P}^k$ I seek the first nearest neighbor in the feature-set $\mathcal{P}^{k-1}$ within a spatially reduced neighborhood by using a gated search region r_{sr}. The search region, see Fig. 4.8b, is based on the minimal area rectangle $r_{minArea}$ spanned by s_i of feature $p_i \in \mathcal{P}^k$. I neglect the dependency of a search region to s_i in the notation, because it should be clear from the context, that it is centered around some feature. some offset l_{sr} is added to the width and height of r_{sr}, such that if s_i consists of a single point, it is centered around this point. Thus each feature is matched to a nearest feature in feature space with some spatial restrictions. In video sequences this might work, as shown in Section 6.2.2, but matching endpoints will most certainly fail with increasing baseline.

NNDR Matching Strategy

The NNDR matching strategy is therefore applied to wide-baseline scenarios, which were introduced in 4.2. Due to the endpoint instability and varying content in the case of wide-baseline, I split edges at high curvature points into shorter and smooth curves and employ

all subsequent description and matching steps on these features. This matching happens at *sub-feature level*. Before describing each (sub-) feature I filter out edge segments, which are shorter than some threshold, as it is applied in wide-baseline approaches [123]. Matching is now achieved between all possible remaining features in $\mathcal{P}^k$ and $\mathcal{P}^{k-1}$, either described by the relations between endpoints or anchor points or a combination of both.

I-BRIEF Matching Strategy

The strategy is based on edges which are split at anchor points and hence, a feature is subdivided into several sub-features, but a match is established on the entire feature $\mathbf{p}_i \in \mathcal{P}^k$. The underlying matching is based on the 1NN matching strategy with the gated area r_{sr} around some feature $\mathbf{p}_i$, but I use mode statistic in the sub-feature level to attain a match based on the feature level. The mode of a discrete distribution is the most frequent value. For example, in a set $t = \{1,1,2,2,2,3,5,7\}$, I define the mode $m_0(t) = 2$ as the most frequent with $|m_0(t)| = 3$ occurrences and $m_1(t) = 1$ as the second most frequent.

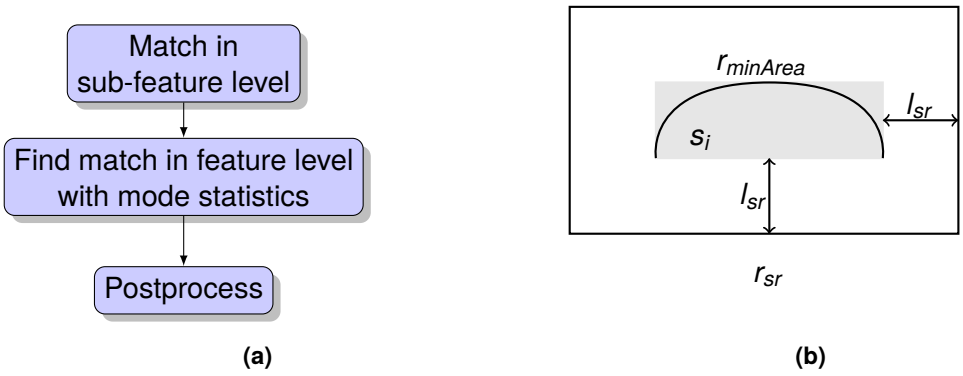


Figure 4.8.: (a) Outline of proposed matching scheme for a single feature. (b) shows the search region for an arbitrary curve.

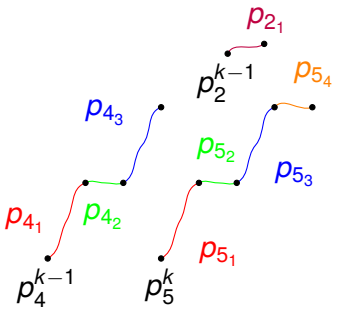
The outline of this approach is depicted in Fig. 4.8a. For the matching, features that are split at anchor points into a subset are utilized. At first for sub-feature $p_{i_j}^k$ with i as feature ID and j as sub-feature ID, the corresponding sub-feature $p_{l_m}^{k-1}$ with feature ID l and sub-feature ID m are found. The distance of sub-features between descriptors $d_{i_j}^k$ and $d_{l_m}^{k-1}$ can be efficiently calculated with the hamming distance, described in Section 4.1.1.

The sub-feature, that minimizes the hamming distance inside the search region r_{sr} around the feature p_i, is taken as the sub-feature correspondence. For the sake of readability I neglect the dependency on frame k and $k-1$ and only use p_{i_j} and p_{l_m}. Thus each feature

p_i will have several matches according to the sub-feature matches, which will be referred to as $\mu_i = \{\mu_i^1, \mu_i^2, ..., \mu_i^{N_{AP}+1}\}$ where $\mu_i^1, \mu_i^2, ...$ represent the according feature ID I of the feature p_{I_m} . I now apply mode statistics on μ_i to determine a match. Therefore I find the most $m_0(\mu_i)$ and second most $m_1(\mu_i)$ frequent IDs occurring $|m_0(\mu_i)|$ and $|m_1(\mu_i)|$ times, with $|m_0(\mu_i)| \geq |m_1(\mu_i)|$. The mean distance $d(m_0(\mu_i))$ and $d(m_1(\mu_i))$ of all sub-feature matches, which contributed a $m_0(\mu_i)$ or $m_1(\mu_i)$ ID to μ_i is calculated. To determine a feature match I consider both $m_0(\mu_i)$ and $m_1(\mu_i)$ sub-feature matches. The correspondence scheme can be formulated as follows:

- If $|m_1(\mu_i)| = 0$ I instantly set $m_0(\mu_i)$ as a match
- If $\frac{d(m_0(\mu_i))}{d(m_1(\mu_i))} < \tau$ I use $m_0(\mu_i))$ as a match
- If $|(m_1(\mu_i)| \geq r \times |m_0(\mu_i)|$ and $d(m_1(\mu_i)) < d(m_0(\mu_i))$ I use $m_1(\mu_i)$ as match
- If nothing fits, reject match

I exemplify the proposed matching scheme with an illustrative example in Fig. 4.9. This example consists of two features p_2^{k-1}, p_4^{k-1} in frame $k-1$ and in frame k only p_5^k. The goal is to establish a match for p_5^k. The example results in $m_0(\mu_i) = 4$ and $m_1(\mu_i) = 2$, with distances $d(m_0(\mu_i)) = \frac{10+11+3}{3} = 8$ and $d(m_1(\mu_i)) = 30$. Going through the correspondence scheme, a match between feature p_5^k and p_4^{k-1} can be found.



(a) Illustrative example

Feature	μ_5^1	μ_5^2	μ_5^3	μ_5^4
p_5^k	4(10)	4(11)	4(3)	2(30)

(b) Matching result

Figure 4.9.: (a) displays an example of the matching approach, where similar colors depict actual correspondence. Along with sub-feature matches and distances in brackets in (b).

With this matching scheme it would inherently allow multiple entries, which only makes sense if the edges split at some point. To additionally filter matches that do not result from split edges the matches are post processed. The distance between each start and endpoint of the features is measured and only multiple entries are kept, if the closest distance is within 3 pixels.

Chapter 5.

2D to 3D Conversion

This chapter first explains the most common depth cues and reviews the current state-of-the-art approaches in generating depth information from either single images or image sequences. Then a lightweight adaptive fusion scheme is introduced, which utilizes the delivered by the *SISeg* algorithm. Additionally, a formal scheme to integrate motion information provided by the Inter-Point-BRIEF is given.

5.1. Related Work on Depth Map Generation

As stated in section 1.2 of the introduction chapter, 2D-3D conversion can be classified in manual, semi-automatic and fully automatic approaches, ranging from low-quality real-time approaches to highest quality approaches demanding high computational effort plus often human interaction. As the approach introduced in this chapter, this related work section concentrates on examples for semi-automatic and fully automatic concepts, focusing on the computational complexities of the various methods.

The *Depth Director* [124] denotes a system, that allows for semi-automatic segmentation, rotoscoping, and depth assignment. The approach supports automatization at all steps, but needs manual interaction finalization and for starting points. For an initial depth assignment, it utilizes a set of depth heuristics and complex depth templates and Structure from Motion, if applicable - which is generally not always the case, as also mentioned in the introduction.

In recent years, many semi-automatic and offline approaches, which prioritize high quality over computational speed, rely on machine learning techniques.

In [97], the authors created a database of images of outdoor scenes including depth and used supervised learning to model depth as a function of the image. They applied a Markov

Random Field and a multi-scale feature vector including both local and global image features. The feature vector is based on the cues of texture gradients and color haze.

In [53], the authors assign depths learned from a RGBD dataset via a methods called non-parametric sampling and depth transfer. This includes finding candidate matches in the database using GIST features, and warping the depth via SIFT-Flow features, followed by a global optimization step. In case of image sequences, they can include temporal information for increasing accuracy. The approach is fully automatic and outperforms other approaches on benchmarks, however, it has a certain binding to the database. It takes one minute per 640×480 image to process on a 3.2 GHz Quad Core using parallel implementation. The authors extended their approach to videos, calling it *depth transfer* [52]. In [90] the authors use machine learning in a random forest based approach to fuse various depth cues.

In the field of automatic approaches, there is also a strong connection between the computational complexity and the cue that is used. Some approaches using various cues will be shortly introduced in the following. For an introduction of the main monocular cues, please also refer to section 1.1.1.

Depending on the imagery, *depth from blur, or defocus* can result in very sound depth maps. Yet, it is rather complex and thus computationally expensive to obtain the blur amount properly. Further, a number of additional steps are necessary to get to a final result, like transforming from sparse to dense image information, moving from relative To estimate the blur, various methods are utilized, like Gaussian Kernels plus gradient ratios [135], inverse diffusion[79], or a focal stack method [44].

An example of a *texture* based approach is given in [70]. Such methods also demand significant computation time, as they prerequisite the description of texture features, the extraction of the textured surfaces, plus affine transformations to estimate the orientation of these surfaces, which then leads to the depth map estimate.

Methods relying on the *depth from shading* cue, utilize a Lambertian reflection model to establish a relation between the reflections of light on surfaces and the orientation of these surfaces. Such estimates thus try to analyze the image brightness, which can be conducted via e.g. minimization [134], propagation [11], local [62], or linear [87] approaches. All of those methods imply very high computational complexity.

The major task to obtain *depth from linear perspective*, is to correctly detect the vanishing points by correctly grouping and classifying the detected lines in the image. In [133], the authors therefore use a RANSAC approach, while [110] apply the graph cut algoririthm for grouping. Subsequently, methods to get from sparse to dense maps need to be applied.

Depending on the approaches, this results in still increased computational effort, yet not as high as the cues mentioned above.

Cues, which can be calculated with considerably low complexity, are for example the *color based statistical patterns*, as well as approaches based on gradients.

Contrary to the approaches mentioned above, the authors of e.g. [115] strongly focus on computational performance. They combine an edge histogram-based global depth gradient with a local color and luminance-based refinement. With these low-complexity cues they achieve 30fps on 1080p video including the DIBR calculation, making heavy use of multicore CPU and GPU optimization.

In [49], the authors combine depth from motion cue with depth from geometry. For computational efficiency they obtain a moving segmentation utilizing the H.264 flow information and create a Gaussian Mixture Model for modeling the background, which is the masked. The vanishing lines are obtained via Hough Transform and a depth gradient is assigned to only one major vanishing point. The fusion module simply takes the depth from motion where motion was classified and the geometry-depth map else. The authors show a set of result depth maps, but don't provide any evaluation results, or computation times.

In [129], the authors combine global depth gradients with local depth refinements to achieve a real-time conversion for videos.

5.2. A Light-Weight Adaptive Depth Map Generation

To achieve a speed-efficient 2D to 3D conversion that achieves a certain quality, there is no single technique or respectively cue to rely on exclusively. Instead, an adaptive combination of known techniques, which are again stripped down to increase computational efficiency, is the key factor. The following listing provides an overview on the four basic depth modules that are the input to the fusion module, which are then detailed in the succeeding sections.

- Texture based depth, section 5.2.1

  This module lends components of an edge detector to calculate gradients hardened against noise. The gradients are further accumulated in small blocks used to estimate surface orientation which is then used by a propagation algorithm to get a depth estimate. Compared to classical texture-based approaches, the method introduced here is rather an approximation of depth from texture.

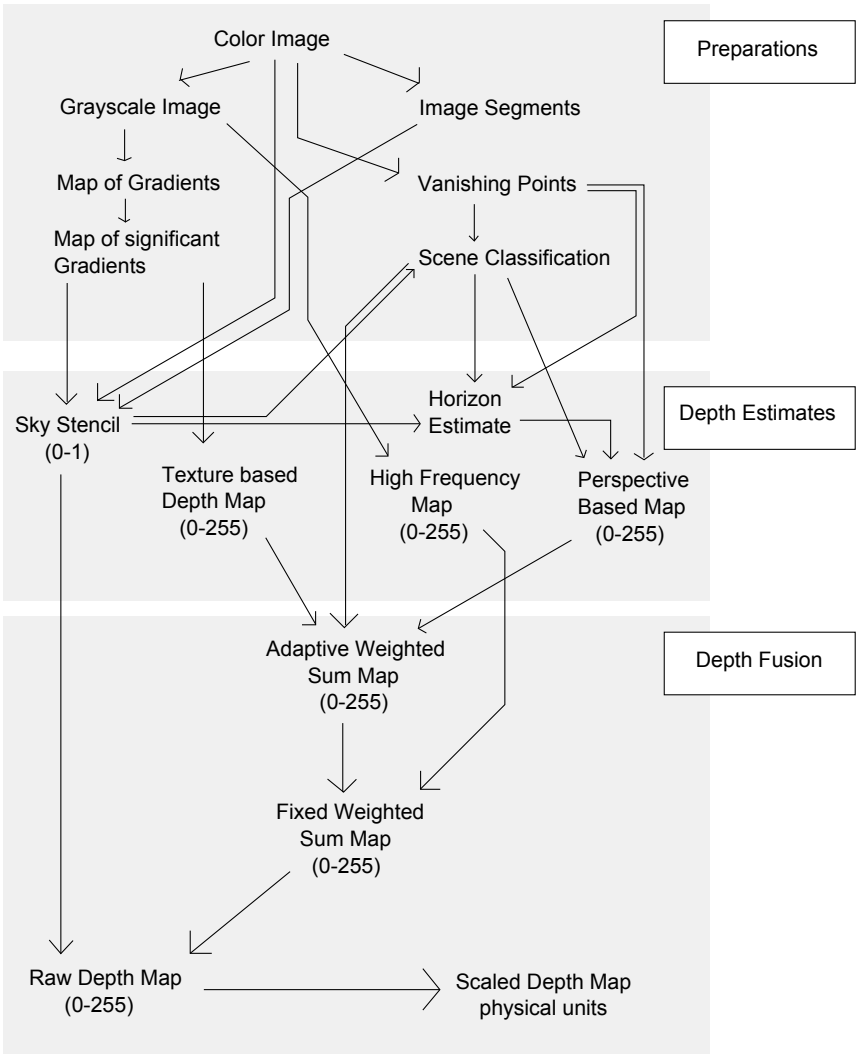


Figure 5.1.: Overview and computational flow of the light-weight adaptive depth map generation module.

- Linear Perspective based depth, sections 5.2.6, 5.2.7 and 5.2.8

  This module first calculates an estimation for a vanishing point. The presence and position of a dominant vanishing point is further used for a general classification of the scene, and derived from this, the choice of scene-based global depth map types.

- High frequent detail based depth, section 5.2.5

  High frequency components of the input image's intensity channel serve as a local refinement input, which is complementary to the global depth map. Fusion of both can enhanced image details and thus the quality.

- Segment wise assignment of sky areas, section 5.2.3

  Utilizing the segmentation outcome, this module decides for each segment whether or not it belongs to the sky and creates a stencil map from this information. This color based depth module is to obtain and provides a significant quality boost for outdoor images.

Section 5.2.9 details the adaptive fusion to join the results of the four modules. The workflow of the overall procedure is shown in figure 5.1.

5.2.1. Construction of a Texture Based Depth Map

A texture based depth map can be approximated using the absolute value of the intensity gradient's vertical component can be tackled, if the following idealized assumptions are considered.

In the most common depth scenario, the ramp, the captured scene has a ground which is close at the bottom and far away at the top of the image. If there are objects in the scene, they are usually considered upright, i.e. each object has a more or less 'single' depth value. The borders of these objects, and consequently the image gradients, also coincide often with color gradients in the image and coevally with depth discontinuities. This idea can be expressed as

$$z_{y,x} = z_0 + a \sum_{k=y_0}^{y} |\nabla G_{y,x}|, \tag{5.1}$$

where $G_{y,x}$ is the intensity of the light at the position $(x, y)^T$ and $z_{y,x}$ the depth at this point. z_0 is a preset distance to the closest point and a is a tuning parameter.

Under some basic assumptions, this concept similarly holds for textured surfaces viewed in different orientations.

Again one can assume the basic shape of a ramp, many scenes feature an elongated horizontal surface in the lower part of the image, e.g. water, streets, grasslands, or floors. Furthermore, the optical axis of the camera is assumed to be horizontally aligned and there are no areas where the ground elevation decreases with increasing distance to the camera. Moreover, the image shall be uniformly illuminated and void of specularity or shadows.

The general idea is that under these assumptions, the surface texture looks different when viewed from different angles. In [70] an approach is presented where local surface orienta-

tion is obtained by comparing Fourier-spectra of the same texture at different locations and determining shortening and stretching of the texture pattern. However, if as simplification the surface color intensity has a similar pattern all over the image, the shortening can be determined by integrating the absolute magnitude of the intensity gradient in vertical direction, leading to a higher frequency in intensity change, if the textured surface lies in a more horizontal angle and is thus viewed from a steeper angle, as displayed in figure 5.2.

Figure 5.2.: The same surface pattern viewed from two different angles. Features are closer to each other when viewed from a steeper angle.

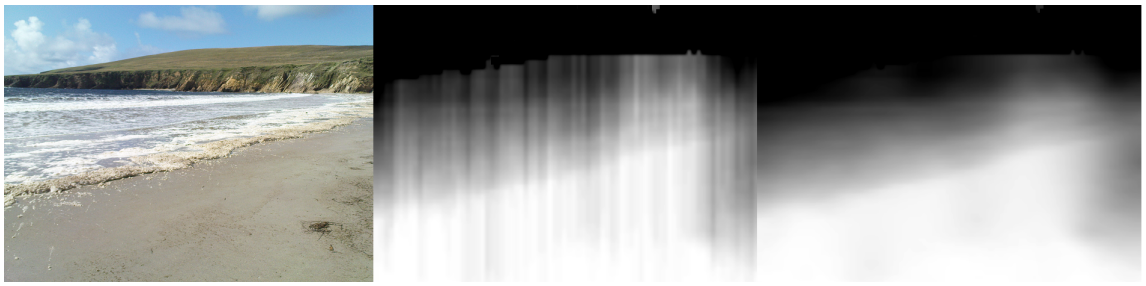


Figure 5.3.: Texture based depth map of a beach scene. There are visible vertical stripes caused by error propagation. Sky removed by the sky detector. Left to right: original, unfiltered, filtered.

However, real world patterns are rarely as even as shown in figure 5.2. Thus the blockwise vertical integration gradients can lead to vertical bars in the depth map with their intensity increasing to the upper end of the image, as displayed in the middle subfigure of 5.3.

To obtain a smooth depth map, the intermediate result of the texture map has to be post-processed. As a standard approach, Gaussian smoothing was applied in the right subfigure of 5.3. Yet, a joint bilateral filter night be a better suited choice, as it does not wash out the depth horizontally over object borders. The filtering step is detailed in section 5.2.2.

5.2.2. Improvement by Post Processing Filter

The smoothing operation applied to the texture image shown in the right of figure 5.3 is a simple Gaussian filter. The gaussian filter holds the drawback that it blurs depth discontinuities. Thus it shall be evaluated in this section, if a joint bilateral filter [114] is able to improve the results. A Gaussian filter is defined as

$$h(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi) c(\xi, x) \mathrm{d}\xi, \tag{5.2}$$

with

$$c(\xi, x) = \frac{1}{2\pi\sigma^2} e^{-\frac{\|x-\xi\|^2}{2\sigma^2}}, \tag{5.3}$$

where $f(x)$ is the input image, $h(x)$ the output image, $c(\xi, x)$ is a measure of geometric closeness between x and the expectation value ξ, defined as two-dimensional Gaussian curve with standard deviation σ.

In the bilateral filter, each pixel is assigned with a weighted sum of neighboring pixels, so in contrast to the Gaussian filter, each weight is not only determined by the relative distance to the target pixel, but also by the similarity of pixel values. This way, a smoothing that preserves contour information can be achieved, because pixels on opposing sides of an edge do not influence each other. The bilateral filter is defined as

$$h(x) = k^{-1}(x) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi) c(\xi, x) s(f(\xi), f(x)) \mathrm{d}\xi, \tag{5.4}$$

with

$$k(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, x) s(f(\xi), f(x)) \mathrm{d}\xi, \tag{5.5}$$

where additionally $s(f(\xi), f(x))$ is called photometric closeness and $k(x)$ is a normalization term.

While the joint bilateral filter itself is applied on the depth map, the input for $s(f(\xi), f(x))$ descends from the brightness information of the original image, to govern the weighting. Thus, edges that are present in the distinct image are preserved in the depth image. In [135] this scheme is implemented to fill a sparse depth map in a way, that locates depth discontinuities near edges and smooths out areas.

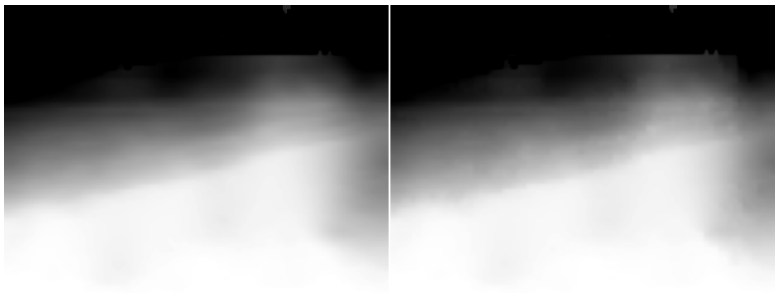


Figure 5.4.: Image filtered using a Gaussian filter (left) and a bilateral filter (right). Note, that the contours on the right are less blurry than on the left.

In this implementation, the weighting of neighboring pixels in $s(f(\xi), f(x))$ is based on the output of the segmentation, so borders of segments are retained. Experiments confirmed, that while the plain Gaussian filter produces reasonable smoothing output when applied a wide range of texture depth images, it weakens the result in some scenes due to the smoothing. Here, the contribution of the joint bilateral filter compensates this weakness and thus justifies its usage.

5.2.3. Sky Color Detector

For depth images created from outdoor scenes, sky detection is an important task, as image regions identified as sky need to be assigned an infinite depth, being infinitely far away from the viewer. Clear blue sky has a high probability of being blue and bright. This case thus can be captured using a simple color heuristic.

Yet, there are more ambiguous scenarios. One is the view of the rising sun, where sky colors range from orange and red over purple to blue, while the sun itself can range from a bright white when saturating the image sensor to more saturated reddish, orange or yellow colors. Another case is a cloudy sky, in which case the clouds can maintain all shades of gray and appear colored in any hue the atmosphere or the sun provide by translucence and reflection. In a naive approach, grey sky can also easily be mistakenly detected, for example in an indoor scene that shows grey concrete walls. A mere color detection over all these ranges would thus logically lead to manifold false positives. Effectively the only color that would never appear in a sky segment is green.

A color normalization as detailed in [118] is one solution to tackle the cloud problem.

If several criteria are applied, one can construct a heuristic that yields good decisions in

many situations. In this implementation, five distinct normalized indicators are calculated on each segmented region, which are introduced in the following:

- I_{bright}: The first one is a brightness indicator which uses the average value in the HSV color space [38] of all pixels. If the average value is is below a given threshold, the indicator is 0 and then increases linearly to 1.0 at the brightest value. Having a value range from 0 ... 255, the minimum brightness threshold is set to 100.
- I_{smooth}: The second indicator is a measure of smoothness within a patch, as usually clouds do not hold sharp edges. Therefore the gradients of each patch are accumulated and thresholded, similarly to the brightness indicator. If the sum of gradients is above a threshold, the indicator is set to 0. If the sum is 0, the indicator is set to 1.0 and decreases linear to 0 at the threshold. The edge detector generates statistics of the gradients, so one can define a significance threshold based on the expectation value and variance of the gradient magnitude.
- $I_{gradient}$: The third indicator is supplementary to the smoothness indicator, and defined by a simple heuristic based on the significance of the gradient magnitudes. A segment is divided into 8×8 pixel blocks, each of which is thresholded to judge the gradient magnitude significance. Then for each segment, the ratio of marked to non-marked pixels. If less than a third of the pixel blocks is marked, the gradient indicator is set to 0, for more than two thirds 1.0. In between, it rises linearly.
- $I_{elevated}$: The next indicator operates on the vertical position of the segment in the image. If the center of area is located in the lowest third of the image, the indicator assumes 0, in the upper third 1.0, again using linear interpolation in between. This indicator accounts for the assumption, that the sky in a (regular) image is supposed to be in the upper of the image.
- I_{blue}: Finally, the most basic indicator is the color indicator. In principle, an arbitrarily complex probability density function based on the color distribution of the whole scene would be required to capture all aspects of the color problem.

  Yet, in combination with the other four indicators, the color heuristic can be utilized in a more simplified manner. Working in HSV-color space, the color indicator in this implementation generally assumes 0 if the hue is outside the range of 170 to 260 degrees. These hue values relates to a range from cyan to blue and captures sky well in the most probable kinds of outdoor scenes, when the sun is not close to the horizon. The indicator is set to 0.5, if the saturation value is above 10. This is necessary, as the hue value becomes very sensitive to image noise for low saturation or low value, as

the color-space conversion has singularities. If saturation rises above 20 % and value above 40 %, the indicator is assigned 1.0. Linear progression is difficult to justify as it would capture brightness and not how “blueish” the color appears. However, saturation and value are also rather crude, as the sky parameters can vary strongly depend on weather, daytime and exposure. Thus no special tone can be defined as a reference for a sky color in order to obtain a simple univariate indicator law. Basically, the ranges are set in a way, that in combination with all indicators, a proper trade-off between true negative and false positive sky detection is achieved.

To decide if a patch is considered to be sky or not, a weighted average of the indicators is calculated. If the sum is greater than 0.5, a patch is considered to show sky. For the weighted average, all indicators are weighted equally, except for the blue indicator, which gets a double weight,

$$\text{segment is } \begin{cases} \text{sky} & \textit{if } \frac{1}{6}(I_{bright} + I_{smooth} + I_{gradient} + I_{elevated} + 2 \cdot I_{blue}) > 0.5, \\ \text{no sky} & \textit{else}. \end{cases} \tag{5.6}$$

5.2.4. Reduction of Segment Interference

As the texture approach is based on a per column accumulation of depth increments, it produces visible artifacts in scenes where an object is located in front of an erect background surface. In such a scene, the ramp assumption does not hold.

The object border between background and the object’s top creates a discontinuity of the estimation, that "blends" into the background’s estimation and deforms it. If one considers the depth assigned to the background’s bottom-most pixels as correct, one views depth information, that is transferred into a segment by the vertically cumulative formula increasingly critically, as the transfer location is more elevated with respect to the "correct" base pixel. With a given segment height of h and base y-coordinate y_B, one can define a blend-in value b on a pixel with coordinates x and y as

$$b_{x,y} = \begin{cases} 0 & \text{if } S(x,y) = S(x,y+1), \\ \frac{y - y_B}{h} & \text{else}, \end{cases} \tag{5.7}$$

with $S(x,y)$ being the segment index of the pixel. Note that $y+1$ refers to the pixel below the current pixel. Applied on the entire image yields to a accumulated blend-image B holding

values between 0 and 1, which is a model for how much each pixel is influenced by the depth at a border that is not at the bottom-most elevation of each segment. This information is utilized to 'deform' the pixel's depth value in the direction of the segment's estimated depth value. The estimation d_{est} of segment i, is calculated by

$$d_{\text{est},i} = \frac{1}{\sum_{(x,y)\in S_i} b_{x,y}} \sum_{(x,y)\in S_i} d_{x,y}\, b_{x,y}, \tag{5.8}$$

where S_i is the set of all pixels in segment i and $d_{x,y}$ is the initial, blend affected depth estimation at location x, y. The deformation is formulated as

$$d_{unblend,x,y} = c\, B_{x,y} d_{est,S(x,y)} + (1 - c\, B_{x,y})\, d_{x,y}, \tag{5.9}$$

where c is an extra factor to control the amount of unblending.

Figure 5.5 shows the blending issue at an upright wall with an object in front of it (left), the occuring blending error that evolves from it (middle), and the correction by the unblending step (right).

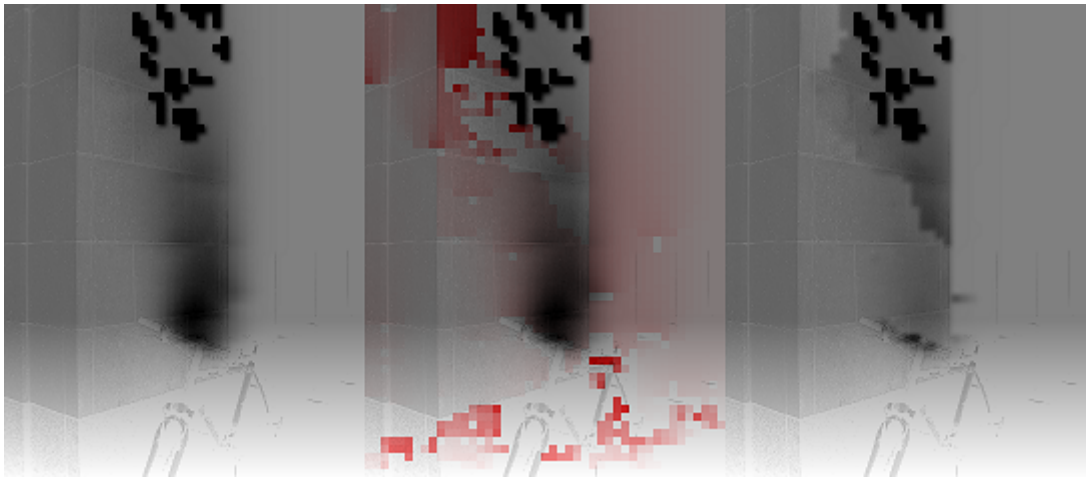


Figure 5.5.: Left to right: uncorrected depth map, uncorrected depth with blend amount overlayed red and corrected version.

5.2.5. Retrieval of High Frequency Depth Features

To add local details to the depth image, a difference of Gaussians method, described in [71] is applied. It isolates an image with higher frequencies, which are located strongly at

high contrast edges. As a normalization step, the result is divided by the brightness value, which effectively shifts the derivative's peak value closer to the dark zones.

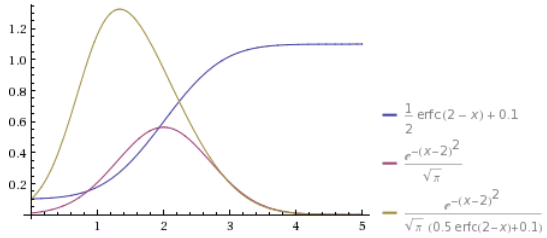


Figure 5.6.: Plot of brightness at a blurred step edge, the gradient and gradient divided by brightness.

In cases, when for example objects edges occlude parts of the sky, this method greatly improves the depth estimation. This step adds a large amount of detail to borders in such cases because of a sound contrast between sky and the foreground objects.

5.2.6. Computation of Vanishing Points

The implementation used as vanishing point detector is based on [133] and [110]. It uses a RANSAC based method to find three mutually orthogonal vanishing points from all detected lines while excluding some lines as noise. Besides the positions of each vanishing point, their strength is also evaluated, measured as the number of detected lines associated with the vanishing point.

5.2.7. Scene Classification by Vanishing Point Constellation

Once the vanishing points are known, one can use their position and number of associated edges to get a basic understanding of the dominant scene geometry. Analogue to the decision whether a segment is part of the sky, five indicators, which characterize the scene are calculated. For the final choice of global basic scene patterns, the Indicators are additionally combined by some rules introduced below. These rules compare the indicators with each other and further improve the estimates by some logical reasoning.

- $I_{muchSky,raw}$: The first indicator takes into account how much of the image area was classified as sky by the algorithm in section 5.2.3. From the no sky condition up to where the sky covers 30% of the image, the indicator grows linearly from 0 to 1.0 and stays there if there is more sky.

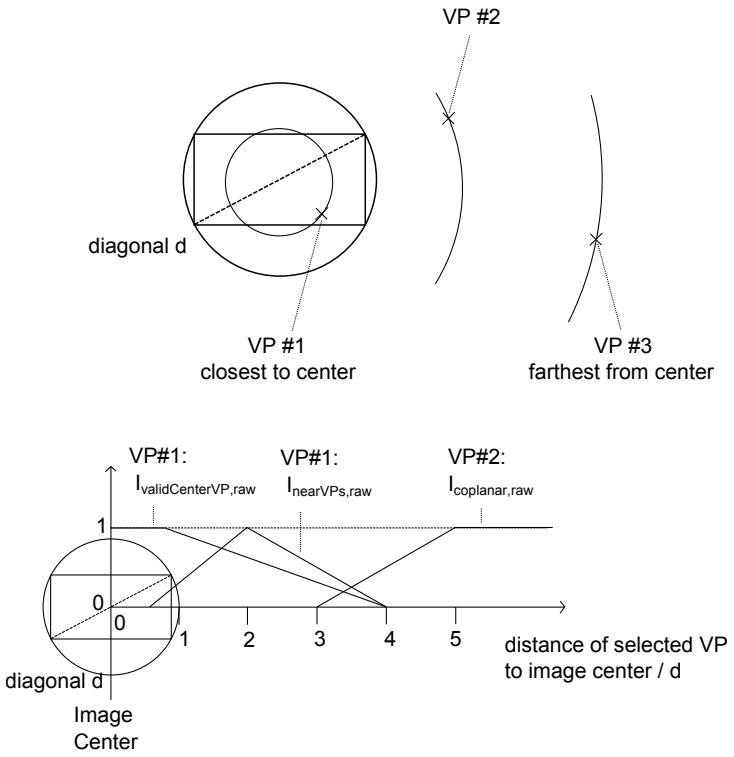


Figure 5.7.: Vanishing points are sorted by the distance to the image center.

- $I_{validCenterVP,raw}$: This indicator captures the existence of a central vanishing point. If the vanishing point closest to the image center has no more than four associated edges, the indicator is assigned to 0. Despite this, if it is positioned in a range between four times the image diagonal and 80% the diagonal, the indicator is a linear interpolation between 1.0 and 0, with zero at the outer rim and 1.0 to the middle. Inside of the circle at 80% of the image diagonal, it is set to 1.0.
- $I_{coplanar,raw}$: This indicator detects a scenario consisting of a coplanar backrgound, i.e. an upright surface with a more or less single depth value. This scenario occurs, if two vanishing points far away from the image center are detected. It has a linear increase from 0 to 1.0, if the second most far away vanishing point is between three and five times of the image diagonal from the center, with being 0 inside and 1.0 outside this area. On some image databases this indicator should be suppressed, if it contains no scenes fitting the coplanar category.
- $I_{noVPs,raw}$: Another indicator describes scenes, where no vanishing points could be detected. It is assigned to 0.01 if the outermost or the second outermost have at least

four associated edges or the most central vanishing point has at least five associated edges. Otherwise, it is calculated by

$$I_{noVPs,raw} = 1 - \frac{0.99}{13}(N_{VP1\ edges} + N_{VP2\ edges} + N_{VP3\ edges}), \tag{5.10}$$

where 0.99 compensates for the residual weight of 0.01 and the denominator normalizing the sum of the three counts of associated edges, the Ns are numbers of lines associated to a vanishing point.

- $I_{nearVPs,raw}$: The last indicator holds in scenes, where there are two similarly distant vanishing points that are relatively close to the center. If the detected vanishing points are orthogonal, or at least close to, it is sufficient to know that there is no central vanishing point and the most central vanishing point is not too far away. So the indicator is defined to be set to 1.0, if the distance of the most central vanishing point is two times the image diagonal, while it decreases at both sides to 0 at 60% and four times the diagonal, respectively.

As mentioned above, additional rules for combining the given indicators can further improve the selection of the base scenario, respectively the scene classification. These rules are displayed in the following.

The first rule improves the estimation of coplanar indicators,

$$I_{coplanar,rawB} = I_{coplanarVPs,raw}(1 - 0.8I_{muchSky,raw})(1 - 0.8I_{validCenterVP,raw}), \tag{5.11}$$

which leads to a prioritization of scenes, such that “much sky” excels ‘coplanar”, and “central vanishing point” again excels “coplanar". Further

$$I_{noVPs,rawB} = 0.8I_{noVPs,raw} + 0.2I_{muchSky,raw} \tag{5.12}$$

weights the sky indicator into the no vanishing points indicator.

All indicators are normalized over their overall sum. For a sum smaller than 0.1, a correction is introduced by

$$I_{sumA} = I_{validCenterVP,raw} + I_{coplanar,rawB} + I_{noVPs,rawB} + I_{nearVPs,raw}, \tag{5.13}$$

with

$$I_{lowConfidence} = \begin{cases} \frac{1}{0.1}(1 - I_{sumA}) & if \;\; I_{sumA} < 0.1, \\ 0 & else. \end{cases} \tag{5.14}$$

Thus, the vanishing point based branch gets less weight, if it detects poor confidence in its results.

Finally, the indicators are normalized by

$$I_{sumB} = I_{validCenterVP,raw} + I_{coplanar,rawB} + I_{noVPs,rawB} + I_{nearVPs,raw} + I_{lowConfidence}, \tag{5.15}$$

with

$$I_{validCenterVP} = \frac{I_{validCenterVP,raw}}{I_{sumB}},$$

$$I_{coplanar} = \frac{I_{coplanar,rawB}}{I_{sumB}},$$

$$I_{noVPs} = \frac{I_{noVPs,rawB} + I_{lowConfidence}}{I_{sumB}},$$

and

$$I_{nearVPs} = \frac{I_{nearVPs,raw}}{I_{sumB}}. \tag{5.16}$$

5.2.8. Generation of Vanishing Point Dependent Geometry

The indicators interact to describe aspects of the general scene layout, as shown in 5.2.8.

Indicator	tunnel	ramp	fixed depth
$I_{validCenterVP}$	+	−	−
$I_{coplanar}$	−	−	+
I_{noVPs}	−	+	−
$I_{nearVPs}$	−	+	−

Figure 5.8.: Listing of how each indicator influences weights of different depth maps. + means: a higher *I* means increased weight of the map, − means: a higher *I* means decreased weight of the map

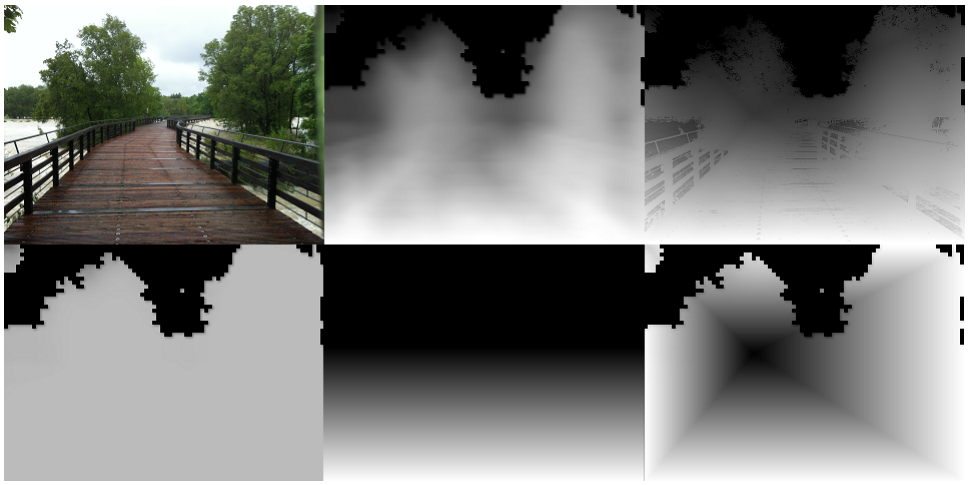


Figure 5.9.: Example image with different basic shapes applied. Upper row, left to right: original, texture based approach, final depth (including high frequency details). Lower row, left to right: coplanar, ramp, tunnel. Sky stencil is always applied.

Ramp and Tunnel Pattern

To generate initial global depth scene estimates based on the indicators calculated in the previous section, a set of three basic shapes is used. In scenes with a central vanishing point present, the global map consists of a rectangular tunnel centered around this point. If no such central vanishing point is present. If there is no central vanishing point, the second basic map is a ramp to the horizon and the third fixed distance map for coplanar scenes.

Thus, a geometrically derived estimation of depth $d_{x,y}$ for each location with coordinates x, y and image height h can be defined by

$$d_{x,y} = I_{validCenterVP}\, d_{x,y,tunnel} + 100\, I_{coplanarVPs} + (I_{noVPs} + I_{nearVPs})\, d_{x,y,ramp}, \tag{5.17}$$

where $d_{x,y,ramp}$ is defined as

$$d_{x,y,ramp} = \begin{cases} \frac{255(h-y)}{h-h_{horizon}} & \text{if } y > h_{horizon}, \\ 255 & \text{else.} \end{cases} \tag{5.18}$$

Note, that for these formula, the depth map is mapped to a byte, thus 255 is the maximum value $d_{x,y}$ can hold. Also, the distance for the coplanar pattern is arbitrarily set to 100, but

the absolute distance will be subject to scaling in consecutive steps anyway, when rendering additional views. The tunnel pattern is generated with a similar formula,

$$d_{x,y,tunnel} = 255 \begin{cases} \frac{x}{x_{vp}} & \text{left triangle,} \\ \frac{x_{width}-x}{x_{width}-x_{vp}} & \text{right triangle,} \\ \frac{y}{y_{vp}} & \text{upper triangle,} \\ \frac{y_{height}-y}{y_{height}-y_{vp}} & \text{lower triangle,} \end{cases} \tag{5.19}$$

with the tunnel centered around $(x_{vp}, y_{vp})^T$ and the image has the dimensions x_{width}, y_{height}. The image is thus divided in four triangles, each spanning from two neighboring image corners to the central vanishing point. While the nearest pixels are located at the edge of the image, all lines parallel to their own triangle's side which touches the image's rim within such a triangle that are iso-distant, thus have an equal value of $d_{x,y,tunnel}$.

Estimation of Horizon

The horizon's elevation $h_{horizon}$ is calculated adaptively. An initial estimate is the sky pixel spotted at the lowest elevation by the pre-marking algorithm as detailed in 5.2.3. Additionally, in the scenario with two nearby vanishing points, both are located on the horizon. This information might be used to detect a horizon that appears sloped due to camera rotation. However this approach shall be limited to sufficiently upright camera scenarios. Thus, the second estimate is the average height of the two nearest vanishing points. Note that even in scenes where the tunnel pattern applies, individual sky pixels always override the tunnels ceiling, so it is appropriate for scenes that consist of e.g. a street and buildings to its left and right. In such scenes the optimal estimate for the horizon is just the y-coordinate of the central vanishing point.

As the indicators are normalized in equation 5.16, the final estimate again is a weighted average of the initial estimates, given by

$$h_{horizon} = I_{validCenterVP} y_{VP1} + (I_{coplanarVPs} + I_{noVPs}) y_{preMark} + I_{nearVPs} \frac{1}{2}(y_{VP1} + y_{VP2}), \tag{5.20}$$

where y_{VP1} and y_{VP2} are the y-Coordinates of the vanishing points, $y_{preMark}$ is the lowest elevation by the pre-marking algorithm as detailed in 5.2.3.

5.2.9. Adaptive Fusion of Depth Maps

At this point, four separate depth maps are created, which now have to be fused to a final depth map:

- texture based map (5.2.1)
- perspective based map (5.2.6, 5.2.7 and 5.2.8)
- high frequency map (5.2.5)
- sky stencil map (5.2.3)

As first step, the two global depth maps, the texture based depth d_{tex} and perspective based depth d_{persp} are fused using following adaptive weighted sum, based on the corresponding indicators

$$d_{x,y,estA} = I_{validCenterVP} + I_{coplanarVPs})d_{x,y,persp} + (1 - I_{validCenterVP} - I_{coplanarVPs})d_{x,y,tex}. \quad (5.21)$$

In the second step, the final depth map gets created by fusing the sky stencil and the high frequency local details, via

$$d_{x,y,final} = 255\ b_{x,y,sky} + (1 - b_{x,y,sky})(d_{x,y,estA} - d_{x,y,highFreq}), \quad (5.22)$$

where $d_{x,y,final}$ is the final depth value, $b_{x,y,sky}$ is the result of the sky detector and either 0 or 1. $d_{x,y,highFreq}$ is the value of the high frequency map.

Remember, the indicators are normalized first, so they add up to 1. The high frequency map is then added to the result. Finally, pixels marked as sky are set to the maximum distance.

Note that the Texture based approach tends to fail in coplanar scenes, so it is attenuated if such a scene is detected. Also, if a scenario with a valid center point was detected, the perspective info with the tunnel pattern overpowers it. However, if the perspective module detects a situation where it s not useful, the texture based approach takes over.

The high frequency information complements the sky stencil in particular as it greatly improves resolution and precision of borders between sky and silhouettes of objects.

Chapter 6.

Evaluation

This chapter presents the evaluation of the three major contributions of this work. First, the introduced *SISeg* image segmentation algorithm will be compared with existing state of the art segmentation algorithms described in section 3.1 using the *Berkeley Segmentation Dataset* [72]. In the subsequent section, both the image information extraction scheme and the IBRIEF image-sequence correspondence matching concept are evaluated utilizing the video benchmark dataset from [40] for narrow baseline and the dataset from [75] for wide baseline matching. Third, results of 2D to 3D depth assignment algorithm of chapter 5.2 are presented and evaluated on the basis of the make3D dataset [97, 98].

6.1. Experiments and Results of SISEG

In this section I evaluate the performance of the SISeg algorithm by applying it to an image dataset plus evaluation benchmark, which I introduce in the following section. I further present and discuss a collection of examples in detail. The outcome is then compared to various approaches from the literature. Further I evaluate the algorithm's computational performance.

6.1.1. Evaluation Dataset and Metrics

As described in section 3.1, a major problem when it comes to evaluating boundary detection or image segmentation algorithms, is the need of a ground truth, and at the same time the lack of uniqueness of such. One of the most extensive and widely used image collections for this purpose is the Berkeley Segmentation Dataset (BSDS) [72], which pro-

vides a set of 500 natural images of size 481×321, each along with 5-10 human-generated segmentations, which serve as ground-truth.

General evaluation metrics for image segmentation are for example the Probabilistic Rand Index [117], the Jaccard index, or the Object-level Consistency Error OCE [89]. Such metrics are usually based on the amount of overlap between ground-truth segments and segments detected by the algorithm. They differ for example in the used distance measures, the punishment for under- or over-segmentation, or the way the overlap is weighted with regard to the segment sizes. However, the metrics mentioned above demand image regions with closed boundaries that are fully separated from each other, so any pixel in the image can be assigned to one segment exclusively. Therefore, those metrics cannot easily be applied to the topic of boundary detection, as many of those algorithms produce potentially non-closed contour estimates rather than segmented regions. Nevertheless, boundary based evaluation metrics exist, like *Pratt's Figure of Merit (FoM)* [92], *Precision-Recall Curves (PRC)* [72], and *Receiver-Operator-Characteristics (ROC)* [14], whereas the *PRC*-based benchmark system in [72] allows for comparison of region-based segmentation and contour detection methods in the same framework. Those methods usually measure the coincidence and the accuracy of boundary or edge pixels compared to ground-truth boundaries.

Along with the BSDS image database, [72] delivers a benchmark, in which the ground-truth data is compared to machine generated output using *Precision-Recall Curves*. *Precision* measures the probability that a machine-generated boundary pixel is a true boundary pixel, while *Recall* gauges the probability that a true boundary pixel is detected, i.e. the amount of ground-truth captured by an algorithm. Again, in order to obtain binarized edge maps from the algorithm's edge saliency output, *Precision* and *Recall* are calculated for a set of varied thresholds, which captures the trade-off between the accuracy and noise, i.e. the false-positives and the true-negatives.

To approximate the *Precision-Recall Curves'* optimal trade-off, the *F-Measure* is calculated for each threshold. The *F-measure* is the harmonic mean of *Precision* and *Recall*, calculated by

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}.$$

Another well-known metric in image evaluation is the *Receiver-Operator-Characteristic*, or *ROC-Curves*, whose axes are called *Fallout* and *Recall*. These metrics are comparable to the *Precision-Recall* metric. The *Recall* is defined identically in both metrics. *Fallout* measures the probability that a true negative was labeled a false positive. Just as the *PR-Curve*, the *ROC-Curve* is a graphical plot that displays the performance of a binary classifier

system with a varied binarization threshold. However, the definition of the *Fallout* makes the *ROC-Curves* dependent on the image resolution, while the *Precision-Recall Curves* have the advantage of being scale-invariant [85] and are therefore preferred.

The BSDS benchmark delivers three major quantities: first the Optimal Dataset Scale (ODS), which is best F-measure on the dataset for a fixed scale or threshold, second the Optimal Image Scale (OIS), which is the aggregate F-measure on the dataset for the best scale in each image, and third the Average Precision (AP) on the full *Recall* range, which is equivalently the area under the *Precision-Recall Curve*.

6.1.2. Examples and Discussion

The BSDS image dataset is partitioned in images for training, testing, and validation. For the evaluation, I ran the algorithm on the test- and validation-set of 300 images overall.

I used the training set to empirically find the optimal parameters for the thresholds used in SISeg for non-maximum suppression and region fusion. The parameter values, that were set when generating the provided outcome, are given in the appendix A.

Initial experiments also revealed a minor drawback of the Berkeley evaluation measure F, as the outcome of evaluation can be influenced by adding some post-processing steps to the initial outcome, like e.g. scaling, or thinning the boundaries. Such steps improve the result, although the initial result of the algorithm is untouched.

Figures 6.1 to 6.4 present example images categorized into four different categories, each along with the ground-truth, boundary images resulting from SISeg, and the corresponding *Precision-Recall Curves* with the optimal *F-measure*. Figure 6.1 shows example images, where the algorithm performs well, as the picture content is quite distinguishable and consists of salient regions. The performance on the images displayed in the introduction (see Figure 6.2) is similar to the average performance over the entire image set. The image content includes a balanced mixture of strong edges, salient regions, and also cluttered or textured regions.

The three example images of Figure 6.3 contain a considerable amount of textured regions, which leads to a significant performance loss of the SISeg-algorithm. This is due to the fact, that textures are not explicitly considered when calculating the region boundary estimates. An improvement is to expect, when the agents are extended to incorporate texture cues like Textons, or Local Binary Patterns into the agent distance value formula. Furthermore, examining the image information inside an agent's neighborhood for repeating patterns

texture-like structures seems promising. This could be subject to investigation in future work.

In [85] the authors argue that contour detection, while being closely related, is not perfectly the same as boundary detection. They state that contours, which might be important in terms of the image content do not necessarily have to be region boundaries, and might thus be excluded in boundary detection. The three example images in Figure 6.4 illustrate the difference between boundaries in a semantic segmentation context, and contours between both objects and object parts. It is obvious that my algorithm detects contours rather than boundaries, e.g. the single windows in the left picture, or the black and white stripes of the zebras. While extracting these salient contours might be desirable in some applications, it significantly degrades the performance of my algorithm with regard to boundary detection, as can be seen in the *PR-Curves*. To a certain extent, these misleading detections might supposedly be alleviated by considering textures, as stated above. For further elimination, additional information, e.g. in form of object or appearance models, knowledge, or training might become necessary.

Further, one must regard, that the ground truth of the Berkeley evaluation benchmark follows the purpose of semantic segmentation, like segmenting a complete human from the background, or one house as a whole from the others. Yet, taking into account my initial purpose to find all segments relevant for 2D to 3D conversion, the slightly over-segmented output of the SISeg algorithm can fulfill this condition better than the goals of the Berkeley dataset evaluation. For a proper result of a depth map, having a complete house as one segment might be too coarse, as one might want to add different depths to parts of the house. Thus, what is an error in semantic segmentation is possibly a desired image detail in depth map generation. However, a segmentation benchmark for the context of depth map generation does currently not exist, and as already mentioned the Berkeley Segmentation Dataset is widely utilized and allows for comparing a variety of approaches.

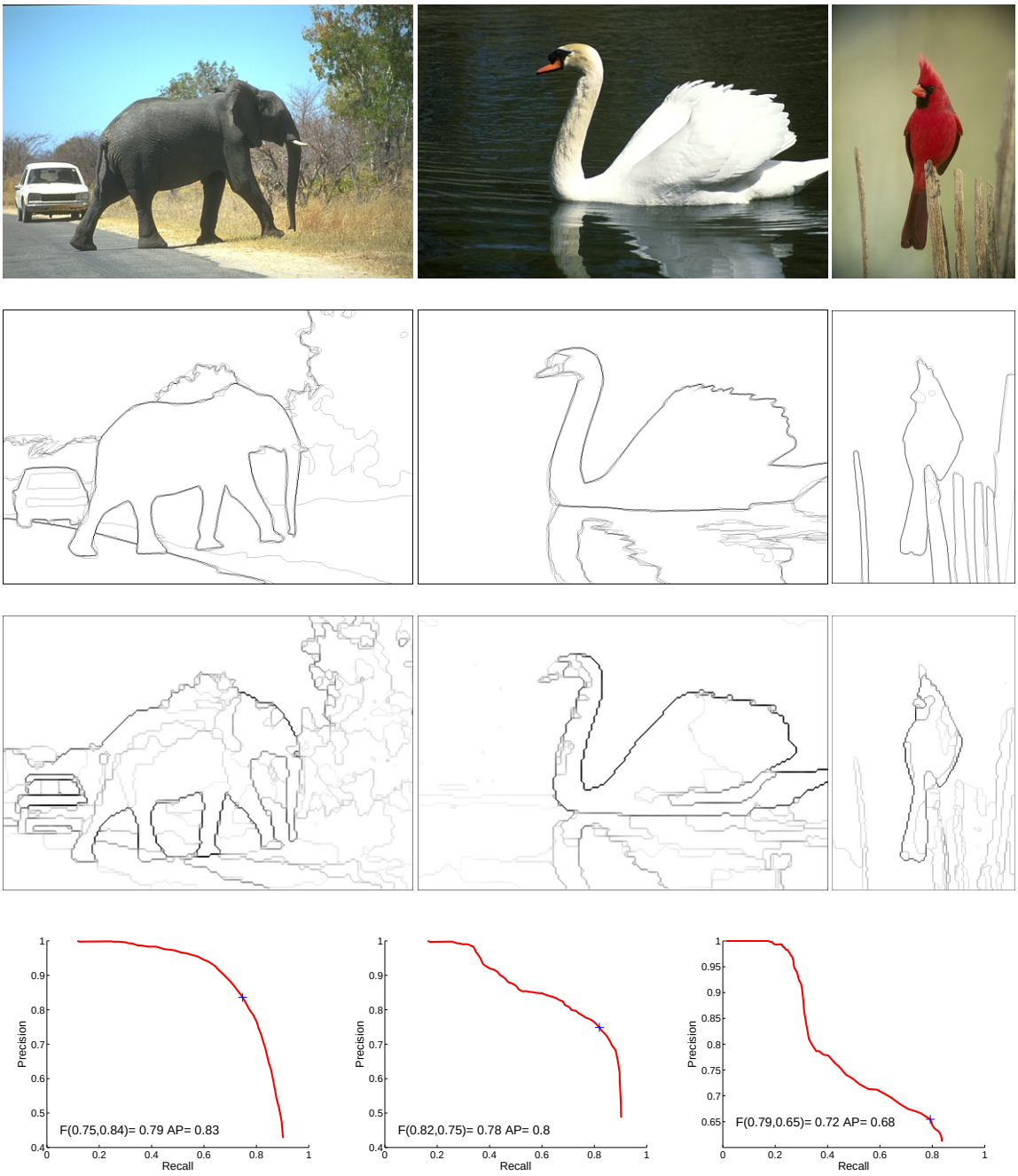


Figure 6.1.: Three sample images with distinctive content from the Berkeley Image Segmentation Dataset (BSDS 500) [6]. From top to bottom: the original images, the summed ground-truth images, the result images and the *precision-recall curves* of the SISeg-algorithm.

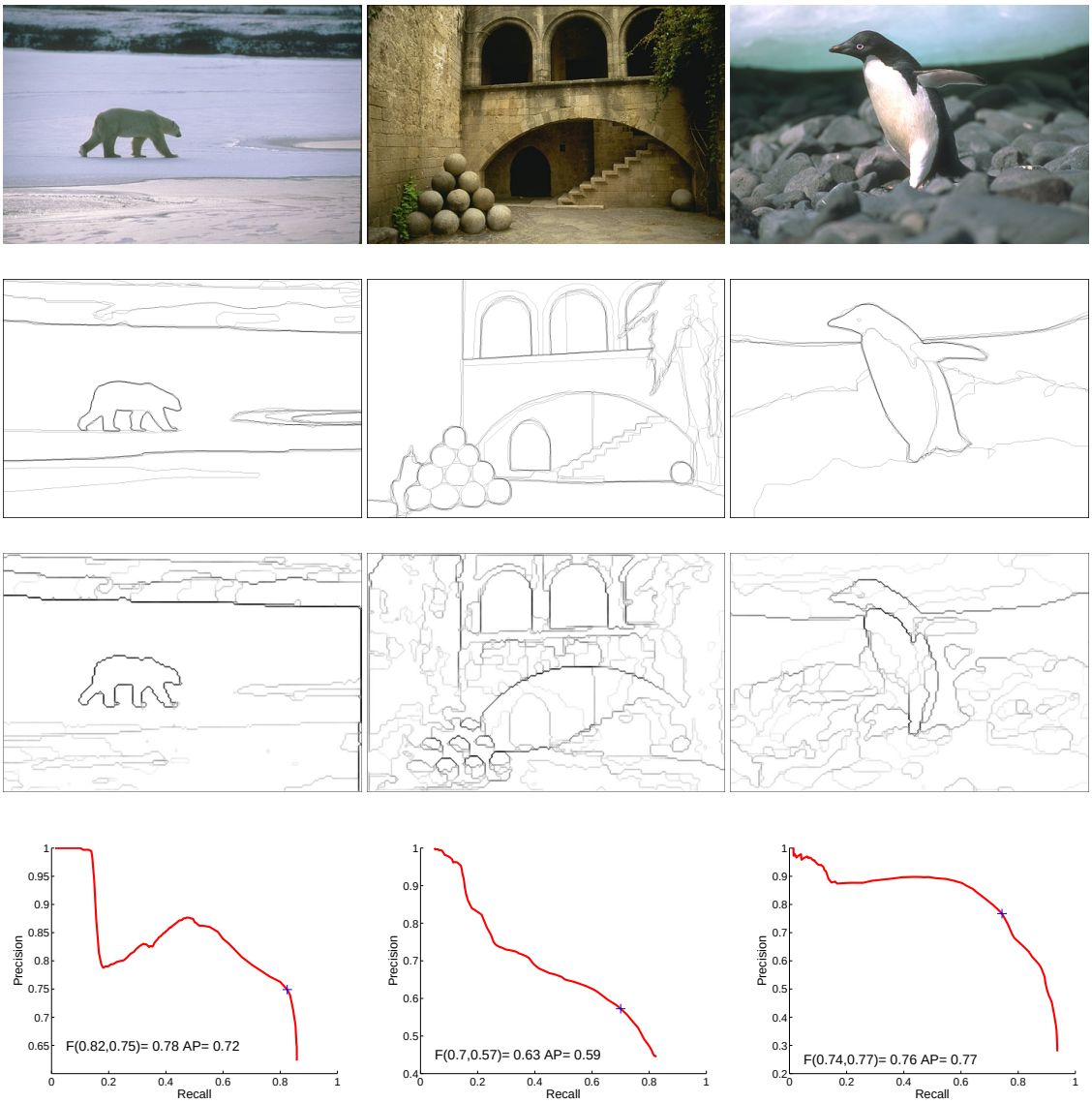


Figure 6.2.: Three sample images from the introduction Section, which yield to average performance with regard to the overall results. Taken from the Berkeley Image Segmentation Dataset (BSDS 500) [6]. From top to bottom: the original images, the summed ground-truth images, the result images and the *precision-recall curves* of the SISeg-algorithm.

6.1.3. Comparison

Table 6.1 shows the evaluation results of the SISeg-algorithm from the BSDS, compared to a variety of approaches from literature. One can generally state, that learning based

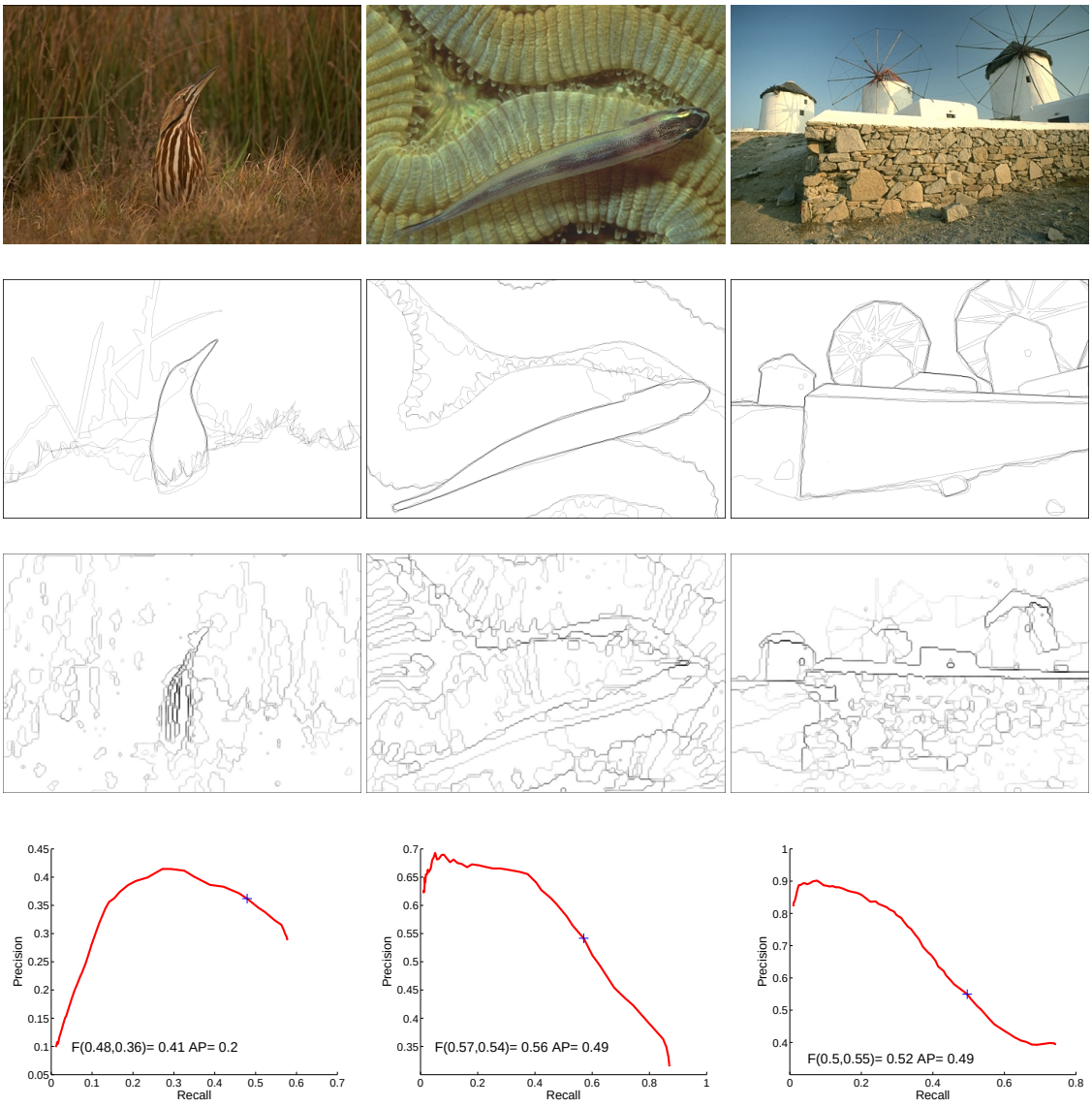


Figure 6.3.: Three sample images containing strong texture. Taken from the Berkeley Image Segmentation Dataset (BSDS 500) [6]. From top to bottom: the original images, the summed ground-truth images, the result images and the *precision-recall curves* of the SISeg-algorithm.

approaches [51, 31, 127, 6, 66] outperform non-learning based approaches, like [24, 25, 35, 106] as well as the SISeg method introduced in this thesis, yet mostly at the price of a high computational complexity overhead.

Similarly, most of the methods compared in Table 6.1 utilize further image cues than simple

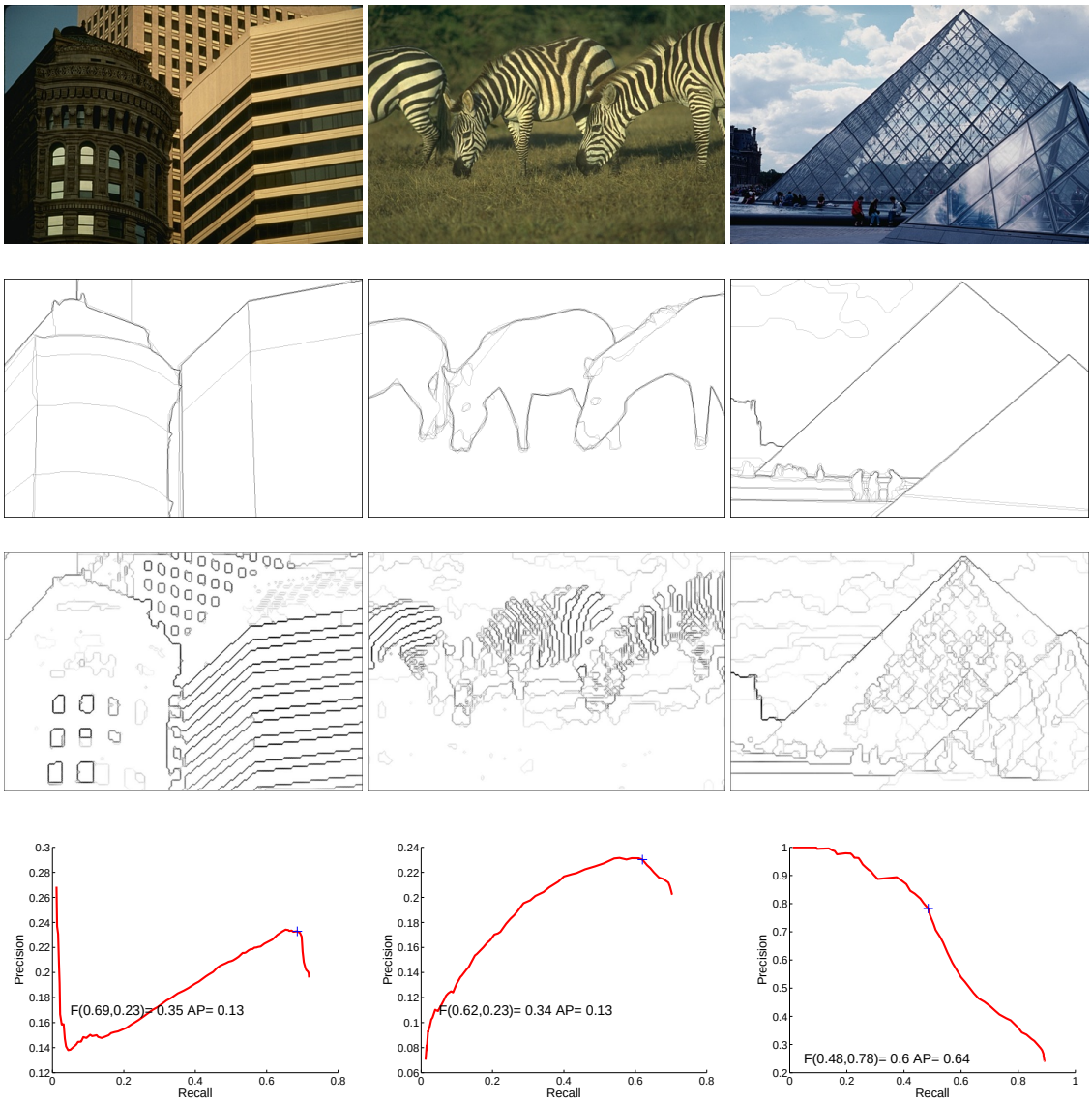


Figure 6.4.: Three sample images containing strong contours that are not boundaries. Taken from the Berkeley Image Segmentation Dataset (BSDS 500) [6]. From top to bottom: the original images, the summed ground-truth images, the result images and the *precision-recall curves* of the SISeg-algorithm.

single scale window patches. Applying various other appearance distance measures, plus analyzing additional image information, like for example texture information is very likely to also improve the SISeg results.

	ODS	OIS	AP
Human	0.80	0.80	–
Crisp Boundaries (MS) [51]	0.74	0.77	0.78
SE (MS, T= 4) [31]	0.74	0.76	0.78
SCG (global) [127]	0.74	0.76	0.77
gPb-owt-ucm [6]	0.73	0.76	0.73
Sketch tokens [66]	0.73	0.75	0.78
gPb [6]	0.71	0.74	0.65
Mean Shift [24]	0.64	0.68	0.56
NCuts [25]	0.64	0.68	0.45
Felz-Hutt [35]	0.61	0.64	0.56
Canny-owt-ucm [6]	0.60	0.64	0.58
Canny [18]	0.60	0.63	0.58
SWA† [106]	0.56	0.59	0.54
SISeg	0.64	0.66	0.60
(Precision, Recall)	*(0.72, 0.57)*	*(0.74, 0.60)*	

Table 6.1.: Boundary benchmark results on the *BSDS*500 test dataset (†:*BSDS*300 dataset) [6], different segmentation methods, or respectively contour detectors plus the SISeg algorithm in the lower Table. The values represent the F-measures using an optimal scale for the entire dataset (ODS) or per image (OIS). The area-precision is expressed in (AP). For SISeg also the values of Precision and Recall corresponding to the F-measure are given.

6.1.4. Computation time

Two parameters have principal influence on the computation time of the SISeg-algorithm. The first is the number of agents N, which in turn depends on the image size, resulting in $\frac{1}{3} \times H \times W$ as worst case, and the second factor is the maximum number of iterations τ^t. With these two, the computation time of the algorithm is linearly dependent as $O(N \cdot M)$. Further I must state, that this worst case is highly theoretical, as it would imply, that all agents pass the non-maximum suppression and iterate until the maximum number of iterations. Both conditions are unlikely to happen.

I ran the algorithm on an Intel Core 4 CPU, 3.30 Ghz and 16 GB of RAM. The algorithm was scripted in MATLAB., with mex-file based *C* implementations for single parts of the algorithm. Besides that no further code optimization was utilized. The algorithm ran with about $0,275$ seconds per image of size 481×321, averaged over the 200 images of the BSDS test dataset.

The overall run-time splits into several phases, which all use a certain amount of time, i.e. calculating the distances, initializing the agents' models, running the boundary grow phase,

and finally execution the image analysis steps. So the algorithm's core from a Swarm Intelligence point of view would be reached in less time. Also, for mere segmentation or boundary detection like in the benchmark shown above, one can skip the time used for calculating the boundaries curvature and anchorpoints. For image sequences, all windows for indices of course need to be calculated only once for all consecutive images. Besides that, this approach with its nature of distributed agents is extremely well suited for parallelisation. In each iteration, each agent senses its envidronment and the executes its model updates independently of all others, thus all agents can update parallel. After this each agent of course needs the updated view on its environment again. In fact, the way it is treated in MATLAB already points in this direction, as agents can easily be stacked in matrices, demanding mainly element-wise operations. Table 6.2 shows an overview of mean computation time for the images of the dataset, split into the consecutive phases. Keeping in mind, that the computation time of C++ implementations usually outperform MATLAB implementations by magnitudes, the computational speed of SISeg can be regarded as promising.

Preparation Windows initialization	**SISeg calculation** distances calculation, initialization & loop phase	**Postprocessing** Extraction phase	**Sum**
$0,029 \pm 0,002$	$0,151 \pm 0,011$	$0,096 \pm 0,009$	$0,275 \pm 0,019$

Table 6.2.: Mean calculation times plus standard deviation, in seconds. Measured on a workstation with 3.30 Ghz, 16 GB RAM. Matlab script-based implementation aided with mex files for some parts of the algorithm.

For comparison, the *gPb*-method from [6] demands around four minutes of computation time for one image of the dataset using a regular *C* implementation, as stated by the authors. As stated in [19], the computation time of the *gPb*-method was reduced to 1.8 seconds utilizing parallelized GPU implementation. Thus, I believe that the computational speed of this method is advantageous when it comes to time-dependent applications, which holds compared to the majority of approaches that use machine learning.

However, there is also one exception [31], which obtains extraordinary results with amazing computation time reported. Yet it must be mentioned, that they achieve this by utilizing massive performance optimization, including multi-core and GPU-based parallelization to reach this. Also, considering merely the high complexity of the input features they utilize, makes optimization steps necessary.

While a Matlab implementation performs considerably slowly compared to a C++ implementation, its matrix structure shows, that the SISeg algorithm is well suited for a speeded

up GPU based variant. Also the agent's distributed nature allows for parallelisation, basically each agent could run in an extra thread in each iteration. This makes the concept of SISeg suited to perform in real-time.

If one compares SISeg's evaluation results to the introduced machine learning-based approaches one can see, that there is a considerable gap in the results. However, the approaches from literature are specifically tailored to the problem of *semantic segmentation* proposed in the Berkeley benchmark. The SISeg segmentation initially has a slightly differing focus delivering a segmentation that is the base for further image analysis providing input to depth conversion. This leads to over-segmented results in terms of the semantic segmentation. As an example, while semantic segmentation defines a human or a house as one segment, this can mean a loss of details if that segment shall be input to depth conversion.

Nevertheless, there is a number of extensions to the current implementation possible, which promise qualitative performance boost. First, there is a variety of alternatives for window based feature extractions and distance metrics, like histograms, texture based features, oriented gradient, alternative image spaces, and so on. Some of these require input windows, that are considerably bigger than the currently used 3×3 windows. This would lead to overlapping agent windows, but it would not represent a problem for the succeeding calculations in the concept, as long as they result a distance measure for each of the eight neighboring directions of each agent.

Further, the concept is feasible to be combined with machine learning concepts. As an example, the image training set plus ground truth within their windows could be used to guide the agents in their boundary and connections decisions, for example by introducing weights in the decisions. Like this, boundaries detected by the agents could be reinforced, if they correspond to boundaries in the ground truth, and weakened, if they don't.

Summarizing, utilizing the principles of SI with focus on self-assembly in my view seems a very promising solution to the segmentation, or respectively contour detection problem. As already stated, the SISeg concept with its fixed grid is only one variant, one can think of. A hill climbing-alike movement step, as introduced in [57], which leads to a warped agent grid, is another reasonable variant.

6.2. Experimental Results of IBRIEF

In this section I show the results of the proposed algorithm and compare it with a state-of-the-art approach, called MSCD, which was introduced in section 4.2. I introduce video sequences as a narrow-baseline dataset and different image sets as a wide-baseline dataset to examine the performance in both use cases. Additionally I introduce the performance measures and how I applied them for edge correspondences. I conduct thorough experiments with different approaches and discuss the results. The results presented in the following are part of the work in [10], which holds further details on results.

6.2.1. Experimental Setup of IBRIEF Results

All of the used software is implemented in *C* + +. OpenCV [1] was used for handling and accessing all image and video related data.The test system was a workstation with an Intel Q6600 processor and 4GB RAM. As the operating system Windows 7 was used. OpenCV was used with version 2.41 and was configured to make use of Intels Threaded Building Blocks (TBB) [2].

All parameters for the experiments were established beforehand empirically. This might not correspond to the best matching results, but the parameters do not change throughout all experiments and are thus comparable and reproducible. The used parameters for the experiments in this chapter, and for all other conducted in this work, are listed in appendix A.

6.2.2. Results for Video Sequences

In this section I introduce the dataset for video sequences. The IBRIEF method is evaluated with the dataset of [40], since it covers tracking from frame to frame, it is publicly available and contains thorough motions and context, which mirror most of the effects encountered in tracking. The dataset in [40] is aimed for visual tracking. The authors evaluate several state-of-the-art interest point feature extractors and descriptors and adapt their measurements to the case of visual tracking. The evaluation is based on a planar acrylic object that embeds different textures and is filmed with different motion and lightning patterns. Effects in tracking that can be evaluated with this dataset are:

[1] http://opencv.org/
[2] http://threadingbuildingblocks.org/

- in-plane rotation
- scale change
- varying light conditions (static, dynamic)
- motion blur

Effects that can not be evaluated by this dataset are thus not evaluated in this work either. These effects are extreme local non-planarity and occlusions and should be considered in follow up work.

Figure 6.5.: Textures for evaluation from left to right and used abbreviation in brackets: wood(wd),bricks(br), building(bu), paris(pa), mission(mi), sunset(su). Images taken from [40]

To measure the performance of feature detectors and descriptors, ground truth needs to be established, relating one point $\mathbf{x}^k$ at frame k with point $\mathbf{x}^l$ at frame l. This is not possible in general for 3D scenes [41]. In this dataset planar scenes, that can be related by a homography $\mathrm{H}^{kl}(q) \in \mathbb{R}^{3\times3}$, such that

$$\mathbf{x}^k = \mathrm{H}^{kl}(q)\mathbf{x}^l. \tag{6.1}$$

Both $\mathbf{x}^k$ and $\mathbf{x}^l$ refer to frames in which distortion, introduced by camera imperfections, are already removed, i.e. undistorted frames. Both coordinates are in homogeneous coordinates, that is $\mathbf{x}^k = (x, y, 1)^T$. The homography is estimated by a set of markers placed on a precisely milled acrylic glass frame, which are tracked with a semi automatic tracking process using an adaptive color model.

The test bed consists of 96 video streams, which include six different textures strapped onto the acrylic glass frame and 16 different motion patterns. The motion patterns are built up from the following scenarios:

- *unconstrained (uc)*: except that image plane is inside the field of view, smooth motion (6×500 frames)
- *panning (pn)*: camera is about $1m$ away from object and camera pans sideways (6×50 frames)

- *rotation (rt)* : camera is located about 1*m* away from object and camera rotates around the optical axis from $[0, 90°]$ (6×50 frames)
- *perspective distortion (pd)*: camera is located above the object (perpendicular) and camera goes down in an arc (6×50 frames)
- *zoom (zm)*: camera moves from 60*cm* to 130$[\pm 10]$*cm*
- *motion blur (m1-m9)*: camera with pan-tilt unit set to 9 different camera-pans each having increased speed. ($6 \times 9 \times [13 - 89]$ frames)
- *static lightning (ls)*: static camera, for different lightning setups ($6 \times 4 \times 20$ frames)
- *dynamic lightning (ld)*: static camera with a lightning transition from bright to dark and back to bright. (6 frames)

In total 6889 frames for the complete dataset.

Figure 6.6.: Some example frames displaying the different motion patterns. From left to right: the first two images show a scale change, rotation, perspective distortion, motion blur and lightning. The evaluation uses only the area inside the black-and-white border pattern.

For video sequences I use the nearest neighbor precision, since for most tracking applications the nearest neighbor is the most important one [40]. The Precision of the first nearest neighbor is calculated by evaluating

$$Precision_{1NN} = \frac{\#\ correct\ matches}{\#\ correct\ matches + \#\ false\ matches}, \tag{6.2}$$

which gives the probability, that a random chosen match was correct. A correct match can be verified by projecting the feature $s_i^k = \mathrm{H}^k * s_i^{k-1}$ from frame $k-1$ into the new frame k and compare it with the best correspondence, having the smallest distance in feature space. The projection is executed for each pixel coordinate in $s_i^k = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ...\}$. Due to noise perturbations and image discretization a feature will mostly never lie on the same exact pixels, therefore the closest feature in Euclidean space which fulfills some overlap criteria is considered the correct match.

Since the features are represented as a set of image points and are of arbitrary shape, one can not simply use the Euclidean distance between the feature $\mathbf{p}_i^k$ and the projected feature

$\mathbf{p}_j^{k-1}$ to verify a correct match. To cope with this problem, a feature that has an overlap of at least τ with the projected feature, is declared to be a correct match. I measure the distance between each point of the feature $\mathbf{p}_i^k$ with the nearest point of the projected feature $\mathbf{p}_j^{k-1}$ and if it is lower than some threshold ϵ $\mathbf{p}_i^k$ is considered to be the correct match of $\mathbf{p}_j^{k-1}$. A pixel-in-polygon test is used, where the distance between the feature point and the query point is measured. Additionally, a search area for the best candidate is defined as the extended minimum area rectangle containing all points of the feature by adding a fixed value l_{sr} to its width and height, as it is already detailed in the I-BRIEF matching strategy in Section 4.4. The rectangle size is set to l_{sr} = 100 for all experiments over all algorithms.

To make the descriptor independent of the feature extractor, features are detected once in the first frame and then re-projected into the subsequent frames using the ground truth warps that are given for each frame k by $\mathtt{H}^k$. This technique is already applied to tune I-BRIEF in Section 4.3.2. To evaluate the feature descriptors in a real-life environment, the same extracted features $\mathbf{p}^k$ are passed to each descriptor.

The proposed edge-based description and matching concept is compared with those proposed in [122], especially with the more general applicable MSCD algorithm discussed in Section 4.2. MSCD is the closest algorithm to this work, since it can handle arbitrary shapes and utilizes feature matching based on distance functions. Additionally, both descriptors have a similar memory footprint. A single MSCD descriptor in the following experiments occupies $72 * 4$ *bytes* = 288 *bytes* with I-BRIEF using only 64 *bytes*.

The major difference between the two concepts is, that MSCD can cope with varying length of the edge segments inherently, whereas IBRIEF needs anchor points. The effects of this difference shall be outlined by employing four different algorithm combinations.

IB64 1NN - I-BRIEF only making use of endpoints with a 1NN matching strategy. It is further described in Section 4.4.

IB64 IBMatcher - I-BRIEF making use of the proposed matching, see Section 4.4.

MSCD 1NN - MSCD descriptor is calculated between endpoints of a feature, similar to *IB64 1NN*. As the matching strategy 1NN from Section 4.1.3 is used.

MSCD 1NN AP - MSCD descriptor is calculated for all sub-features, which result from splitting features at high curvature points. The same matching strategy as for *MSCD 1NN* is applied.

Results

In this experiment, the behavior of MSCD and I-BRIEF are examined in a tracking scenario, using two different environmental settings. In the first, the input is detected only once and the detected contours are projected into subsequent frames. In the second, the edges are detected in each frame and passed to all descriptors under test. The first setting thus represents a theoretical scenario with perfect repeated edge extraction as input, while the second displays a more realistic, “imperfect” input. The results for the first and second setting are presented in Fig. 6.7 and Fig. 6.8.

In the first setting I-BRIEF outperforms MSCD in each category. It is notable that *IB 1NN* and *IB IBMatcher* achieve almost perfect results throughout the testbed, while *MSCD-1NN* outperforms the use of anchor points in MSCD. The reason for this might be, that the anchor points can split contours with a similar appearance, confusing the matching inside of the search region. This first setting proves, that I-BRIEF is able to achieve the task of matching arbitrarily shaped edges with high precision.

In the second setting, which is intended to test the algorithms under more realistic conditions, I-BRIEF still achieves good results. The distance between MSCD and I-BRIEF methods shrunk and is now bounded by the fidelity of the input, especially by the endpoint stability. In the case of increasing speed, as given in the scenes of $\{m1 - m9\}$, MSCD can outperform I-BRIEF. Since in the ”perfect” setting I-BRIEF achieves nearly stable results throughout all different speeds, it stands to reason, that in these cases of high motion blur, end points and anchor points vary too heavily for I-BRIEF. *IB64 IBMatcher* stands out in means of nearest neighbor precision against all others, which can be explained by the more profound use of anchor points. Taking a closer look at the recall of *IB64 IBMatcher*, which gives the probability that a randomly chosen feature is correctly matched, it can be observed that the recall is lower than for the other algorithms. This stems from the post processing step of the matching. This additional step here seems too restrictive to handle the events of split edges. On the one hand it rejects false matches correctly resulting in high precision, but on the other hand it prunes possible further correct matches. A more improved detection of split edges is therefore needed. Even without the specially tailored matching, it achieves slightly better results in terms of mean precision throughout all the averaged sequences and in the dynamic lightning sequence, but performs worse in the motion blur scenes $\{m1 - m9\}$.

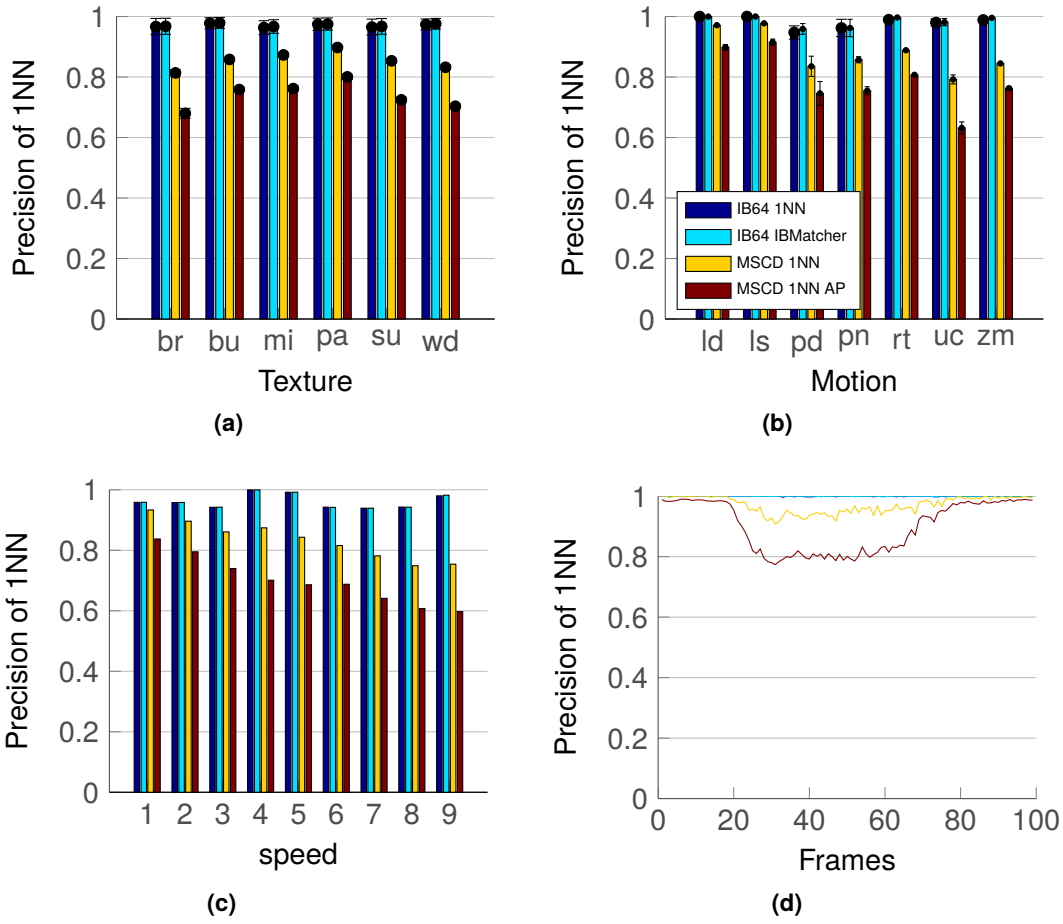


Figure 6.7.: Results for narrow-baseline with perfect edge detection. (a) and (b) show the mean precision and variance-averaged over all motion sequences and textures respectively, except $\{m1 - m9\}$. (c) depicts the mean precision and variance for increasing speed sequences $\{m1 - m9\}$ (d) distributes the mean precision in each frame for the dynamic lightning sequence.

6.2.3. Results for Wide-baseline Image Pairs

Although not of primary research in the task of image sequence correspondences, as in regular image sequences, the baseline can be considered as small, the I-BRIEF performance was also evaluated for wide-baseline-matching, which was introduced in section 4.2.

For the wide-baseline matching I use a subset of the dataset, which was introduced in [75] to measure the performance of local descriptors and has become a de-facto standard for this purpose.

I will only focus on three image pairs of this dataset, which are called 'boat', 'graf', and

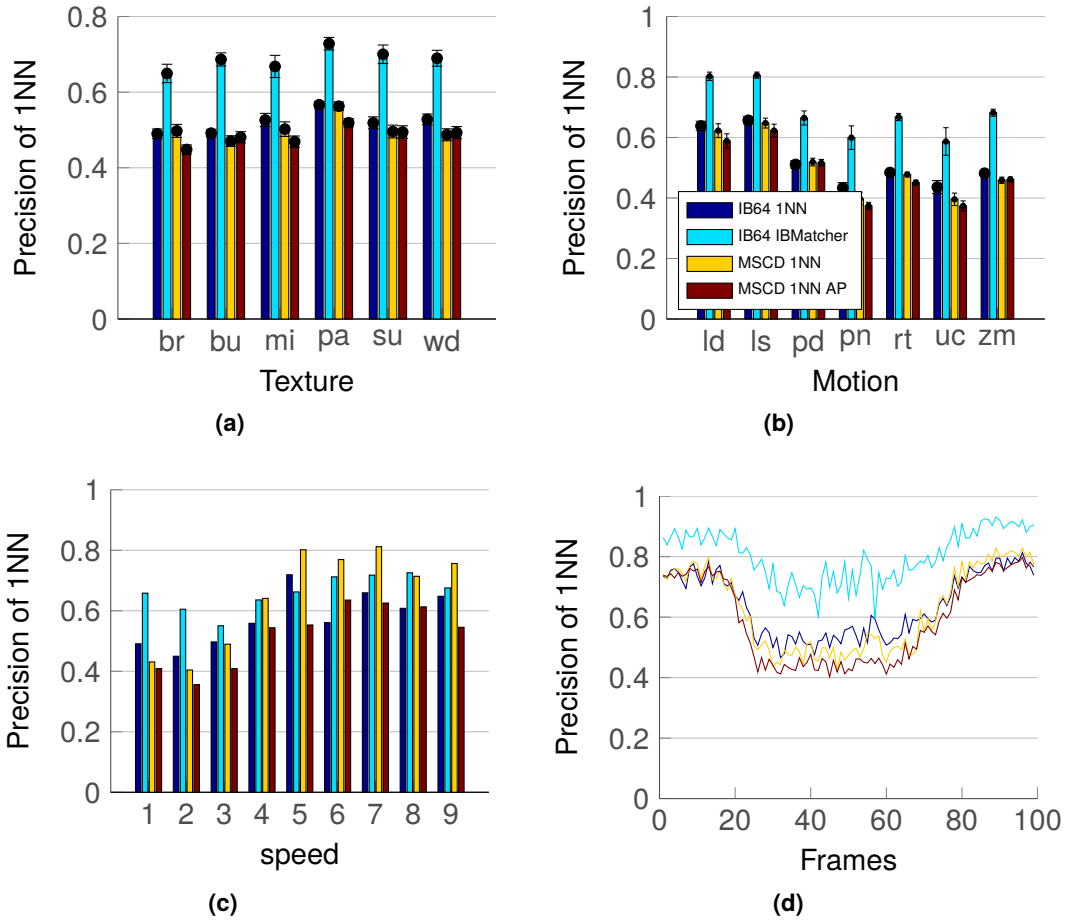


Figure 6.8.: Results for narrow-baseline with the edge feature extractor introduced in this thesis. (a), (b), (c) and (d) depict the same testbed as in Fig. 6.7, but with as input.

'wall', because image blurring and light change are already covered by the narrow baseline evaluation. Figure 6.9 shows example images of the three sets. Additional aspects will thus come from the scenes depicting zoom+rotation (boat) and view point change (graf, wall). The subset contains three different scenes (boat, graf, wall) from different angles with significant viewpoint change as can be seen in Fig. 6.9.

The first image is always used as the reference and compared to the remaining image, which results in 3×4 image pairs ($1-2$, $1-3$, $1-4$, $1-5$) for evaluation, with increasing baselines between the images in order of the images. The evaluation process is carried out manually, since not all images are near-planar and display 3D scenes, which might lead to incorrect behavior of the ground truth estimation. This results in a list of correct and incorrect

Figure 6.9.: Showing 4 images of each dataset. Images shown of each sequence (boat, graf, wall) $\{1, 2, 3, 5\}$

matches, using the same performance criteria as in [122]. The first is the count of correct matches (CM) and the correct ratio (CR) of total matches of an entire image sequence as

$$CR = \frac{\#\ correct\ matches}{\#\ total\ matches}. \tag{6.3}$$

In this experiment, the results of IBRIEF are again compared to MSCD, using the NNDR matching strategy for both approaches. The general process is the same as explained in Section 4.4 for both algorithms, using a minimal length of 20 pixels. The remaining edge segments are then described with either I-BRIEF-64 or MSCD, with both I-BRIEF and MSCD given their best possible parameter setting. The NNDR ratio is set to $r_{NNDR} = 0.8$ for the boat and graf sequence, and ratio of $r_{NNDR} = 0.6$ is applied to the wall sequence, where the lower ratio for the wall sequence is chosen to keep the manual evaluation labor a reasonable task.

Results

Results for the image pair of the three sequences are given in Table 6.3, giving the correct and false matches in brackets for each image pair and the results by the means of Correct Match and Correct Ratio for all image pairs in each scene. The first row provides the number of detected input features.

The results of CM are very similar in the graf and wall sequence. For the boat sequence MSCD clearly outperforms I-BRIEF. Here, I-BRIEF can only find a reasonable amount of correct matches in the first image pair. The overall bad performance of both approaches

can be explained by the high change in viewpoint and rotation, which is not appropriately handled in the edge extraction. This could be improved via a multi-scale edge extraction. In the graf and wall sequence, where viewpoint change is not as extreme as in the boat sequence, both do reasonably well and are of similar quality by means of CR.

MSCD is able to establish more CM in all sequences. This might be a hint, that it is more distinctive for wide base-line matching, since the NNDR ratio are the same for both algorithms. This higher distinctiveness is achieved by evaluating the entire surrounding region of the edge-based feature opposed to evaluating only regions around two points.

Method	1-2	1-3	1-4	1-5	CM	CR
I-BRIEF	508,459	285	218	186	-	-
MSCD	216(24)	23(55)	2(58)	6(44)	268	0.597
IB64	103(10)	0(23)	0(24)	1(12)	104	0.375

(a) boat

Method	1-2	1-3	1-4	1-5	CM	CR
I-BRIEF	499,512	555	557	661	-	-
MSCD	199(32)	70(41)	33(51)	6(44)	308	0.647
IB64	56(12)	37(11)	4(9)	1(12)	98	0.690

(b) graf

Method	1	1-3	1-4	1-5	CM	CR
I-BRIEF	542,507	465	389	395	-	-
MSCD	121(1)	66(0)	29(0)	5(0)	308	0.995
IB64	129(4)	82(1)	32(0)	4(0)	247	0.980

(c) wall

Table 6.3.: Results of wide-baseline matching using NNDR for each sequence. Tables ((a)), ((b)) and ((c)) give the number of correct and false matches in the form correct(false). Additional the retrieved number of features are given in the row called I-BRIEF.

6.2.4. Computation Time

This section presents the execution times of the evaluated approaches from Section 6.2.2. Let it be noted that both implementations might not be optimal and that there might be still some optimization potential. Nevertheless, the results here indicate that the introduced approach is superior to MSCD by means of execution time for description and matching. The results were taken from all six unconstrained video streams of the narrow-baseline dataset, resulting in an average of 2500 frames for each algorithm. In addition to the *IB-64 1NN*

variant, the results for the 16 and 32 bytes based I-BRIEF are given, showing the averaged computational time over all six image sequences. The results of the used implementation MSCD are of the same magnitude as in [132], where only values for MSLD are given. The execution of MSCD is directly proportional to the length of a features, since for all pixels on a curve the gradient must be extracted, where as I-BRIEF execution time is only depending on the number of anchorpoints used. Table 6.4 shows the computation times for each method, along with the mean number of correspondences used in this experiment, plus the detailed share of the computational time for description and matching, given per feature in milliseconds. As can be seen the all I-BRIEF methods are very similar and are computationally efficient while occupying only small sizes of memory. The advanced *IBMatching* scheme, where many additional steps are conducted and matching is actually done with sub-features, takes twice the time 1*NN* matching does per feature. The slowest, but best performing, I-BRIEF description and matching concept achieves a $6 - 13$ fold speed up compared to MSCD.

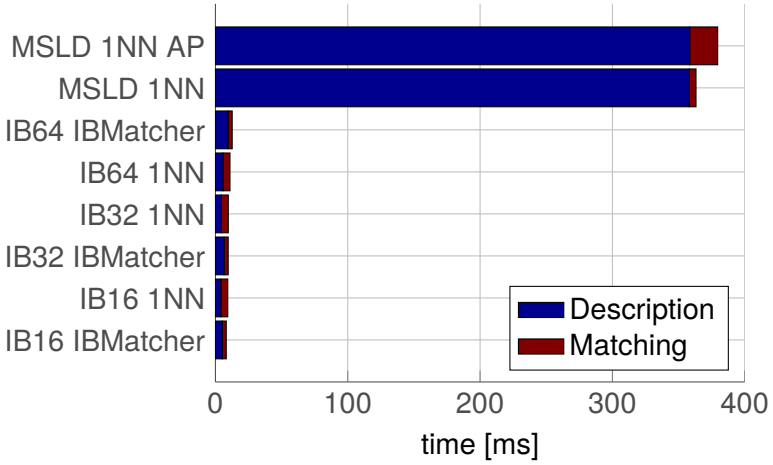


Figure 6.10.: Mean execution time of six different unconstrained sequences from the narrow-baseline dataset. The average correspondences used for $\{1NN, IBMatcher, 1NNAP\}$ are $\{57, 26, 130\}$

6.3. Experiments and Results of Depth Assignment

As stated in the state of the art section 1.2 of the introduction, depth map generation approaches can be categorized into manual, semi-manual, and automatic approaches, with the latter including both in offline as well as real-time variants. Generally speaking, the quality of results decrease in this order, as well as the computational effort to be utilized.

Method	$t_D[ms]$	$t_M[ms]$	#corresp	$t_f[ms]$
IB16 1NN	5.1322	3.0530	55.5952	0.1472
IB16 IBMatcher	4.1167	5.2242	25.9218	0.3603
IB32 1NN	4.5100	5.2206	55.5952	0.1750
IB32 IBMatcher	6.5444	3.1148	26.4269	0.3655
IB64 1NN	5.7565	5.2468	55.5952	0.1979
IB64 IBMatcher	9.4276	3.2411	26.3607	0.4806
MSCD 1NN	358.1339	5.0384	55.9158	6.4950
MSCD 1NN AP	358.3864	21.1687	130.8858	2.9000

Table 6.4.: Computational time in milliseconds for description time t_D, matching time t_M and overall time per feature t_f, plus the average number of correspondences *#corresp*

Attempts to objectively evaluate the quality of approaches in this field of research suffer from the fact, that to this date there are not exactly many well working possibilities for objective evaluation, like datasets of reasonable size along with universally valid proper benchmark measures.

The second, more substantial issue is, that although there are at least some evaluation datasets available, the majority of publications do not evaluate their approaches using a framework. Instead, usually only some example result images are given for subjective evaluation. This makes objective comparison impossible in many cases. Further, it seems that publications delivering high-quality approaches, which might include e.g. learning and offline methods are more likely to consider evaluation frameworks, than approaches which aim into low-cost and real-time approaches.

However, for the evaluation of the lightweight depth map converter, I will try to give both an objective result of an evaluation dataset (though this can also not be regarded as perfect, as stated in the next section), as well as a number of result images for subjective evaluation by the reader, showing good and bad cases. The results presented in the following are part of [45], which holds further details on results.

6.3.1. Experimental Setup of Depth Assignment Results

All depth related computation is implemented in C++ ad a single threaded implementation, using the open source computer vision library OpenCV for filtering and image segmentation.

The vanishing point detector is introduced in [82].

Make 3D dataset

To achieve an objective comparison of the lightweight 2D to 3D conversion to related publications to some extend, I apply the Make 3D dataset, which provides color images with corresponding depth data. It was introduced alongside the depth learning technique presented in [97, 98] and [100].

It has also been used to compare results of monocular depth estimation algorithms to the original work in [47, 99, 46, 67, 64, 63], and [53]. It consists of several hundred of color images with ground truth depth data. All images are 1704×2272 pixels in size, while all ground truth maps are 360×472 in size.

However, it comes with limitations. The depth serving as ground truth is measured by a separate laser ranging device, which is not perfectly aligned with the camera. Thus, some edges that are in a neighborhood of strong depth discontinuities, for example trees against the sky, show up as errors in an evaluation of an estimation, even if a hypothetical estimation was completely correct. Furthermore, on some surfaces the laser reflection is not picked up, leading to inconsistent ground truth maps. Moreover, the employed laser range finder is limited to a maximum range of 81 meters, so higher distances are also not depicted correctly in the ground truth. Further, the focus of this conversion approach is to provide a depth map, which produces a credible and comfortable visual effect after depth-image-based rendering. Related to the dataset, this means, as pointed out in [60], using current 3D display devices, objects with large convergence distances are displeasing to the viewer. Because of this, depth ranges have to be limited anyway so the subjective loss in effect quality is not as severe as it might seem at first, if the precision is not entirely verified at a range of 81 meters.

6.3.2. Depth Assignment Results

The make3D dataset provides results for two error measures. The first is the root mean square error (*RMSE*), defined by

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} (\hat{y}_t - y_t)^2}{n}}, \tag{6.4}$$

where n is the number of measurements, y_t is a measurement and $\hat{y}_t$ is the ground truth value. The second is the relative error δx, defined by

$$\delta x = \frac{\Delta x}{x}, \tag{6.5}$$

where Δx is the absolute error and x is the ground truth value.

Additionally, the results for the signal to noise ratio *SNR* are given, which is defined as

$$\text{SNR} = \frac{P_{signal}}{P_{noise}}, \tag{6.6}$$

plus the *Estimation error relative to value range exE*, which denotes

$$exE = \frac{\Delta x}{x_{max} - x_{min}}. \tag{6.7}$$

In order to compare the results of the conversion approach introduced in this work, table 6.5 shows state of the art results that are reported for the Make 3D dataset.

Algorithm	Main contribution	Relative Error	RMSE
Data baseline	Predict Average	0.698	–
Saxena et al. [97]	Learn Depth: MRF	0.530	16.7m
Saxena et al. [99]	Pointwise MRF	0.458	–
Saxena et al. [99]	Superpixel MRF	0.370	–
Hoiem et al. [47]	Surface Layout	1.423	–
Heitz et al. [46]	Cascaded Models	–	15.4m
Cherian et al. [22]	Ground Plane	–	22m
Liu et al. [67]	Semantic Labels	0.375	–
Li et al. [63]	Feedback Cascades	–	15.2m
Li et al. [64]	theta-MRF	–	15.0m
Karsch et al. [53]	depth transfer	0.361	15.1m
Proposed method	Combination of cues	1.2	20m

Table 6.5.: Results of the approach of this work on the make 3D dataset, compared to the results reported on the make 3D dataset website, if they were reported.

Each row in figure 6.11 show an example result, where the lightweight converter performed considerably good (first and second row), or respectively rather bad (third and fourth row).

Qualitative measures for these examples are given within the left most images. As one can see, the algorithm is clearly limited to scenes, where the basic scenarios of either ramp or tunnel apply. If the image consists of e.g. upright walls, or trees, as in the bad examples, the algorithm falsely chooses the best base scenario out of those it has. This shows one clear starting point for future work, to extend the basic models, it can choose from.

6.3.3. Computation Time

On a standard desktop PC Intel Core *i*7 with 3.40*GHz* and 16.0 *GB* RAM, the adaptive lightweight depth map generation takes 31 ± 5 milliseconds in average.

These roughly 40 millisceconds of computation time correspond to a conversion framerate of 25 Hz. For HD or even 4k or 8k movies, which additionally utilize a higher framerate as well as higher image resolutions as the ones computed here, the conversion thus cannot entirely run in real-time, at least for a standard desktop PC. Yet, it comes close.

The focus of the lightweight depth conversion introduced in this thesis was clearly set on delivering reasonable quality with a high computational efficiency. Nevertheless, there are extensions thinkable, which are likely to boost the qualitative performance.

The most obvious extension is to add motion information based on the I-BRIEF results into the concept, as there exist computationally efficient motion depth cues. Unfortunately, benchmarks for qualitative depth conversion evaluation to this point are mainly image-based. Besides that, further cues are promising, as well as the extension of the information gain of the used cues. In [68], we introduced an elaborate version of vanishing point-based global depth map generation, considering all three points as well as the image objects they are related to. Additionally this work presents concepts for recognizing further image information, like human skin color, grass, sea, and mountains. As further cue, the segmentation results could also be used for an efficient depth from blurr concept. Extending the concept towards learning, the ground truth depth images could be used to used to learn weights for the indicators in the adaptive fusion formula.

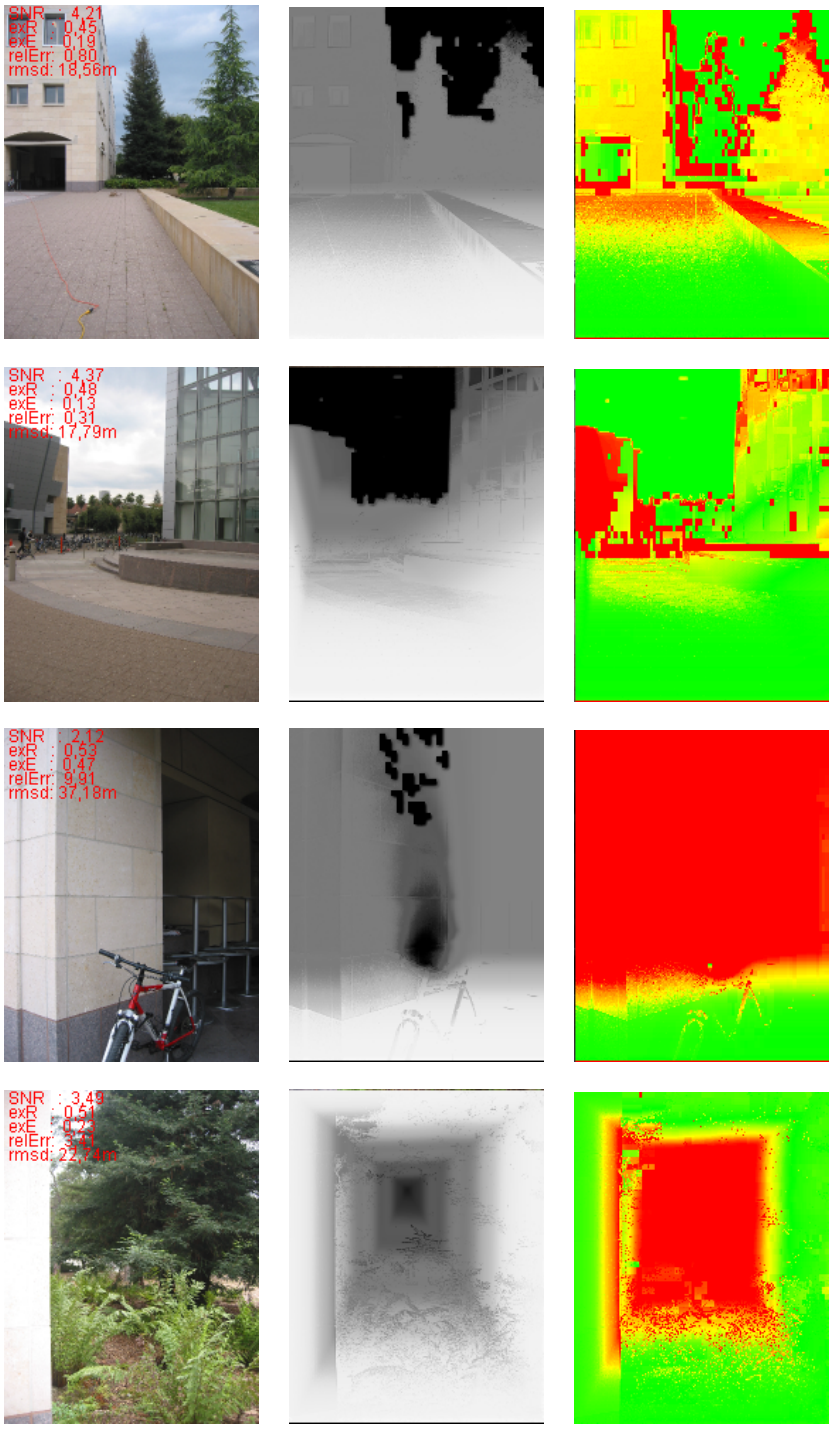


Figure 6.11.: Figure shows two examples from the make 3D dataset, where the lightweight depth map generation delivers proper results (first two rows), respectvely performs rather bad (last two rows). The input images are shown in the left column, the resulting depth maps in the middle, and the right column shows the error of the depth map related to the ground truth (blue being low to red being high error).

Chapter 7.

Conclusion

In this thesis, I investigated the problem 2D to 3D conversion, which aims at the creation of 3D images or image sequences from originally 2D content. Therefore I classify and analyze current approaches in terms of quality and computational efficiency, which range from low quality real time conversion to highest quality and high-priced manual conversion for cinema. Focusing on the task chain of depth map generation, which is the crucial issue in 2D to 3D conversion, I designed three modules, which allow for boosting the quality of automatic real-time conversion systems, while remaining fully automatized and keeping the increase of computational effort low.

The major component is an image analysis algorithm. It utilizes the principles of Swarm Intelligence to provide an image segmentation framework, which delivers an exhaustive image analysis including region appearance, a description of shape and position of object borders, an extraction of straight lines, and of meaningful feature points. Swarm Intelligence algorithms rely on distributed interacting agents, which achieve complex solutions via emergence. Their decentralized structure makes them robust and efficient in computation. Compared to Swarm Intelligence algorithms from the field of optimization, the approach introduced in this thesis is unique in utilizing the principles of Self-Assembly, in which agents link themselves into temporary physical structures. The thesis proves, that this concept is very well suited to the task of image segmentation.

As second module, a description and matching algorithm is established, which allows tracking arbitrarily shaped edges at high computational efficiency. It extends the efficient BRIEF-feature point matching to edges, keeping the principle of extracting binary relations. With this, it achieves very high quality results for image sequences and small baseline scenarios, and reasonable results for wide-baseline scenarios. Combined with the image analysis framework it allows for a dense image matching description at high computational speed.

The third module serves as a proof of concept for efficient depth map generation. I utilize this input for various depth cues and propose a light-weight concept for depth-map creation. The efficiency of the approach results from the combination and approximation of cues, like color statistics, texture, high frequency details, etc., plus a global scene classification allowing for a wide range of scenes. The introduced concept works at a computational speed close to real time and allows for considerable depth map quality.

Moreover, I present example results, analyze all three modules comprehensively in various evaluation frameworks, and compare each module as standalone solution to approaches from the literature. I Further, I give an exhaustive discussion on the results in context of the evaluation benchmarks, as well as in context of the overall task, the depth map generation.

This thesis proves, that Swarm Intelligence principles are well-suited to the problem of comprehensive image analysis. The framework suits the various inputs of depth cues. Yet, it can also be utilized for a number of computer vision applications, which demand segments, edges, lines, or point-like features as input, like e.g. object and shape recognition, SLAM algorithms, or robot vision. The combination of the image analysis with the matching algorithm further allows for arbitrary dense tracking algorithms, or stereo image matching as input to triangulation.

Bibliography

1. C. Akinlar and C. Topal. Edlines: Real-time line segment detection by edge drawing (ed). In *IEEE International Conference on Image Processing*, pp. 2837–2840. 2011.

2. A. Amali Asha, S. Victor, and A. Lourdusamy. Feature extraction in medical image using ant colony optimization: A study. In *International Journal on Computer Science and Engineering*, 3(2), pp. 714–721, Feb. 2011.

3. C. Anderson, G. Theraulaz, and J.L. Deneubourg. Self-assemblages in insect societies. In *Insectes Sociaux*, 49(2), pp. 99–110, 2002.

4. N. Ansari and E.J. Delp. On detecting dominant points. In *Pattern Recognition*, 24(5), pp. 441–451, 1991.

5. N. Apostoloff and A. Fitzgibbon. Learning spatiotemporal t-junctions for occlusion detection. In *Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pp. 553–559. 2005.

6. P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. In *Transactions on Pattern Analysis and Machine Intelligence*, 33(5), pp. 898–916, May 2011.

7. A.V. Baterina and C. Oppus. Image edge detection using ant colony optimization. In *WSEAS Transactions on Signal Processing*, 6(2), pp. 58–67, April 2010.

8. H. Bay, V. Ferrari, and L. Van Gool. Wide-baseline stereo matching with line segments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 329–336. 2005.

9. G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. In *NATO Advanced Workshop on Robots and Biological Systems*, pp. 26–30. July 1989.

10. T. Bichlmeier. *Salient Edge Detection and Matching in Images*. Diploma thesis, Lehrstuhl für Datenverarbeitung, Technische Universität München, 2013.

11. M. Bichsel and A.P. Pentland. A simple algorithm for shape from shading. In *Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 459–465. 1992.

12. J. Bishop. Stochastic searching networks. In *Proc. 1st IEE Conf. on Artificial Neural Networks, London, UK*, pp. 329–331. 1989.

13. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., 1999.

14. K. Bowyer, C. Kranenburg, and S. Dougherty. Edge detector evaluation using empirical roc curves. In *Computer Vision and Image Understanding*, 84(1), pp. 77 – 103, 2001.

15. M. Calonder, V. Lepetit, M. Ozuysal et al.. BRIEF: Computing a local binary descriptor very fast. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7), pp. 1281–1298, 2012.

16. M. Calonder, V. Lepetit, C. Strecha et al.. BRIEF: Binary robust independent elementary features. In *European Conference on Computer Vision*, pp. 778–792. 2010.

17. K. Cannons. *A review of visual tracking*. Technical Report, York University, 2008.

18. J. Canny. A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), pp. 679–698, June 1986.

19. B. Catanzaro, B.Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. In *IEEE Computer Vision*, pp. 2381–2388. October 2009.

20. J. Chang and J. Fisher. Efficient mcmc sampling with implicit shape representations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2081 –2088. june 2011.

21. J. Chen, T. Pappas, A. Mojsilovic, and B. Rogowitz. Adaptive perceptual color-texture image segmentation. In *Transactions on Image Processing*, 14(10), pp. 1524–1536, October 2005.

22. A. Cherian, V. Morellas, and N. Papanikolopoulos. Accurate 3D ground plane estimation from a single image. In *International Conference on Robotics and Automation*, pp. 2243–2249. 2009.

23. R. Cipolla, Y. Okamoto, and Y. Kuno. Robust structure from motion using motion parallax. In *Fourth International Conference on Computer Vision*, pp. 374–382. 1993.

24. D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), pp. 603 –619, 2002.

25. T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1124–1131. june 2005.

26. J. Crowley, P. Stelmaszyk, and C. Discours. Measuring image flow by tracking edge-lines. In *International Conference on Computer Vision*, pp. 658–664. 1988.

27. W. Cui, Z. Guan, and Z. Zhang. An improved region growing algorithm for image segmentation. In *International Conference on Computer Science and Software Engineering*, pp. 93–96. Dec. 2008.

28. S. Das and S. Sil. Kernel-induced fuzzy clustering of image pixels with an improved differential evolution algorithm. In *Information Sciences*, 180(8), pp. 1237–1256, April 2010.

29. P. Denis, J.H. Elder, and F.J. Estrada. *Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery*, pp. 197–210. Springer Berlin Heidelberg, 2008.

30. R. Deriche and O. Faugeras. Tracking line segments. In *Image and Vision Computing*, 8(4), pp. 261–270, November 1990.

31. P. Dollár and C.L. Zitnick. Structured forests for fast edge detection. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, pp. 1841–1848. 2013.

32. M. Dorigo. *Optimization, learning and natural algorithms*. Ph.D. thesis, Politecnico di Milano, Italy, 1992.

33. E. Eade and T. Drummond. Edge landmarks in monocular SLAM. In *Image and Vision Computing*, 27(5), pp. 588–596, 2009.

34. S.A. Etemad and T. White. An ant-inspired algorithm for detection of image edge features. In *Applied Soft Computing*, 11(8), pp. 4883–4893, 2011.

35. P.F. Felzenszwalb and D.P. Huttenlocher. Efficient graph-based image segmentation. In *International Journal of Computer Vision*, 59(2), pp. 167–181, September 2004.

36. D. Feng, S. Wenkang, C. Liangzhou, D. Yong, and Z. Zhenfu. Infrared image segmentation with 2-d maximum entropy method based on particle swarm optimization (pso). In *Pattern Recognition Letters*, 26(5), pp. 597–603, 2005.

37. L. Fisher. *Schwarmintelligenz: Wie einfache Regeln Großes möglich machen*. Eichborn Verlag, 2010.

38. A. Ford and A. Roberts. Colour space conversions. In *Westminster University, London*, pp. 1–31, 1998.

39. M. Galabov. 2D to 3D conversion algorithms. In *RCITD, Research Conference In Technical Disciplines*, pp. 91–93. 2014.

40. S. Gauglitz, T. Höllerer, and M. Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. In *International Journal of Computer Vision*, 94(3), pp. 335–360, 2011.

41. L. Gruber et al.. The city of sights: Design, construction, and measurement of an augmented reality stage set. In *IEEE International Symposium on Mixed and Augmented Reality*, pp. 157–163. 2010.

42. K. Hammouche, M. Diaf, and P. Siarry. A multilevel automatic thresholding method based on a genetic algorithm for a fast image segmentation. In *Computer Vision and Image Understanding*, 109(2), pp. 163–175, February 2008.

43. C. Harris and C. Stennet. RAPiD – a video-rate object tracker. In *British Machine Vision Conference*, pp. 73–77. 1990.

44. S. Hasinoff and K. Kutulakos. Light-efficient photography. In *Transactions on Pattern Analysis and Machine Intelligence*, 33(11), pp. 2203–2214, 2011.

45. M. Heinrich. *An adaptive approach to high performance depth map generation*. Bachelor's thesis, Lehrstuhl für Datenverarbeitung, Technische Universität München, 2014.

46. G. Heitz, S. Gould, A. Saxena, and D. Koller. Cascaded classification models: Combining models for holistic scene understanding. In *Advances in Neural Information Processing Systems*, volume 1, pp. 1–8. Vancouver, BC, 2008.

47. D. Hoiem, A.A. Efros, and M. Hebert. Recovering surface layout from an image. In *International Journal of Computer Vision*, 75(1), pp. 151–172, 2007.

48. M.K. Hu. Visual pattern recognition by moment invariants. In *IRE Transactions on Information Theory*, 8(2), pp. 179–187, 1962.

49. X. Huang, L. Wang, J. Huang, D. Li, and M. Zhang. A depth extraction method based on motion and geometry for 2D to 3D conversion. In *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, volume 3, pp. 294–298. Nov 2009.

50. I. Ideses, L.P. Yaroslavsky, and B. Fishbain. Real-time 2D to 3D video conversion. In *Journal of Real-Time Image Processing*, 2(1), pp. 3–9, 2007.

51. P. Isola, D. Zoran, D. Krishnan, and E.H. Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*. 2014.

52. K. Karsch, C. Liu, and S.B. Kang. Depth transfer: Depth extraction from video using non-parametric sampling. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), pp. 2144–2158, 2014.

53. K. Karsch, C. Liu, and S.B. Kang. Depth extraction from video using non-parametric sampling. In *European Conference on Computer Vision*, pp. 775–788. Springer, 2012.

54. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948. 1995.

55. F. Keyrouz, U. Kirchmaier, and K. Diepold. Three dimensional object tracking based on audiovisual fusion using particle swarm optimization. In *11th International Conference on Information Fusion, FUSION 2008, Cologne, Germany, June 30 - July 3, 2008*, pp. 1–5. 2008.

56. U. Kirchmaier, S. Hawe, and K. Diepold. Dynamical information fusion of heterogeneous sensors for 3D tracking using particle swarm optimization. In *Information Fusion*, 12(4), pp. 275–283, 2011.

57. U. Kirchmaier, S. Hawe, and K. Diepold. A swarm intelligence inspired algorithm for contour detection in images. In *Applied Soft Computing*, 13(6), pp. 3118–3129, June 2012.

58. U. Kirchmaier, S. Hawe, and K. Diepold. A line detection and description algorithm based on swarm intelligence. In *Proc. SPIE*, pp. 753–757. 2010.

59. E. Lakehal. A swarm intelligence based approach for image feature extraction. In *International Conference on Multimedia Computing and Systems*, pp. 31–35. april 2009.

60. M. Lang, A. Hornung, O. Wang, S. Poulakos, A. Smolic, and M. Gross. Nonlinear disparity mapping for stereoscopic 3D. In *ACM Transactions on Graphics*, 29(4), pp. 75–80, 2010.

61. C.Y. Lee, J.J. Leou, and H.H. Hsiao. Saliency-directed color image segmentation using modified particle swarm optimization. In *Signal Processing*, 92(1), pp. 1–18, 2012.

62. C.H. Lee and A. Rosenfeld. Improved methods of estimating shape from shading using the light source coordinate system. In *Artificial Intelligence*, 26(2), pp. 125–143, 1985.

63. C. Li, A. Kowdle, A. Saxena, and T. Chen. A generic model to compose vision modules for holistic scene understanding. In *Trends and Topics in Computer Vision*, pp. 70–85. Springer, 2012.

64. C. Li, A. Saxena, and T. Chen. θ-mrf: Capturing spatial and semantic structure in the parameters for scene understanding. In *Advances in Neural Information Processing Systems*, pp. 549–557. 2011.

65. L. Li and D. Li. Fuzzy entropy image segmentation based on particle swarm optimization. In *Progress in Natural Science*, 18(9), pp. 1167–1171, 2008.

66. J.J. Lim, C.L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *Conference Computer Vision and Pattern Recognition*, pp. 3158–3165. 2013.

67. B. Liu, S. Gould, and D. Koller. Single image depth estimation from predicted semantic labels. In *Conference on Computer Vision and Pattern Recognition*, pp. 1253–1260. IEEE, 2010.

68. M. Lück and U. Kirchmaier. *Geometrically based depth assignment using a single image*. Technical report, Lehrstuhl für Datenverarbeitung, Technische Universität München, 2016.

69. W. Ma and B.S. Manjunath. Edgeflow: a technique for boundary detection and image segmentation. In *Transactions on Image Processing*, 9(8), pp. 1375–1388, 2000.

70. J. Malik and R. Rosenholtz. Computing local surface orientation and shape from texture for curved surfaces. In *International Journal of Computer Vision*, 23(2), pp. 149–168, 1997.

71. D. Marr and E. Hildreth. Theory of edge detection. In *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167), pp. 187–217, 1980.

72. D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision*, pp. 416–423. July 2001.

73. D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. In *Transactions on Pattern Analysis and Machine Intelligence*, 26(5), pp. 530–549, May 2004.

74. J. Meltzer and S. Soatto. Edge descriptors for robust wide-baseline correspondence. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8. 2008.

75. K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *Transactions on Pattern Analysis and Machine Intelligence*, 27(10), pp. 1615–1630, 2005.

76. Y. Mingqiang, K.K. Idiyo, and R. Joseph. *A Survey of Shape Feature Extraction Techniques*, pp. 43–90. IN-TECH, 2008.

77. R.J. Mullen, D.N. Monekosso, and P. Remagnino. Ant algorithms for image feature extraction. In *Expert Systems with Applications*, 40(11), pp. 4315–4332, 2013.

78. R.J. Mullen, D. Monekosso, S. Barman, P. Remagnino, and P. Wilkin. *Artificial Ants to Extract Leaf Outlines and Primary Venation Patterns*, pp. 251–258. Springer Berlin Heidelberg, 2008.

79. V. Namboodiri and S. Chaudhuri. Recovery of relative depth from a single observation using an uncalibrated (real-aperture) camera. In *Conference on Computer Vision and Pattern Recognition*, pp. 38–68. 2008.

80. S.G. Narasimhan and S.K. Nayar. Vision and the atmosphere. In *International Journal of Computer Vision*, 48(3), pp. 233–254, 2002.

81. P. Neubert, P. Protzel, T. Vidal-Calleja et al.. A fast visual line segment tracker. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 353–360. 2008.

82. M. Nieto and L. Salgado. Real-time robust estimation of vanishing points through nonlinear optimization. In *IS&T/SPIE Int. Conf. on Real-Time Image and Video Processing*, volume 7724, pp. 1–14. 2010.

83. T.L. Ooi, B. Wu, and Z.J. He. Distance determined by the angular declination below the horizon. In *Nature*, 414(6860), pp. 197–200, 2001.

84. R.P. O'Shea, S.G. Blackburn, and H. Ono. Contrast as a depth cue. In *Vision research*, 34(12), pp. 1595–1604, 1994.

85. G. Papari and N. Petkov. Edge and line oriented contour detection: State of the art. In *Image and Vision Computing*, 29(2-3), pp. 79–103, 2011.

86. K.M. Passino. Bacterial foraging optimization. In *Int. J. Swarm. Intell. Res.*, 1(1), pp. 1–16, 2010.

87. A. Pentland. Shape information from shading: a theory about human perception. In *Spatial vision*, 4(2-3), pp. 2–3, 1989.

88. D. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm. technical note. In *Manufacturing Engineering Centre, Cardiff University, UK*, pp. 1–57, 2005.

89. M. Polak, H. Zhang, and M. Pi. An evaluation metric for image segmentation of multiple objects. In *Image and Vision Computing*, 27(8), pp. 1223–1227, July 2009.

90. M.T. Pourazad, P. Nasiopoulos, and A. Bashashati. Random forests-based 2D-to-3D video conversion. In *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, pp. 150–153. 2010.

91. M.T. Pourazad, P. Nasiopoulos, and R.K. Ward. An h. 264-based scheme for 2D to 3D video conversion. In *IEEE Transactions on Consumer Electronics*, 55(2), pp. 742–748, 2009.

92. W.K. Pratt. *Digital Image Processing*. 3rd edition. John Wiley & Sons Interscience, 2001.

93. Y. Ramadevi, T. Sridevi, B. Poornima, and B. Kalyani. Segmentation and object recognition using edge detection techniques. In *International Journal of Computer Science & Information Technology*, 2(6), pp. 153–161, December 2010.

94. C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, 21(4), pp. 25–34, 1987.

95. G.G. Rigatos. Multi-robot motion planning using swarm intelligence. In *International Journal of Advanced Robotic Systems*, 5(2), pp. 139–144, 2008.

96. S. Saatchi and C.c. Hung. *Swarm intelligence-based image segmentation*, pp. 163–178. Itech Education and Publishing, December 2007.

97. A. Saxena, S.H. Chung, and A.Y. Ng. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems*, volume 18, pp. 1–8. 2005.

98. A. Saxena, S.H. Chung, and A.Y. Ng. 3-d depth reconstruction from a single still image. In *International Journal of Computer Vision*, 76(1), pp. 53–69, 2008.

99. A. Saxena, M. Sun, and A.Y. Ng. Learning 3-d scene structure from a single still image. In *11th International Conference on Computer Vision*, pp. 1–8. IEEE, 2007.

100. A. Saxena, M. Sun, and A.Y. Ng. Make3D: Learning 3D scene structure from a single still image. In *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, 31(5), pp. 824–840, May 2009.

101. D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *International Journal of Computer Vision*, 47(1), pp. 7–42, May 2002.

102. C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. In *International Journal of Computer Vision*, 37(2), pp. 151–172, 2000.

103. C. Schmid and A. Zisserman. Automatic line matching across views. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 666–671. 1997.

104. N. Senthilkumaran and R. Rajesh. Edge detection techniques for image segmentation – a survey of soft computing approaches. In *International Journal of Recent Trends in Engineering*, 1(2), pp. 250–254, May 2009.

105. M. Setayesh, M. Zhang, and M. Johnston. Edge detection using constrained discrete particle swarm optimisation in noisy images. In *IEEE Evolutionary Computation (CEC)*, pp. 246–253. June 2011.

106. E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. In *Nature*, 42(7104), pp. 810–813, 2006.

107. T. Smith and M. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, 147(1), pp. 195–197, 1981.

108. A. Smolic, P. Kauff, S. Knorr, A. Hornung, M. Kunter, M. Muller, and M. Lang. Three-dimensional video postproduction and processing. In *Proceedings of the IEEE*, 99(4), pp. 607–625, 2011.

109. K. Śorensen. Metaheuristics—the metaphor exposed. In *International Transactions in Operational Research*, 22(1), pp. 3–18, 2015.

110. X.Y. Stella, H. Zhang, and J. Malik. Inferring spatial layout from a single image via depth-ordered grouping. In *Conference on Computer Vision and Pattern Recognition*. 2008.

111. R. Szeliski. *Computer Vision: Algorithms and Applications*. New York: Springer-Verlag, 2010.

112. W.J. Tam, C. Vázquez, and F. Speranza. Three-dimensional TV: A novel method for generating surrogate depth maps using colour information. In *IS&T/SPIE Electronic Imaging*, pp. 72371–72371. International Society for Optics and Photonics, 2009.

113. S. Thilagamani and N. Shanthi. A survey on image segmentation through clustering. In *International Journal of Research and Reviews in Information Sciences*, 1(1), pp. 16–19, March 2011.

114. C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE Sixth International Conference on Computer Vision*, pp. 839–846. 1998.

115. S.F. Tsai, C.C. Cheng, C.T. Li, and L.G. Chen. A real-time 1080p 2D-to-3D video conversion system. In *ICCE, IEEE International Conference on Consumer Electronics*, pp. 803–804. 2011.

116. T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. In *Foundation and Trends in Computer Graphics and Vision*, 3(3), pp. 177–280, 2008.

117. R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. In *Transactions on Pattern Analysis and Machine Intelligence*, 29(6), pp. 929–944, June 2007.

118. J. Van De Weijer and C. Schmid. Coloring local feature extraction. In *European Conference on Computer Vision*, pp. 334–348. Springer, 2006.

119. R. Veltkamp. Shape matching: Similarity measures and algorithms. In *IEEE International Conference on Shape Modeling and Applications*, pp. 188–197. 2001.

120. H. Wang and J. Oliensis. Generalizing edge detection to contour detection for image segmentation. In *Computer Vision and Image Understanding*, 114(7), pp. 731–744, 2010.

121. L. Wang, U. Neumann, and S. You. Wide-baseline image matching using line signatures. In *IEEE International Conference on Computer Vision*, pp. 1311–1318. 2009.

122. Z. Wang, W. F., and Z. Hu. MSLD: A robust descriptor for line matching. In *Pattern Recognition*, 42(5), pp. 941–953, 2009.

123. Z. Wang, H. Liu, and F. Wu. Hld: A robust descriptor for line matching. In *IEEE International Conference on Computer-Aided Design and Computer Graphics*, pp. 128–133. 2009.

124. B. Ward, S. Bing Kang, and E.P. Bennett. Depth Director: A System for Adding Depth to Movies. In *Computer Graphics and Applications, IEEE*, 31(1), pp. 36–48, 2011.

125. Q. Wei. *Converting 2D to 3D: A Survey*. Research assignment, TU Delft, 2005.

126. J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. In *Science*, 343(6172), pp. 754–758, 2014.

127. R. Xiaofeng and L. Bo. Discriminatively trained sparse code gradients for contour detection. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 584–592. Curran Associates, Inc., 2012.

128. A. Yilmaz, O. Javed, and M. Shah. Object tracking: a survey. In *ACM Computing Surveys*, 38(4), pp. 1–45, 2006.

129. S. Yin, H. Dong, G. Jiang, L. Liu, and S. Wei. A novel 2D-to-3D video conversion method using time-coherent depth maps. In *Sensors*, 15(7), pp. 15246–15264, 2015.

130. W. Yong and C. Jun. Using ant swarm intelligence for data clustering analysis. In *IEEEComputer Science and Information Technology*, pp. 429–432. August 2009.

131. D. Zhang and G. Lu. Review of shape representation and description techniques. In *Pattern Recognition*, 37(1), pp. 1–19, 2004.

132. L. Zhang and R. Koch. Line matching using appearance similarities and geometric constraints. In *Pattern Recognition*, Lecture Notes in Computer Science, pp. 236–245. Springer Berlin Heidelberg, 2012.

133. L. Zhang and R. Koch. Vanishing points estimation and line classification in a manhattan world. In *Asian Conference on Computer Vision*, pp. 38–51. Springer, 2013.

134. Q. Zheng and R. Chellappa. Estimation of illuminant direction, albedo, and shape from shading. In *Conference on Computer Vision and Pattern Recognition*, pp. 540–545. 1991.

135. S. Zhuo and T. Sim. On the recovery of depth from a single defocused image. In *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, CAIP '09, pp. 889–897. 2009.

Appendix A.

Test Parameters

The following are the standard parameters used for all experiments for each individual component. Except a difference is concretely mentioned in the experiment description, then these values are employed instead of the ones mentioned here.

A.1. SISeg

1. Gaussian prior image smoothing variance σ = 1.5,
2. Gaussian prior image smoothing Kernel size = 3,
3. Line-condition threshold 1 τ^{O} = 0.5
4. Line-condition threshold 2 τ^{o} = 0.7
5. Distance sum weight 1 w_1^s = 2
6. Distance sum weight 2 w_2^s = 1
7. NMS maximum threshold τ^{max} = 20,
8. NMS minimum threshold τ^{min} = 6,
9. NMS relative threshold τ^{rel} = 1.1,
10. Region fusion threshold τ^{reg} = 0.25.
11. Maximum number of iterations τ^{t} = 25.

A.2. Anchor Points

1. Gaussian standard deviation is fixed to σ_{AP} = 3.65.

2. Gaussian Kernel size is fixed to 21.
3. Curvature threshold set to $\epsilon_{AP} = 0.01$.

A.3. I-BRIEF

1. Smoothing is done via integral images,
2. Kernel size of box filtering is set to 9,
3. Descriptor size in bytes $k = 64$,
4. Method is set to GII,
5. Patch size $S = 48$,
6. Suppression area height $I_s = 3$.

A.4. MSLD and MSCD

1. number of sub-regions is set to 9,
2. size of the sub-region is set to a window of size $(5, 5)$,
3. threshold for maximal values is set to 0.4 .

A.5. Matching Strategies

1. 1NN $I_{sr} = 100$,
2. IB Matching $I_{sr} = 100$, $\tau = 0.9$, $r = 0.5$,
3. NNDR only used in Section 6.2.3. For boat and graf sequences $r_{NNDR} = 0.8$, for wall sequence $r_{NNDR} = 0.6$.