Extending Image Segmentation by Ascertaining Object Permanence Using a Layer-based Memory

Diplomarbeit, vorgelegt von

Johannes Dörr

aus Kiel, angefertigt im
Dritten Physikalischen Institut der
Georg-August-Universität Göttingen

2011

Acknowledgments

At first, I would like to thank Prof. Dr. Florentin Wörgötter without whom this work would not have been possible. He gave me the chance to independently solve the issues of object permanence which we frequently talked about and allowed me to pursue my own approaches until the very end. These discussions and many helpful suggestions pointing into the right direction have fostered the outcome of this new method.

A big thank you also goes to the whole vision group, namely Alexey Abramov, Eren Aksoy and Babette Dellen. I truly enjoyed working with you on the vision system and I am grateful that you included me in your research from the first day on. Hopefully, the *vision machine* will grow and, in some time, power numerous camera-based robotic systems. Notably, I want to thank Alexey for spending so much time on reviewing my thesis and giving such valuable feedback. In addition, a great thanks goes to the whole group around Florentin for the fantastic time! I will definitely miss the Kicker games after lunch as well as the cheerful meetings for dinner and beer in the evenings.

Furthermore, I would like to thank my friends and fellow students Waldemar Kornewald and Thomas Wanschik. Besides discussing work-related problems and supporting me in my studies, we planed our future venture which was quite inspiring and I am looking forward to this new work. At this point, again thanks to Florentin who will counsel our upcoming project.

Another special thanks goes to my girlfriend Nina Hentschel not only for her support and patience, especially in the last, stressful month but also for cross-reading my thesis. Last but not least, I am grateful to my parents Galia and Hans Dörr for supporting me in all I am doing. I am glad you gave me the opportunity to study in Göttingen and for letting me do the things I love to do.

Again, thanks to all of you for everything you have done for me!

Contents

Chapter 1

Introduction

1.1 Motivation

In the field of computer vision, the development of algorithms that enable a robot to comprehend what it sees just as humans make sense of what they perceive, is an extremely complex task. The raw data obtained using one or more cameras contains a large amount of data which is inapplicable as input for logic or learning algorithms. For that reason, recognizing objects and tracking their movement along the image sequence is one of the main issues in robotics. There has been plenty of work concerning these topics as shown in section 1.2. Many algorithms are optimized for tracking specific objects such as cars or humans in order to recognize certain situations in traffic scenes or interactions between people. A very common method that allows a robust tracking is the creation of an appearance model of the target object and then finding a position in the image that fits this model best.

However, a tracking method for unknown objects cannot rely on any model. In order to solve this problem, it is useful to find regions of pixels with similar color which represent cohesive units in the scene. Finding subparts in an image with similar color values is known as *image segmentation*.[1] This method can also be used for tracking by maintaining consistent labels of segments in a sequence. The advantage is that all parts of the image are tracked and not only a predefined selection of elements. The disadvantage, however, is that the found segments cannot be seen as an equivalent to real objects: Firstly, it is very probable that objects are represented by multiple segments, which requires further reasoning to treat multiple segments as one object or, in other words, one needs to decide whether two segments correspond to one or two objects. Secondly, ordinary color

[1] Segmentation in general is dividing and image into subparts with respect to a certain measure of homogeneity e.g. the similarity of the color.

segmentation is not able to handle occlusions. Particularly, if one segment moves over another, the segment in the background can split into two parts since they are not connected anymore on a 2-dimensional level. Obviously, splitting of an object needs to be dealt with before further working with such an algorithm.

This work presents a layer-based method which extends image segmentation by ascertaining *object permanence* [5]. This is done by implementing a segment memory which stores the shape of each segment independently. In case of an occlusion the original shape is preserved and splits can be resolved. The content of each layer is updated during the image sequence so that segments which have been partially occluded in the initial frame are completed as soon as additional parts become visible. Since each segment can move, rotate and perspectively distort, a homography matrix is computed from frame to frame that transforms the corresponding layer to the current view. For that purpose, *optical flow* which provides point correspondences between adjacent frames, and line correspondences obtained by *Hough transform* which is being optimized for binary images are used. The latter requires knowledge about the relative depths of the segments which is gained by analyzing their relative movements.

Furthermore, correlations between segments are recognized in order to group them to objects. This enables the system to track completely hidden segments by observing the visible parts of the object to which they belong. While some previous works have shown to also be able to handle complete occlusions by assuming that a hidden object moves similarly to its occluder (see section 1.2), the presented algorithm handles perspective deformations rather than only affine motions.

The proposed framework has to be seen as a postprocessing unit that works with the segmentation results and thereby greatly influences the final results. The segmentation algorithm itself can be interchanged. However, for this work, the framework proposed in Abramov et al. [2] was used which performs in real-time. As a result, the presented framework renders an image segmentation similar to the input but with partly relabeled segments securing object permanence.

The purpose of this work is not tracking itself but rather to obtain an abstract description of the scene. As already mentioned, recognizing actions which are observed by a camera is an intricate topic that cannot be solved by using the pixel data directly. In fact, a more abstract representation of each frame, and by that, of the whole sequence is needed. Only then, algorithms that work on a small number of abstract descriptors (symbols) can be applied. Finding this reduced representation without prior knowledge of the data (model free) thus represents a major challenge in cognitive-vision applications. This problem is also known as the signal-symbol gap [27].

Even though the focus of this work lies on the mentioned tracking problems, the developed framework is also used for the synthesis of semantic scene descrip-

tions out of image sequences. The proposed framework can analyze the gained image segments and derive *semantic scene graphs* that store the spatial relations of the segments. Those graphs are then used to classify the observed actions (see section 2.1). By that, it is possible to find identical actions in different movies and furthermore recognize objects that play the same role. As this classification process was published in Aksoy et al. [3], the current study, especially its results section, is focused on the improvement of segmentation. However, since the spatial relations of segments are partly used by the proposed algorithms providing object permanence, the derivation of the semantic scene graphs is also part of this work.

1.2 Related Work

There has been plenty of work on object tracking in image sequences. This section recapitulates the different approaches for occlusion problems that are known from the literature.

Pfinder [46] is a real-time system for tracking a single person and interpreting its behavior. The system uses a statistical model of color and shape to obtain a 2D representation of the different body parts. Starting with an empty scene to establish the static background, the system finds hands and head by analyzing the 2D contour of the person entering the scene. The found body parts are then tracked by measuring the likelihood of each image pixel to belong to the corresponding model. The system is stable against partial occlusions as it automatically removes or adds hidden or reappearing body parts.

The Hydra system [16, 15] is able to track several people even when they are in a groups. A requirement is that each head is part of the group's silhouette which is used to create a shape model by using corners of the convex hull and the projection histogram. It is able to keep track of people in a group with partial occlusions but has problems with large rotations causing a significant change of shape. This problem was addressed by McKenna et al. [31] making strong use of color information (in form of color histograms) rather than shape in order to create an appearance model. Elgammal et al. [12] also track like that but divide the body into subparts with their own color histograms for higher accuracy. Furthermore, they derive, like in my work, a relative depth relation between objects by including it to the likelihood measurement. Hence, an explicit reasoning about occlusions is done here.

Khan et al. [23] presented a method that can identify people after complete occlusion, however only if they moved little and did not change in color.

Many systems, like the ones mentioned, make hard decisions concerning assigning consistent labels to the tracked objects, Marques et al. [30] on the con-

trary take a statistical approach. On top of low level tracking of regions (based on background separation), a second layer is implemented which manages the labeling of those regions using a statistical model based on Bayesian networks. It can handle partial and total occlusions, group mergings and splittings in complex scenes by always providing the most probable label configuration.

Koller et al. [26] use *active contours* to track cars on a highway. The tracking process is divided into two steps: The movement of each car is assumed to be linearly approximable by an affine motion, e.g. translation and scaling. The deformation of the object (due to perspective or occlusion) is independently handled by the active contouring. Since the movement is restricted to a plane (the street) and the camera is fixed, the depth order of the tracked objects can simply be derived by comparing their y-coordinates since far away objects are located in the upper part of the image. By that, occluded parts can easily be ignored in the shape estimation which improves the results.

All of the mentioned works use a foreground/background segmentation as a preprocessing step which is known to yield quite robust results. This is done either by subtracting an image of the raw background or by motion detection which requires the environment to be more or less static and the camera to be fixed.

By Papadakis et al. [32] (inspired by [29, 8]), a contour-based method for tracking multiple objects is proposed that does not assume a static background. However, a manual initialization of the objects to be tracked is needed. Each object is assumed to consist of a set of pixels which can be visible or occluded. In each frame, a prediction of their position in the next frame is made using the mean velocity of the object's pixels and a dynamic model. The final tracking is then realized as a minimization of an energy function using Graph Cuts [7]. The system can handle even total occlusions as long as the corresponding object fits the dynamic model.

Most of the above mentioned approaches rely on partial observations which makes them inapplicable in situations with complete occlusions, especially when their assumption about small and consistent movements does not hold. By Huang et al. [20], a framework is proposed that handles occlusions under unpredictable motions. This is done by assuming that hidden objects still exist in the close proximity of their occluders. The tracking is divided into two parts: The region-level finds foreground regions (using background subtraction) and tracks their movements during the sequence. Each region can contain several objects which are tracked on the object-level: Each region constrains the admissible location of the contained objects. In case of occlusion, the corresponding regions merge to one. The now occluded object can still be tracked by means of the including region and identified as soon as it becomes visible again. Obviously, hidden objects within one group have to be well distinguishable by the object model.

For that reason, the authors include color histogram, spatial distribution and occlusion relationships into the model.

The aim of the previously mentioned method is similar to that of this work concerning total occlusions. However, the position of a currently occluded object is known only to be within a corresponding region, no exact position is provided by the system. Furthermore, that and all other methods mentioned here only consider affine movements while assuming that the object's shapes stay more or less unchanged. In contrast to my work, they do not explicitly handle perspective deformations caused by the 3D structure of the objects which is fine though if the distances to the camera are large enough.

The work of Wang et al. [45] is only weakly related to the classical tracking problem. However, just like this work, they represent coherently moving regions of an image sequence by independent layers that are warped over time according to the velocity map.

Jojic et al. [22] and Tao et al. [42] represent videos as so-called dynamic layers that can contain non-rigid objects, hence flexible data like the legs of a walking person. Zhou et al. [50] extend this by explicitly inferring the depth order of the layers. An essential part of these works is finding the decomposition into layers and the main intention is using those layers for tasks like postproduction, e.g. inserting artificial objects into a camera sequence.

1.3 Structure of this Work

Chapter 1 gives an introduction to the topic and an overview of the related work concerning tracking with occlusions. Chapter 2 includes a short introduction to *action recognition* which is a main application of the proposed framework (section 2.3). Moreover, it contains information about the used image segmentation algorithm (section 2.2) and optical flow estimation (section 2.3) used by the framework as external modules which means that no modifications are done on them in this work. Furthermore, a short introduction to *homogeneous coordinates* is given (section 2.4) since they play an important role in this thesis.

Before all components of the proposed system are described in detail, an outline is given in section 1.4. All parts of the layer framework are described in chapter 4. Its first sections describe the essential parts of the system while section 4.8 introduces some extensions which increase the system's accuracy. The layer framework is then used in chapter 5 to recognize segment splittings and to track hidden segments.

The performance of the system is presented in chapter 6. A conclusion and an outlook concerning still unsolved problems and possible approaches can be found in chapter 7.

1.4 Overview of the System

For a better overview and better navigation through this work, Fig. 1.1 shows an graphical outline of the system.

The images obtained by a camera are divided into segments by the algorithm described in section 2.2. Furthermore, optical flow is estimated between subsequent frames (section 2.3). The implementations of both modules have to be seen as independent from the proposed framework and could be replaced by other algorithms.

The proposed framework is based on storing segments in different layers. For each new segment, hence for all segments in the initial frame and for any other segment appearing during the image sequence, a new layer is initialized (section 4.1).

Secondly, the segmentation results are analyzed in order to obtain the spatial relations of the segments (section 3). Those relations can be represented as a graph.

Furthermore, for each segment, the boundary lines are established in case there are any (section 4.4). In case of occlusion, boundary lines found for a segment can correspond to a foreground object instead of the object of the segment. The framework has some mechanisms to remove those "wrong lines" as described in section 4.8.1. Some of them make use of the segment relations.

The established boundary lines and the data obtained by optical flow are used to estimate, for each segment, a homography matrix between the previous and the current frame (section 4.3). This transformation is used by the framework to map the segment memory, thus the corresponding layer, to the current frame as described in section 4.2. The precision of this transformation estimation depends on the configuration of the segment's pixels. Particularly, if the layer contains lots of pixels that are, due to occlusion, currently invisible in the frame, the transformation estimate might be erroneous. However, the proposed system computes the expected deviation in order to establish the range of trustable values (section 4.10).

During the homography estimation, the framework uses the RANSAC algorithm to find outliers (section 4.3.5) in the optical flow values which is essential for a stable performance. "Wrong lines" which have not been recognized so far, are also excluded that way (section 4.8.1).

After the transformation matrices have been established, the framework analyzes the movements of the layers to obtain their relative depths (section 4.7). This is done by finding regions that became visible after occlusions or vice-versa. Both situations have to be caused by an occluder which naturally has to be above the occluded entity. As soon as the depth relation between two segments is known, "wrong lines" can easily be ignored in all subsequent frames (section

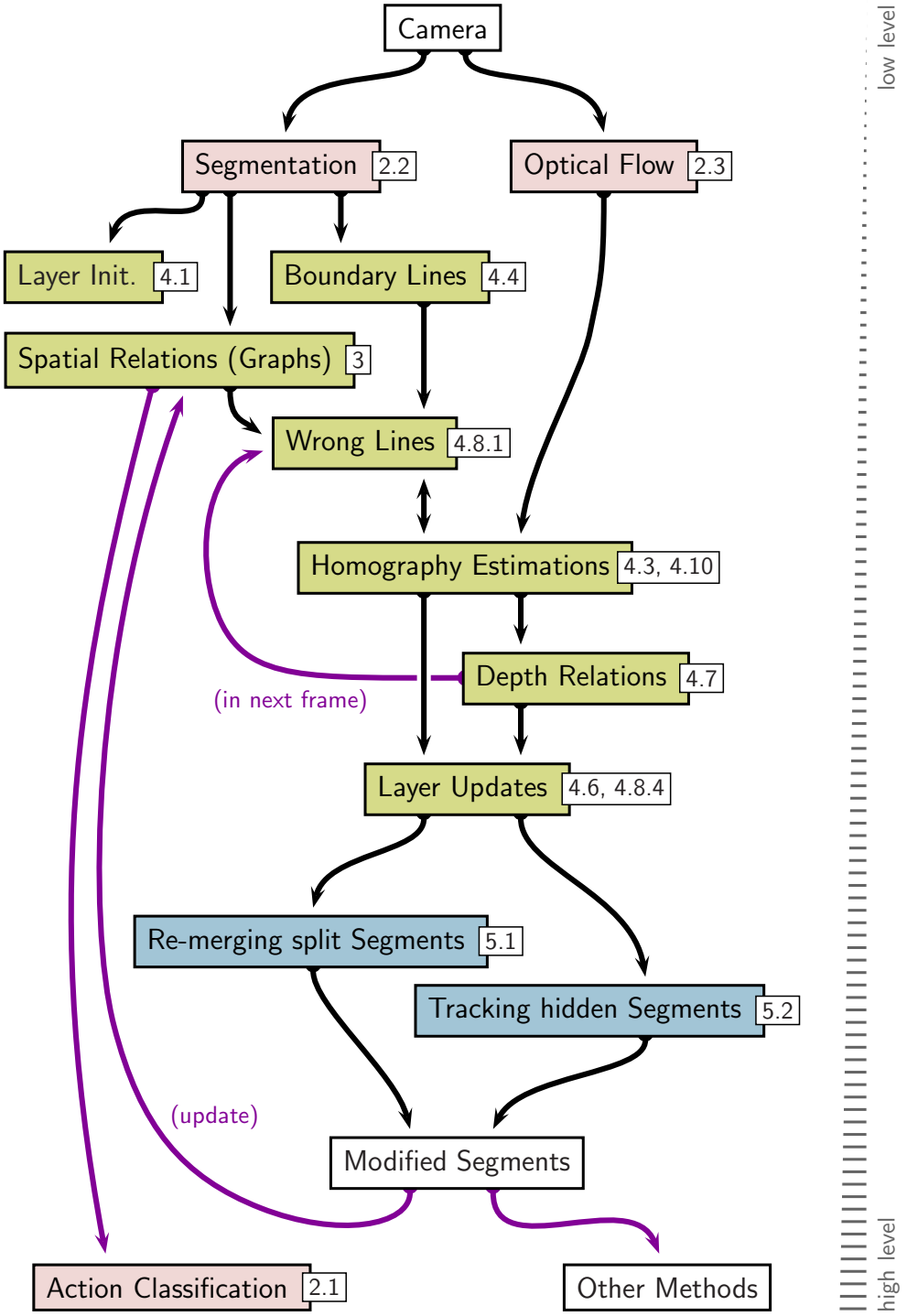


Figure 1.1: Structure of the proposed framework. The numbers correspond to sections in this work. Colors: external algorithms, layer framework, high level reasoning, → data flow during first iteration, → data flow after first iteration. The abstraction level increases from top to bottom.

4.8.1).

Subsequent to the homography estimation and the computation of the depth relations, the layers are updated in case new parts of a segment have become visible (section 4.6.1). Furthermore, with the help of the relative depths, noise in the layers is removed (section 4.6.2). Besides that, the perspective of a layer can be changed if it offers better conditions for storing the data (section 4.8.4).

The segment memory given by the layers is used by two high level modules ("Re-merging split Segments" and "Tracking hidden Segments"). The first recognizes segment splits and relabels the subparts back to the original segment's label (section 5.1). The second module tracks hidden segments by finding correlations to other segments in advance (section 6.3). That way, the hidden segment's position is known during occlusion. When it reappears with a new label, it is labeled back to its original label. Both modules ascertain object permanence which is the main goal of this work.

As both high level modules modify the original segmentation, the spatial relations, which have been computed erlier, have to be updated. The proposed algorithm (section 3) is able to do this without re-analyzing the segmented image but by using its internal representation which has been updated during all changes.

The spatial relations, which are represented as a graph, are used as input by the action classification framework which is developed in the same study group and shortly described in section 2.1. Other methods could also use the modified segments map.

This overview shows how the low-level data provided by the camera is step-by-step becoming more abstract. Although the representing of the scene using a graph is done very early after the segmentation, the framework completes it with object permanence by successively obtaining correct boundaries, movement estimation, relative depth and by remembering segment parts during occlusions.

The gained scene representation is then used to perform a very high level task, namely *action recognition*.

Chapter 2

Basics and Theory

2.1 Action Recognition in Videos Using Semantic Scene Graphs

Vision-based *action recognition* deals with classifying image sequences with respect to their semantic content. In other words, the goal is to understand what happens in an image sequence and, as a part of that, group together different videos that show the same action. Note that the term *video* also includes real-time data (live streams) of images since the developed algorithms are also intended to be used on robots in order to give them the ability to recognize and execute taught actions. While the field of recognizing e.g. human motion patterns is well established, only few solutions for recognizing manipulation actions have been proposed so far [44, 40, 24]. The following description is taken to a large degree from Aksoy et al. [3].

It is long known that raw observation and naive copying are insufficient to execute an action by a robot. Execution requires capturing the action's essence. Humans are – without problems – able to capture "the essence" and recognize the consequences of their own actions as well as those performed by others. While the mirror-neuron system is suspected to be involved in this feat [37], it is until now completely unknown how newborns learn to recognize and imitate and thereby develop advanced motor skills aiding their cognitive development.

Part of the problem lies in the fact that usually there are many ways to perform a certain manipulation. Movement trajectories may be different and even the order of how to perform it may change to some degree. On the other hand, certain moments during a manipulation will be similar or even identical. For example, during a manual assembly process certain object combinations must occur without which mounting would render nonsense.

This points to the fact that at certain pivotal time-points one needs to com-

prehend specifically the momentarily existing relation between manipulator (hand) and manipulated object as well as the resulting relations (and their changes) between objects and object parts.

Some methods for action recognition use *object recognition* as a preprocessing step. By that, the action recognition is separated from the vision problem since it works on the found objects and not on the original images anymore. The disadvantage of that is that prior object knowledge is needed. Furthermore, this object knowledge has a large impact on the actions that can be classified. Let one action be "putting a pen into a pencil case". In this example, the pen and the pencil case play the important roles. Another action might be "putting the cap on the pen". Here, the pen consists of two parts that have to be distinguished. Obviously, it depends on the action whether the whole object or subparts need to be considered.

Furthermore, the same object can play different roles in different actions and the same action might be compatible with different objects. An example for that is "putting an object into a container". It is obvious, that this action can be performed with many different objects and that they define a common object class in this action context. Hence, more important than recognizing the actual object is recognizing its role.

These examples show that entirely separating object from action recognition limits the capabilities of the system. Furthermore, it has recently been shown in psychophysical experiments that also action context can facilitate human object recognition [18], which shows that the human brain also works in such a way.

However, the raw data obtained by cameras contains a large amount of data which is in general inapplicable as input data for logic or learning algorithms. For that reason, a model-free scene description is needed which can be derived from the input images and then be used as input for action recognition.

In Aksoy et al. [3] a novel, model-free encoding scheme for manipulations is introduced. The so-called "Semantic Event Chain" (SEC) fulfills important requirements: it is obtainable with sensors, learnable, view-invariant, compressed and human-comprehensible.

The algorithm proceeds as follows: First, all frames from the movie are segmented by a method like the one described in section 2.2 for a consistent markerless tracking of the individual segments. The scene is then represented by graphs the nodes of which represent segments and the edges their neighborhood relations. Regarding the whole image sequence, those graphs can change by continuous distortions like lengthening or shortening of edges, or, more importantly, through discontinuous changes because nodes or edges can appear or disappear. Such a discontinuous change represents a natural breaking point: All graphs before are topologically identical and so are those after the breaking point. Given the graphs derived from the spatial segment relations, an exact graph-matching method can

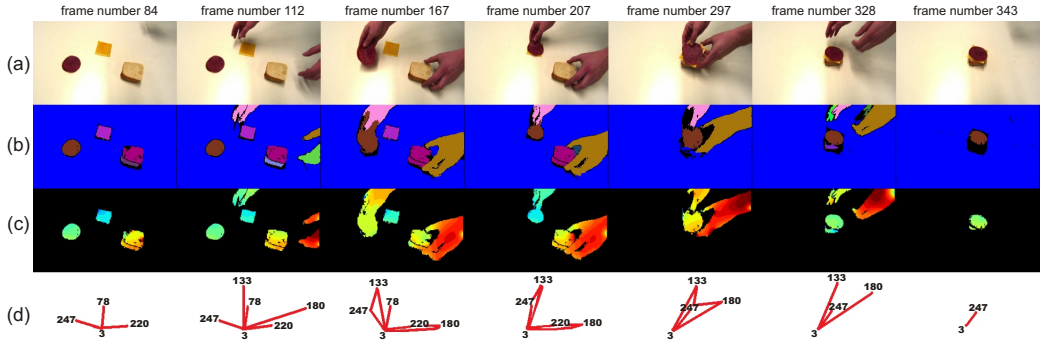


Figure 2.1: Results for the sample action "Making a sandwich". (a) Original frames from the left image sequence. (b) Extracted segments for frames of the left sequence. (c) The dense disparity maps obtained for extracted stereo segments. The disparity values are color-coded from blue (small) to red (large disparity values). (d) Final 3D semantic scene main graphs, representing action primitives. [3]

be applied at each breaking point to extract the corresponding topological main graph. The sequence of these main graphs thus represents all structural changes in the scene. Fig. 2.1 shows an example sequence with corresponding main graphs.

The action recognition is then performed on these main graphs which are transformed to a table-based format, the mentioned SEC. Note that the shown image sequence consists of nearly 350 frames but is represented by only 7 main graphs. For a detailed description of the classification algorithm see [3].

Obtaining the main graphs from an image sequences which contain different actions as well as different versions of the same action has been a main concern of the author of this work. The developed method used in [3] for finding the segment's neighborhood relations in each frame is presented in section 3. However, the retrieval of few main graphs from the whole graph sequence requires further filtering. Besides the removal of flickering segment relations, stereo-vision has been used to remove connections between segments which are not connected in the 3D domain – the resulting graphs were called "3D semantic scene graphs". This increases the degree of view-invariance of the scene description.

Here we are also confronted with the object permanence problem which is not covered in [3]: The nodes of the graphs do not necessarily correspond to real objects and are futhermore not stable against occlusions. Thus, my work addresses (a) the improvement of the semantic scene description and (b) finding potential solutions for the object permanence problem.

2.2 Efficient Image Segmentation

This section describes the framework for efficient segmentation of videos which is used in this thesis. Parts of this description are taken from Abramov et al. [2], where the whole system is described in detail.

Image segmentation, i.e. the partitioning of an image into disjoint parts based on some image characteristics, such as color information, intensity or texture is one of the most fundamental tasks in computer vision and image processing and of large importance for many kinds of applications, e.g. object tracking, classification and recognition. As a consequence, many different approaches for image segmentation have been proposed in the last twenty years, e.g. methods based on homogeneity criteria inside objects of interest, clustering, region-based growing, graph cuts and mean shift segmentation. One can distinguish between parametric (model-driven) and nonparametric (data-driven) techniques. If little is known about the data being segmented, nonparametric methods have to be applied. The method of superparamagnetic clustering is a nonparametric method which solves the segmentation problem by finding the equilibrium states of the energy function of a ferromagnetic Potts model (without data term) in the superparamagnetic phase [36, 11, 6].

The Potts model [36], which is a generalization of the Ising model [21], describes a system of interacting granular ferromagnets or spins that can be in q different states, characterizing the pointing direction of the respective spin vectors. Depending on the temperature, i.e. disorder introduced to the system, the spin system can be in the paramagnetic, the superparamagnetic, or the ferromagnetic phase. In the ferromagnetic phase all spins are aligned while in the paramagnetic phase the system is in a state of complete disorder. In the superparamagnetic phase regions of aligned spins coexist. Blatt et al. (1996) applied the Potts model to the image segmentation problems in a way that in the superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data [6]. Finding the image partition corresponds to the computation of the equilibrium states of the Potts model.

Those equilibrium states of the Potts model have been approximated in the past using the Metropolis-Hastings algorithm with annealing [14] and methods based on cluster updating, which are known to accelerate the equilibration of the system by shortening the correlation times between distant spins.

The segmentation framework used in this work performs a real-time model-free image segmentation on GPUs[1] based on a novel parallel method, combined with real-time phase-based optical flow [35] and stereo [38], executed on GPU as well, for segment tracking. A detailed description is given in [2].

[1] Graphical Processing Unit

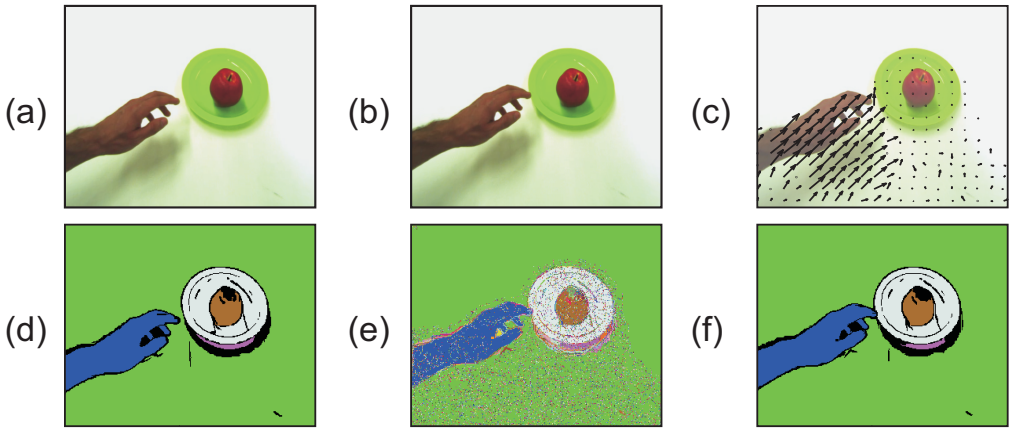


Figure 2.2: Segmentation of two adjacent frames in a sequence. (a) Original frame t. (b) Original frame $t+1$. (c) Estimated optical flow field from phase-based method. (d) Extracted segments S_t for frame t. (e) Initialization of frame $t+1$ after the spin transfer. (f) Extracted segments S_{t+1} for frame $t+1$. [2]

First, a parallel Metropolis spin relaxation procedure is developed and used to partition the first image of the sequence into disjoint regions. Then, information about pixel correspondences between subsequent frames is obtained via a phase-based optical flow algorithm applied to the frame sequence, as described in section 2.3. For pixels in weakly-textured regions, where no optical flow could be established, the assumption is made that they did not change position between the current and the next frame.

The initial label configuration for the next frame is then found by translating all labels according to the derived flow maps. Suppose frame t is segmented and S_t is its final label configuration (see Fig. 2.2(d)). The labels can be transferred from frame t to frame $t+1$ according to

$$S_{t+1}(x_{t+1}, y_{t+1}) = S_t(x'_t, y'_t)$$

with $x'_t = x_{t+1} - u_{\mathrm{x}}(x_t, y_t)$ and $y'_t = y_{t+1} - u_{\mathrm{y}}(x_t, y_t)$ where u_{x} and u_{y} are the values obtained by optical flow. Labels which did not obtain an initialization by that are then given a randomly chosen label between 1 and q.

Once frame $t+1$ is initialized (see Fig. 2.2(e)), another Metropolis relaxation process is started in order to fix erroneous bonds. Compared to the number of iterations in the first frame, this process can be stopped after a short time since most of the labels are already correct.

2.3 Phase-based Optical Flow

Since fast processing is a very important issue in the segmentation algorithm described in 2.2, it uses the GPU-based real-time optical flow algorithm proposed by Pauwels et al. [35]. This algorithm belongs to the class of phase-based techniques, which are highly robust to changes in contrast, orientation and speed. This particular algorithm integrates the temporal phase gradient (extracted from five subsequent frames) across orientation and gradually refines its estimates by traversing a Gabor pyramid from coarser to finer levels. The algorithm provides a vector, at each pixel indicating its motion

$$\mathbf{u}(x,y) = (u_{\mathrm{x}}(x,y), u_{\mathrm{y}}(x,y)) \ .$$

This allows for linking pixels of two subsequent frames t and $t+1$ in the image segmentation algorithm as described in 2.2 and shown in Fig. 2.2(e). Furthermore, the proposed framework uses optical flow to estimate homography matrices between subsequent frames.

2.4 Homogeneous Coordinates

Many important vector transformations like scaling or rotation are linear with the property

$$f(c\mathbf{x}_1 + \mathbf{x}_2) = cf(\mathbf{x}_1) + f(\mathbf{x}_2) \ .$$

When dealing with image coordinates, the mapping is given by

$$\mathbb{N}^2 \to \mathbb{N}^2 : \mathbf{x} \mapsto f(\mathbf{x}) \ .$$

It is known from linear algebra that those mappings can be expressed by the product of a 2×2 matrix and the vector:

$$f : \mathbf{x} \mapsto \mathbf{M}\mathbf{x} \ .$$

The columns of this matrix are given by the transformed basis vectors. One advantage of using matrices is that subsequent transformations like $(f \circ g)(x) = f(g(x))$ are given by the product of the corresponding matrices:

$$\mathbf{M}_{f \circ g} = \mathbf{M}_f \cdot \mathbf{M}_g \ .$$

However, the translation, another important transformation, is not a linear function since it requires a vector addition. Transformations that include translations

(e.g. rotation around an arbitrary point which consists of translation to the origin, rotation, translation back to the original position) can only be described by *affine transformations* with the form

$$f_{\text{aff}} : \mathbf{x} \mapsto f(\mathbf{x}) + \mathbf{t} = \mathbf{M}_{\text{lin}}\mathbf{x} + \mathbf{t} \ , \tag{2.1}$$

where $\mathbf{t}$ is a 2×1 translation vector. In euclidean coordinates, this cannot be expressed by a simple matrix multiplication. Since it is inconvenient to have two different types of matrix operations, *homogeneous coordinates* are often used in computer graphics. They allow us to express affine mappings by matrix multiplications.

This is achieved by extending the euclidean vectors by an additional coordinate with value 1:

$$\mathbf{x}_{\text{eucl}} = \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \mathbf{x}_{\text{hom}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} .$$

Given a homogeneous vector, the corresponding euclidean vector can be obtained by

$$\mathbf{x}_{\text{hom}} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \mapsto \mathbf{x}_{\text{eucl}} = \begin{pmatrix} u/w \\ v/w \end{pmatrix} .$$

Note that homogeneous coordinates allow us to express points at infinity by setting $w = 0$. This plays an important role in *Projective Geometry* where all, even parallel lines have an intersection point.

One important fact is that multiplying a homogeneous vector with a constant does not change the corresponding euclidean vector. In other words, by using homogeneous coordinates, points are represented by lines.

A translation can now be expressed by the matrix multiplication

$$\begin{bmatrix} 1 & 0 & t_{\text{x}} \\ 0 & 1 & t_{\text{y}} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} u + t_{\text{x}}x_3 \\ v + t_{\text{y}}x_3 \\ w \end{pmatrix} \mapsto \begin{pmatrix} u/w + t_{\text{x}} \\ v/w + t_{\text{y}} \end{pmatrix} .$$

More generally, the matrix of an affine transformation is given by

$$\mathbf{M}_{\text{aff}} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_{\text{lin}} & \mathbf{t} \\ {}^{\text{t}}\mathbf{0} & 1 \end{bmatrix}$$

where $\mathbf{M}_{\text{lin}}$ is the 2×2 matrix corresponding to the linear part of the affine transformation in (2.1), and $\mathbf{t}$ the translation vector.

Affine transformations allow rotation, scaling, shear, mirror and parallel projection. Lines are always mapped to lines (not curves) and parallelism is preserved. In this work, a more general type of transformation is used, the *projective transformation* or *homography*. Lines are still mapped to lines but parallelism is not preserved. By that, perspective transformations can be modeled that describe the morphing of an object's projection on the image plane. The corresponding *homography matrix* has the form:

$$\mathbf{M}_{\text{proj}} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & 1 \end{bmatrix}.$$

The transformation matrices can be obtained by point correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$. It is easy to see that $\mathbf{M}_{\text{aff}}$ has 6 degrees of freedom, while $\mathbf{M}_{\text{proj}}$ has 8. A method for estimating $\mathbf{M}_{\text{proj}}$ is described in section 4.3. There it is shown that (at least) 4 point correspondences are needed for the estimation.

Chapter 3

Spatial Relations between Image Segments

The method to classify actions recorded with a camera, that was shortly described in section 2.1, is based on segmentation. The main idea is that the spatial relations of those segments behave in a way that is characteristic for the observed action. Their temporal development might differ in different videos of the same action, but should have a higher intrinsic similarity compared to completely different actions. Consequently, the first task is to extract the spacial relations between segments from each frame.

The following possible relations between two segments are defined: *No Connection* (the segments are not neighbors), *Touching* (the segments are neighbors) and *Overlapping* (one segment is surrounded by the other). While the first two relations are simple to recognize, the third one is more challenging.

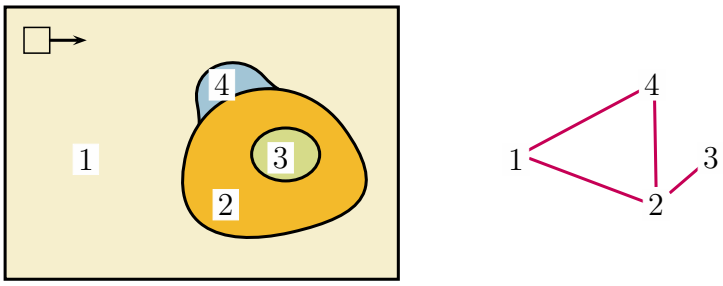
In the following, some simple methods are described. Since they all are too unstable, a new method is proposed.

3.1 Window-based Algorithm

The old algorithm used by our group is only able to find *Touching* relations [4]. This is done by moving a window through the image and checking at each step, if pixels of two or more different segments are located in it (see Fig. 3.1). In this case, the relations of those segments to each other are set to *Touching*. After scanning the whole image, all other pairs of segments are assigned the relation *No Connection*.

Given the relations, a graph can be build that has the segment labels as nodes and in cases of a *Touching* relation an edge between the respective nodes.

The recognition of *Overlapping* is exclusively based on the analysis of the



(a) Scanning (b) Computed graph

Figure 3.1: Window-based algorithm to find neighbored segments

created graph. If one node has only one edge, it is assumed that the corresponding segment is completely surrounded. Regarding the example in Fig. 3.1, the segment pair $(2, 3)$ is recognized as *Overlapping* while the pairs $(1, 2)$ and $(1, 4)$ are only *Touching* and $(1, 3)$ and $(3, 4)$ have *No Connection*.

This example already shows the drawbacks of this algorithm. The relations of $(1, 2)$ and $(1, 4)$ should be *Overlapping* because 2 and 4 are completely surrounded by 1. Since 2 and 4 are connected to other segments, thus their nodes have more than only one edge, the algorithm fails. Even a more complex analysis of the graph cannot lead to a correct recognition of the *Overlapping* relation, which becomes obvious when looking at the graph. The nodes 1 and 4 hold no information that could be used to find out, that 1 surrounds 2, but 4 does not. In other words, swapping the labels 1 and 4 would not cause a topological change of the graph.

3.2 Alternative Methods

Some other methods that try to solve the proposed problem are using the *Minimum Bounding Rectangle* (MBR) [33]. In doing so, the smallest rectangle that completely covers a segment is established. All those rectangles are then tested for *Overlapping*. Since the MBR is only an approximation of the actual shape, this will return wrong results when treating slightly more complex segments. Fig. 3.2 shows an example for that.

By approximating the segments e.g. by polygons the accuracy of the method can be increased. The computational cost also increases, though.

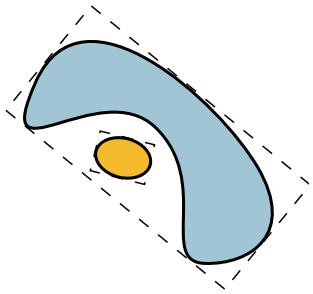


Figure 3.2: Minimum Bounding Rectangle of two segments

3.3 New Algorithm

The now proposed method works in a different way and provides exact results in a very short time. The algorithm consists of two parts.

In the first step, the segmented image is scanned horizontally (from left to right) and vertically (from top to bottom). By doing that, the occurring sequences $i_1, i_2, ...$ of segments are recorded. One sequence S can usually be found in multiple lines (for simplicity, this includes both lines and columns of the image). This amount n_S of lines (not of pixels) is stored in the *sequence list*

$$L: \quad (i_1, i_2, i_3, ...) \mapsto n_S \ .$$

The example in Fig. 3.3 contains $L((1)) = 185$ and $L((1,2,1,3,1)) = 15$ and others. After finding all sequences, the scanning of the image is finished. The established sequences hold all information that is needed for the next step.

In the second part of the algorithm the sequences are analyzed. The following two rules can be found:

- **"Touching"**: Two segments following one right after the other are *Touching*, just like 2 and 3 in $(..., 2, 3, ...)$ or $(..., 3, 2, ...)$, since there is no other segment in between.

- **"Overlapping"**: (i) If one segment occurs twice in a sequence, all segments in between are *Overlapping* with it. For example, 1 surrounds 3 in the sequence $(..., 1, 3, 1, ...)$. In $(..., 1, 5, 3, 1, ...)$, it surrounds even two segments, namely 3 and 5. (ii) Besides that, such segments that are being surrounded cannot surround each other. As an example, 5 cannot surround 3 in a sequence like $(..., 1, 5, 3, 1, ...)$ since it would need to occur twice as described in (i).

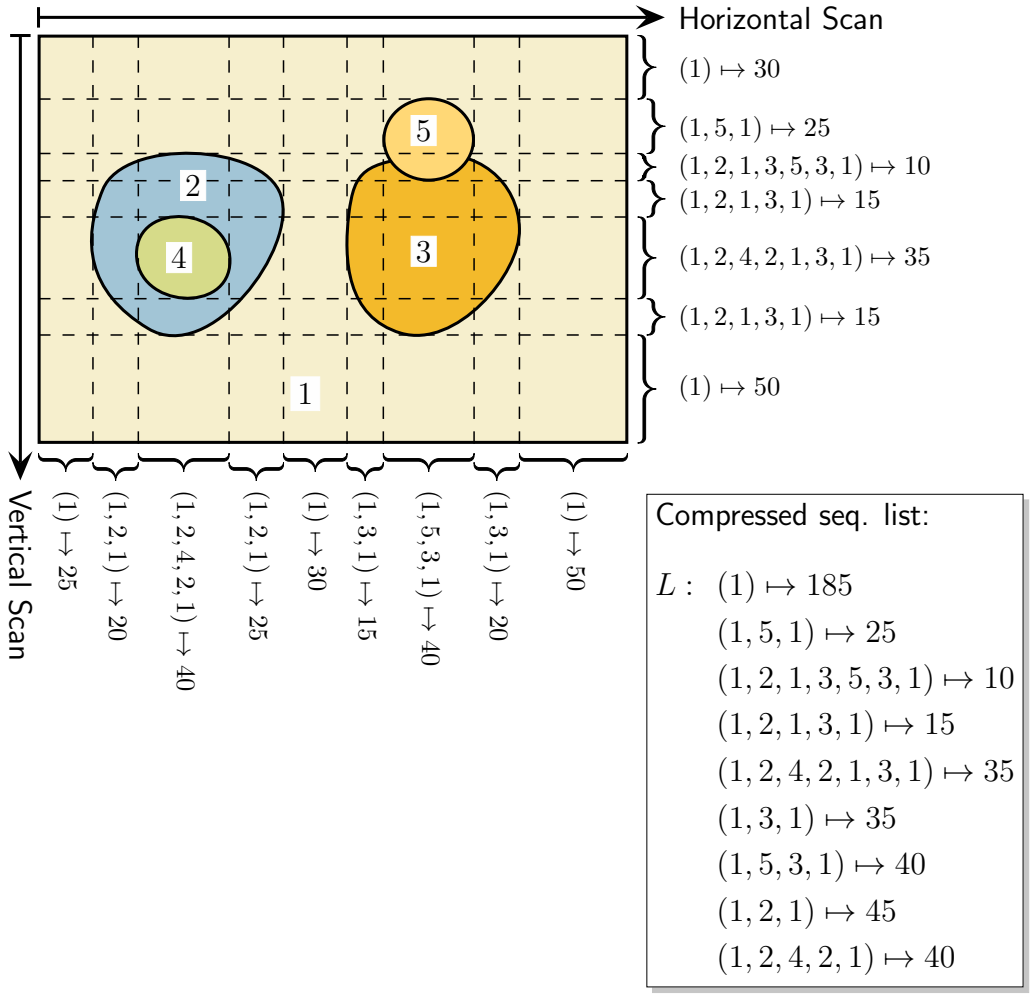


Figure 3.3: Scanning procedure of a segmented image with size $165\text{px} \times 180\text{px}$ in order to compute the segment's spatial relations.

Those rules are applied to all sequences. A pair's relation that was found in one of the sequences is not necessarily correct in the context of the whole image. On top of that, the analysis of two different sequences can provide contradictory results. That is why a counting matrix is introduced for each of the two rules. Its elements (i, j) are incremented when the respective relation between segments i and j was found. This proceeding can be regarded as a voting procedure.

- $\mathbf{C}^{\mathrm{t}}_{i,j} \mapsto n_t$: Number of votes for i and j being neighbors (*Touching*).

- $\mathbf{C}^{\mathrm{o}}_{i,j} \mapsto n_o$: Number of votes that j is surrounded by i (*Overlapping*).

Since the *Touching* relation is not directed, we have $\mathbf{C}^{\mathrm{t}}_{i,j} \equiv \mathbf{C}^{\mathrm{t}}_{j,i}$. Contrary to that, $\mathbf{C}^{\mathrm{o}}_{i,j}$ is not symmetric.

The analysis of each sequence is independent and works as follows: All elements $(i_1, i_2, i_3, ...)$ of sequence S are copied to a stack one after the other. During that process, the described rules are applied to the stack as follows.

After an element i_n has been copied to the stack, the first rule indicates that this element is *Touching* the previous one i_{n-1}. Since the whole sequence S was found $L(S)$ times in the image, this corresponds to as many votes. Accordingly, the counter is increased by

$$\mathbf{C}^{\mathrm{t}}_{i_{n-1},i_n} \mathrel{+}= L(S) \ .$$

Example: The sequence $S := (1, 5, 3, 1)$, that was found 40 times, is considered. In the first step, the element $i_1 = 1$ is copied to the stack. Since all rules require at least two elements on the stack, the algorithm immediately skips to adding the second element $i_2 = 5$. The first rule indicates *Touching* for the pair $(1, 5)$. Hence, $\mathbf{C}^{\mathrm{t}}_{1,5}$ is increased by $L(S) = 40$. The same operations are executed for $(5, 3)$ in the next step.

To fulfill the second rule the previously stored element needs to be checked whether it is already in the stack. In this case, the elements between this first occurrence i_s and i_n are recognized as having the *Overlapping* relation with i_n. Therefore, the corresponding counter will be updated as follows:

$$\mathbf{C}^{\mathrm{o}}_{i_n,j} \mathrel{+}= L(S), \quad \forall j \in \{i_{s+1}, ..., i_{n-1}\} \ .$$

Example: In the same sequence as given in the previous example, the next element $i_4 = 1$ is added to the stack and $\mathbf{C}^{\mathrm{t}}_{1,3}$ is incremented by 40 (first rule). Since i_4 occurred earlier ($i_s = i_1$), for all elements in between, hence $i_2 = 5$ and $i_3 = 3$, the corresponding counters $\mathbf{C}^{\mathrm{o}}_{1,5}$ and $\mathbf{C}^{\mathrm{o}}_{1,3}$ are increased by $L(S) = 40$.

The second rule also indicates that those inner elements j do not overlap with each other, which results in the following update:

$$\mathbf{C}^{\mathrm{o}}_{j_n,j_m} \mathrel{-}= L(S), \quad \forall j_n, j_m \in \{i_{s+1}, ..., i_{n-1}\}, \ n \neq m \ .$$

Example: According to that, $\mathbf{C}^{\mathrm{o}}_{3,5}$ and $\mathbf{C}^{\mathrm{o}}_{5,3}$ are decreased by 40.

Next, the inner elements are removed from the sequence. This is essential in cases of recursive *Overlapping* situations. The following example for that relates to the segments 1, 2 and 4 in Fig. 3.3.

Example: Let $S := (1, 2, 4, 2, 1)$ be the regarded sequence. In the fourth step, $i_4 = 2$ is copied to the stack. After that, the stack contains $[1, 2, 4, 2|$. By

considering the description given above, $\mathbf{C}^{\mathrm{t}}_{2,4} += L(S)$ (first rule) and $\mathbf{C}^{\mathrm{o}}_{2,4} += L(S)$ (second rule) is computed. The elements i_3 and i_4 are then removed from the stack, which contains after that the elements $[1, 2|$. The algorithm is continuing by adding $i_5 = 1$ to the stack. Besides *Touching* ($\mathbf{C}^{\mathrm{t}}_{1,2} += L(S)$), *Overlapping* of $(1, 2)$ is recognized and $\mathbf{C}^{\mathrm{o}}_{1,2} += L(S)$ is computed. At the end it is observed that segment pairs $(2, 4)$ and $(1, 2)$ have the *Overlapping* relation, however, $(1, 4)$ has *No Connection*.

Second example: The results of different sequences can be contradictory as already mentioned. Fig. 3.4 shows an example which includes the sequences $(1, 2, 3, 2, 1)$ and $(1, 2, 3, 1)$. Due to rule 2(i), it is determined in the first sequence that 3 is surrounded by 2. However, the second sequence creates votes for the opposite, since 2 and 3 cannot be *Overlapping* according to rule 2(i). Therefore, the corresponding counter gets both incremented and decremented. Its final value for the example in Fig. 3.4(a) is $\mathbf{C}^{\mathrm{o}}_{2,3} = L((1, 2, 3, 2, 1)) - L((1, 2, 3, 1)) = 20 - 30 = -10$. (Negative values do not have a special meaning, they are just representing a very low number of hints.) For the example in (b) it is $L((1, 2, 3, 2, 1)) - L((1, 2, 3, 1)) = 85 - 10 = 75$. The value of the *Touching* counter $\mathbf{C}^{\mathrm{t}}_{2,3}$ is 70 or 180, respectively.

Once all sequences are iterated, the values in $\mathbf{C}^{\mathrm{t}}_{i,j}$ and $\mathbf{C}^{\mathrm{o}}_{i,j}$ are used to compute the final spatial relations of the segments. In order to deal with noise effects, each counter has to reach a minimum value so that the corresponding relation becomes valid. Since a big segment can be found in more sequences, the counter values concerning that segment are higher in general. Moreover, the number of errors also increases with the segment's size. That's why an absolute threshold cannot be used for this task. Instead, the entries in the counter matrices need to get normalized. This is done as follows.

In the best case, each entry of the matrix should provide how many votes compared to the maximum possible votes were found. In order to find a *Touching* or *Overlapping* of (i, j) in a sequence S, both segments obviously have to be found in the sequence. Note that there can be more than one match for a relation in one sequence. Let $N_i(S)$ be the number of occurrences of i in S. Then, the maximum possible number of votes for *Touching* in this sequence is

$$c^{\mathrm{t,max}}_{i,j}(S) = N_i(S) + N_j(S) - 1 \ .$$

For *Overlapping*, it is

$$c^{\mathrm{o,max}}_{i,j}(S) = \min(N_i(S), N_j(S)) \ .$$

The normalized counter matrices are then given by

$$\hat{\mathbf{C}}^{\mathrm{t}}_{i,j} := \frac{\mathbf{C}^{\mathrm{t}}_{i,j}}{c^{\mathrm{t,max}}_{i,j}} \quad \text{and} \quad \hat{\mathbf{C}}^{\mathrm{o}}_{i,j} := \frac{\mathbf{C}^{\mathrm{o}}_{i,j}}{c^{\mathrm{o,max}}_{i,j}} \ ,$$

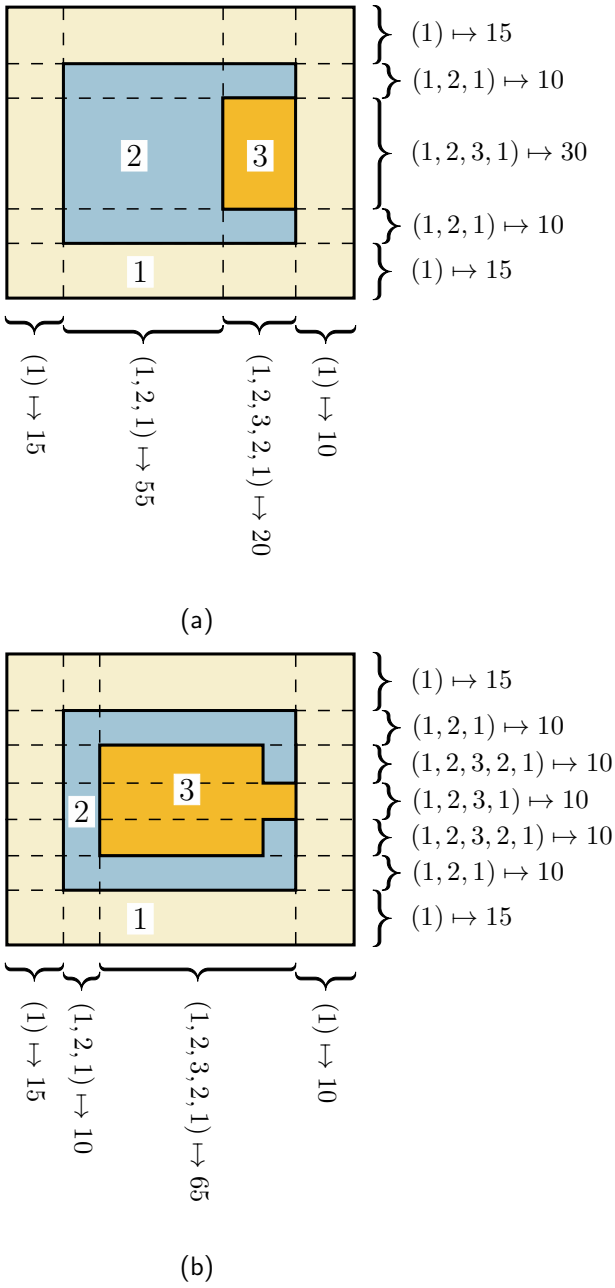


Figure 3.4: Two examples (image size is $100\text{px} \times 90\text{px}$) where the sequences give contradictory results. In this case, the relation of $(2, 3)$ is concerned. By setting a threshold to a proper value, the relation is recognized as *Touching* in (a) and *Overlapping* in (b). Hence, the threshold has to be set according to the desired result.

where $c_{i,j}^{\mathrm{t,max}}$ and $c_{i,j}^{\mathrm{o,max}}$ were summed over all sequences S containing i and j. To save computational time and memory, the simplified normalization

$$\hat{\mathbf{C}}_{i,j}^{\mathrm{t}} := \frac{\mathbf{C}_{i,j}^{\mathrm{t}}}{\min(N_i, N_j)}$$

is used (analog for $\mathbf{C}_{i,j}^{\mathrm{o}}$), where N_i specifies, how often segment i was found in all sequences. Note that this is not a derivable approximation but an experimental value that yields good results.

The elements of the normalized counter matrices have to reach a threshold, so that the corresponding relation is set to *True*. When analyzing those matrices, the first thing to check is whether a *Touching* relation can be found:

$$\hat{\mathbf{C}}_{i,j}^{\mathrm{t}} \stackrel{?}{>} t_{\mathrm{t}} \ .$$

For the threshold $t_{\mathrm{t}} \approx 0.1$ a small value is chosen in order to recognize even small tangencies. In cases of a *Touching*, *Overlapping* is checked:

$$\hat{\mathbf{C}}_{i,j}^{\mathrm{o}} \stackrel{?}{>} t_{\mathrm{o}} \ ,$$

where $t_{\mathrm{o}} \approx 0.95$ corresponds to a high value. Only almost complete surroundings of segments are this way recognized as *Overlapping*.

The proposed algorithm has the following advantages:

- **Effectivity:** The algorithm is able to recognize even recursive *Overlapping* situations, the previous algorithm was not capable of. Compared to the methods based on the Minimum Bounding Rectangle, it does not rely on approximations of the segment shapes.
- **Computational efficiency:** The line- and column-wise scanning (first part of the algorithm) can be done in a parallel way by dividing the image into parts. Furthermore, the analysis of the sequences can be parallelized. For each thread processing one sequence, independent counter matrices can be created and merged afterwards, hence there is no need for a shared memory which is always a plus when dealing with multiple processors.
- **Removal of segments:** In this work, methods are developed to recognize and remove erroneous segments, or to relabel them. After that, a re-computation of the spatial relations becomes necessary. Instead of doing this in the image domain and reanalyze the frame again completely, those operations can be done in the sequences. E.g. to delete a segment, it has to be removed from each sequence. Then, only the analysis of the sequence has to be redone, while a second scanning of the image (first part) is not necessary.

Chapter 4

Layer-based Segment Memory

In this chapter, a new approach for a segment memory is proposed. It will be used later to handle situations of occluded and split segments.

As described in section 2.2, the color-based image segmentation is used to find homogeneous regions in the image. It is model-free which means that no prior knowledge about the shape or any other properties of the objects in the scene is needed. This also means on the other hand that no recognition of physical objects is provided at that stage. One object can be represented by more than one segment what is crucial for the action classification method described in section 2.1.

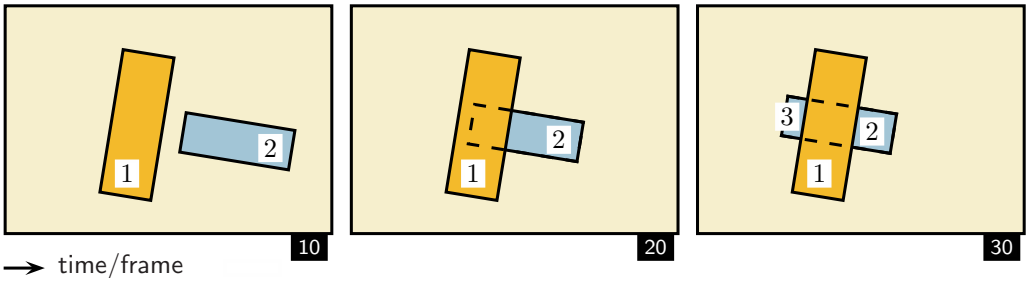


Figure 4.1: In this example segment 2 becomes occluded by 1. An algorithm is needed to ensure object permanence when the object comes out at the other side. The standard image segmentation will treat the new segment as a new object instead of a part of the already known object.

Image segments are computed frame-wise. If one segment becomes occluded in a later frame, its original shape gets lost. The idea of the proposed algorithms to overcome this drawback is very simple: The segments are stored in a memory and maintained when parts are invisible in the camera image.

4.1 Initialization of Layers

When processing the first frame of an image sequence, for each segment i a copy L_i is created (see Fig. 4.2). These copies are called *layers* in the following. A layer is implemented as a two-dimensional array storing boolean values. *True* indicates pixels that belong to the corresponding segment while *False* stands for empty. In order to save memory, the layers are kept as small as possible: Since there is no need to store empty regions at the border, the layers are cropped to a minimum size. As a consequence, the pixel coordinates in the layer arrays are different from the image coordinates.

In order to be able to change between the coordinate systems, a mapping

$$\mathbf{M}^i : L_i \to \mathrm{Image}$$

is introduced that, at this stage, only performs a translation $(t_\mathrm{x}, t_\mathrm{y})$ corresponding to the layer's offset. By using homogeneous coordinates (see 2.4) this mapping can be expressed as a matrix

$$\mathbf{M}^i = \begin{bmatrix} 1 & 0 & t_\mathrm{x} \\ 0 & 1 & t_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix}$$

which will turn out to be very convenient. Since this matrix only contains a simple shift of the layer with respect to the image origin, this will be denoted as "placement matrix".

Note concerning notation: Although layers are implemented as arrays they will sometimes be treated as sets in this work, since some algorithms can be described easier using set operations. If the layer L contains binary data, the corresponding array L^arr is of type boolean. Then, in terms of sets, L^set consists of tuples that refer to coordinates of the array with value *True*:

$$L^\mathrm{set} = \{\, {}^\mathrm{t}(x, y) \,|\, \forall L^\mathrm{arr}_{x,y} = \mathrm{True}\}$$

Since both notations L^arr and L^set are equivalent, both will be denoted as L.

4.2 Tracking of Segments

For simplicity, we assume that in the first frame the segments of interest are completely visible (this assumption will be dropped in section 4.6). Consequently, their whole shapes are stored in the layers immediately. If one segment gets occluded in the further development of the scene, the layer still holds fullsegment information. However, the mappings $\mathbf{M}^i$ might be wrong now due to the segment's movement. That's why the algorithm has to track all movements in order to update the mappings.

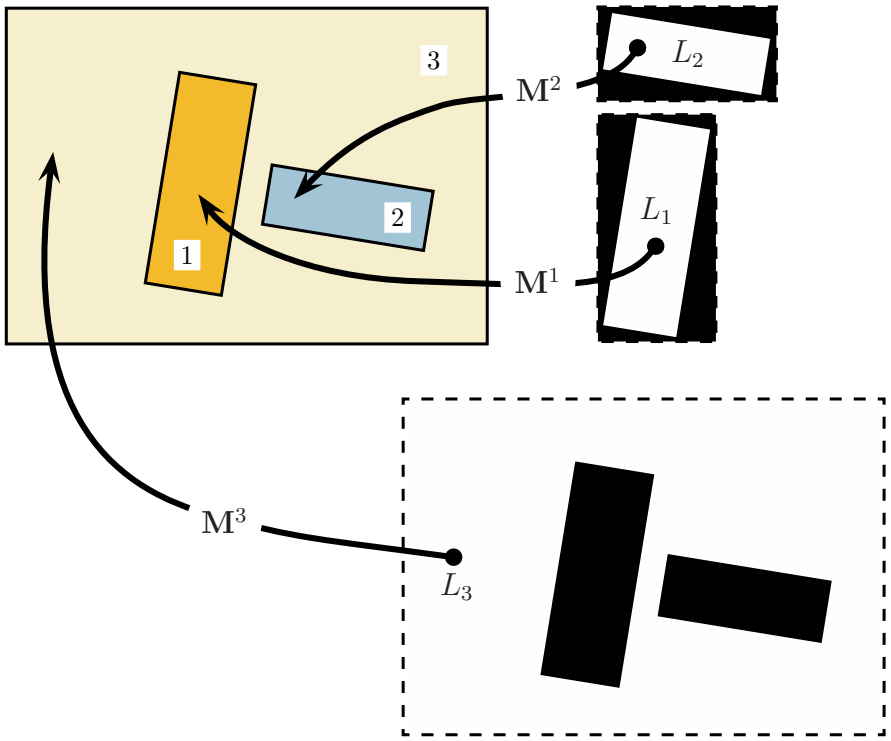


Figure 4.2: For each segment i, a layer L_i is created in the first frame. Besides that, a mapping $\mathbf{M}^i$ is defined that maps from layer coordinates to image coordinates. As soon as a segment moves, the corresponding mapping has to be updated.

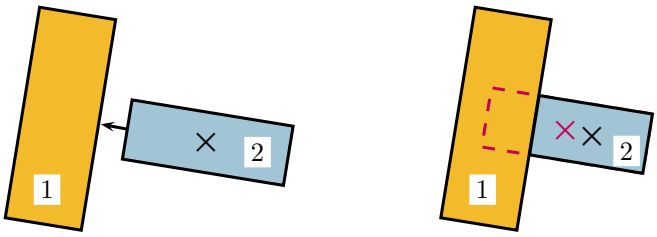
Translational Movements

The segmentation does not provide a unique position but only a set of pixels which represent a segment. The center of weight (COW)

$$\mathbf{x}_S = \frac{1}{N_S} \sum_{p \in S} \mathbf{p} \ , \ \mathbf{p} \in \text{Image} \ ,$$

is a very simple way to obtain an average position value for a segment, but it has a big drawback: It is unstable against occlusions as shown in Fig. 4.3. Since solving occlusions is essential for the proposed system, a different tracking method has to be chosen.

An option is to use optical flow vectors to estimate the movements of the segments. The optical flow algorithm (see section 2.3) provides pixel correspondences between subsequent frames. In general, those correspondences are not given for all pixels of the image, e.g. in regions with no texture. By considering



(a) (b)

Figure 4.3: Using the center of weight (×) for tracking leads to errors in cases of occlusion because it only takes the visible pixels into accout. In order to obtain the correct position (×), e.g. optical flow can be used.

only the pixels of one segment, the average optical flow vector can be used as movement of the whole segment.

Problems with that method arise when shadows are present in the scene. A shadow moving over an object causes optical flow although the object itself is static. This effect can be prevented, at least to a certain extend, by considering the velocity histogram within the segment. Without any shadow, the histogram has a peak around the actual movement of the segment. If a shadow is present, there should be two peaks corresponding to the pixels from shadow and outside. This, of course, requires that there are always enough pixels without shadow.

The histogram is then fitted by the sum of two Gaussian functions. One of the peaks is assumed as movement of the shadow while the other is assumed as movement of the segment. In order to choose the right one, both peaks are compared to the movement of the center of weight.

After the movements t_x and t_y of segment i between two subsequent frames $n-1$ and n have been obtained, the mapping for the current frame n is then given by

$$\mathbf{M}_n^i = \begin{bmatrix} 1 & 0 & t_\mathrm{x} \\ 0 & 1 & t_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{M}_{n-1}^i.$$

Note that $\mathbf{M}_0^i = \mathbf{M}^i$ for the first frame ($n = 0$).

Perspective Deformations

Although a stable tracking can be implemented as described, it can only capture translations but not rotations or perspective deformations. That's why a more sophisticated method is needed.

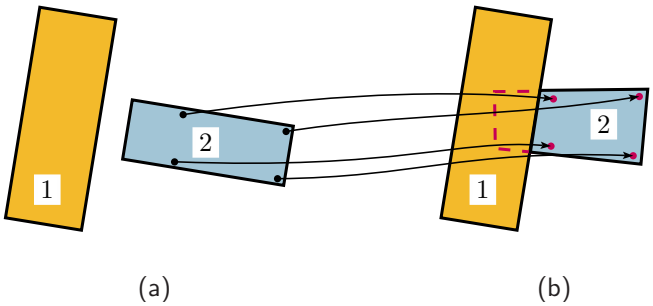


Figure 4.4: Using homography estimation for tracking allows for treating perspective transformations. At least 4 point correspondences are needed, e.g. from optical flow. The estimated transformation also applies to occluded pixels.

Using homogeneous coordinates, a 3×3-matrix can be used to describe all kinds of transformations that can happen to a segment caused by the three-dimensional movement of the corresponding object (see Fig. 4.4). This type of transformation is called *projective transformation* or *homography*. Note, that it is assumed at this point, that all segments correspond to planar objects or, in other words, do not have a three-dimensional shape.

Given the optical flow inside of a segment, the *homography matrix* for one segment between two frames can be computed. At least 4 point correspondences are needed, but the computation becomes more stable with an increasing number of correspondences. This computation is described in detail in section 4.3. Wrong optical flow values caused by errors or shadows are recognized as outliers as described in section 4.3.5.

The Updating Process

For simplicity, the segment number i will be omitted from now on in all formulas and all computations are explained for one segment. It is important to note that, in the real framework, they are also executed for every other segment in the frame.

Fig. 4.5 shows the update process of the mapping. The matrix $\mathbf{P}_n$ corresponds to the transformation between frames n and $n-1$ obtained by a homography estimation.

To map the layer onto the current frame n, all transformations between the previous frames have to be taken into account:

$$\mathbf{M}_n = \mathbf{P}_n \cdot \mathbf{P}_{n-1} \cdot ... \cdot \mathbf{P}_1 \cdot \mathbf{M}_0$$

Instead of storing all $\mathbf{P}_n$ and do the above computation, the matrix can be

computed using the previous value

$$\mathbf{M}_n = \mathbf{P}_n \cdot \mathbf{M}_{n-1} \ .$$

4.3 Estimation of the Homography Matrix

As already mentioned, a homography matrix can be obtained from the point correspondences between two frames. It is also possible to include other geometric elements, like lines. In [10] was shown that combining points and lines in the estimation process improves the accuracy of the obtained matrix. However, in this framework, lines are included for another reason: While the point correspondences are acquired from optical flow, the lines will be derived from the segmented image which is described in section 4.4. Hence, two different and independent methods are combined for a better performance.

This section covers the homography estimation process using point and line correspondences between subsequent frames. The methods are adopted from [17, 49, 10, 48].

4.3.1 Point Correspondences

Let $\mathbf{x} = {}^{\mathrm{t}}(x_1, x_2, x_3)$ be a point that corresponds to the euclidean coordinates $\frac{x_1}{x_3}, \frac{x_2}{x_3}$. Given the coordinates $\mathbf{x}'$ of these points in the next frame, the goal is to find a matrix $\mathbf{P}$ so that

$$c\mathbf{x}' = \underbrace{\mathbf{P}\,\mathbf{x}}_{\mathbf{v}} \ . \tag{4.1}$$

Multiplying $\mathbf{P}$ with a nonzero constant c does not change the transformation since parallel vectors in homogeneous coordinates correspond to the same point. Thus, the matrix has 8 degrees of freedom; $\mathbf{x}'$ and $\mathbf{P}\mathbf{x} =: \mathbf{v}$ are parallel but not necessarily equal.

Either by using the intercept theorem (see Fig. 4.6) or by expressing the parallelism of $\mathbf{x}'$ and $\mathbf{v}$ by

$$\mathbf{x}' \times \mathbf{v} = 0 \ ,$$

the following two equations can be derived:

$$\begin{aligned} x'_2\, v_1 - x'_1\, v_2 &= 0 \\ x'_3\, v_1 - x'_1\, v_3 &= 0 \end{aligned} \tag{4.2}$$

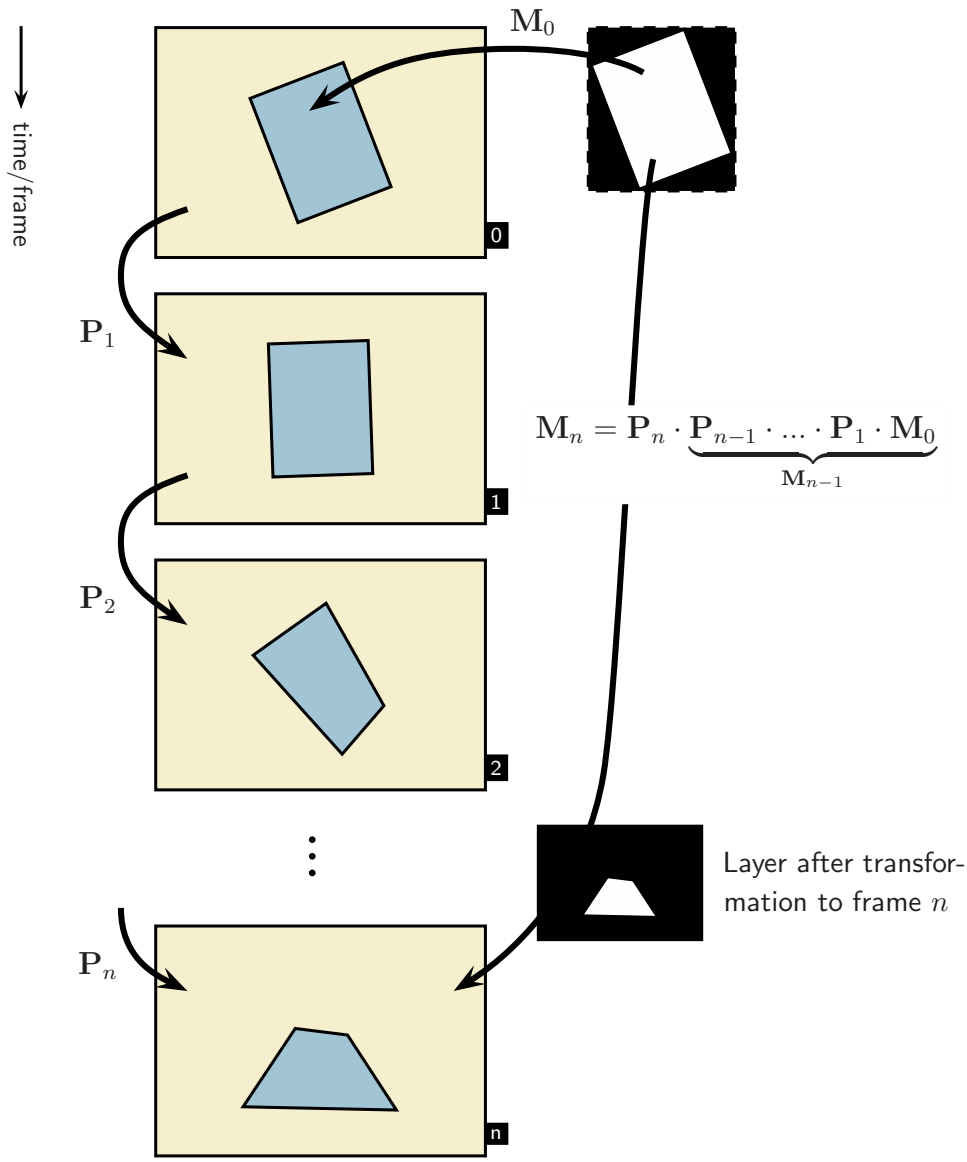


Figure 4.5: The transformation matrix $\mathbf{P}_n$ of the segment between frame n and $n-1$ is computed using optical flow. $\mathbf{M}_0$ is the matrix that maps the layer to the first frame. The mapping $\mathbf{M}_n$ can then be computed using $\mathbf{P}_n$ and $\mathbf{M}_{n-1}$.

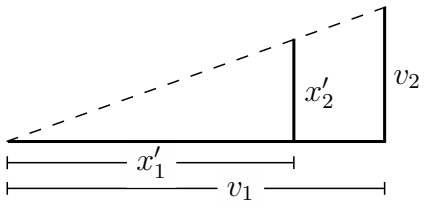


Figure 4.6: The intercept theorem for parallel vectors $\mathbf{v}$ and $\mathbf{x}'$.

The third equation obtained by the cross product, $x'_2\,v_3 - x'_3\,v_2 = 0$, is linearly depending on the two equations as it can be computed by subtracting the first one multiplied with x'_3 and the second one multiplied with x'_2. As it does not contain further information, it is usually omitted.

For $\mathbf{v}$ we can write

$$\mathbf{v} = \mathbf{P}\mathbf{x} = \begin{bmatrix} p_{11} & \cdots & p_{13} \\ \vdots & \ddots & \vdots \\ p_{31} & \cdots & p_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} v_1 = p_{11}\,x_1 + p_{12}\,x_2 + p_{13}\,x_3 \\ v_2 = p_{21}\,x_1 + p_{22}\,x_2 + p_{23}\,x_3 \\ v_3 = p_{31}\,x_1 + p_{32}\,x_2 + p_{33}\,x_3 \end{array} .$$

Then, it is easy to see that equation (4.2) is equivalent to

$$\mathbf{B}\,\mathbf{p} = 0$$

with $\mathbf{p} = {}^{\mathrm{t}}(p_{11}, p_{12}, ..., p_{33})$ and

$$\mathbf{B} = \begin{bmatrix} x'_2x_1 & x'_2x_2 & x'_2x_3 & -x'_1x_1 & -x'_1x_2 & -x'_1x_3 & 0 & 0 & 0 \\ x'_3x_1 & x'_3x_2 & x'_3x_3 & 0 & 0 & 0 & -x'_1x_1 & -x'_1x_2 & -x'_1x_3 \end{bmatrix}. \tag{4.3}$$

Since two equations are gained from one point correspondence $\mathbf{x} \leftrightarrow \mathbf{x}'$, at least 4 points are needed to compute the homography matrix with its 8 degrees of freedom.

4.3.2 Line Correspondences

A line in homogeneous coordinates is given by

$$l_1\,x_1 + l_2\,x_2 + l_3\,x_3 = 0 \;\Leftrightarrow\; \mathbf{l}\cdot\mathbf{x} = 0\ . \tag{4.4}$$

Note that the line equation in euclidean coordinates is $\mathbf{n}\cdot\mathbf{r} - d = 0$ where d is the distance of the line to the origin and $\mathbf{n}$ the (normalized) normal vector. It is obvious that the expression in homogeneous coordinates is a lot shorter and easier to compute.

The scalar product in (4.4) can be written as matrix multiplication $\mathbf{l} \cdot \mathbf{x} = {}^{\mathrm{t}}\mathbf{l}\,\mathbf{x}$. Let $\mathbf{x}_i$ be the points on line $\mathbf{l}$. Then, the following can be derived:

$$\begin{aligned} {}^{\mathrm{t}}\mathbf{l}\,\mathbf{x}_i = 0 \;&\Leftrightarrow\; {}^{\mathrm{t}}\mathbf{l}\,\mathbf{P}^{-1}\,\mathbf{P}\,\mathbf{x}_j = 0 \\ &\Leftrightarrow\; \underbrace{{}^{\mathrm{t}}\mathbf{l}\,\mathbf{P}^{-1}}_{{}^{\mathrm{t}}\mathbf{l}'}\,\mathbf{x}'_i = 0 \;\Rightarrow\; {}^{\mathrm{t}}\mathbf{l}'\,\mathbf{x}'_i = 0. \end{aligned}$$

This result shows that all points $\mathbf{x}_i$ of line $\mathbf{l}$ are on the line $\mathbf{l}'$ in the next frame. Due to the use of homogeneous coordinates, $\mathbf{l}'$ is only defined up to a constant factor c:

$$\mathbf{l}' = c\;{}^{\mathrm{t}}\mathbf{P}^{-1}\,\mathbf{l} \;\Leftrightarrow\; c\,\mathbf{l} = \underbrace{{}^{\mathrm{t}}\mathbf{P}\,\mathbf{l}'}_{\mathbf{k}}$$

In the last step, the vector $\mathbf{l}$ that belongs to the current frame is brought to the left side contrary to (4.1), where $\mathbf{x}'$ is at this place. The reason for this choice is that, otherwise, the inverse of ${}^{\mathrm{t}}\mathbf{P}$ would have to be computed in the later steps. Apart from this difference, the next steps are analog to those in section 4.3.1.

For $\mathbf{k}$ we can write

$$\mathbf{k} = {}^{\mathrm{t}}\mathbf{P}\,\mathbf{l}' = \begin{bmatrix} p_{11} & \cdots & p_{31} \\ \vdots & \ddots & \vdots \\ p_{13} & \cdots & p_{33} \end{bmatrix} \begin{pmatrix} l'_1 \\ \vdots \\ l'_3 \end{pmatrix} \Rightarrow \begin{array}{l} k_1 = p_{11}\,l'_1 + p_{21}\,l'_2 + p_{31}\,l'_3 \\ k_2 = p_{12}\,l'_1 + p_{22}\,l'_2 + p_{32}\,l'_3 \\ k_3 = p_{13}\,l'_1 + p_{23}\,l'_2 + p_{33}\,l'_3 \end{array}.$$

Since $\mathbf{l}$ and $\mathbf{k}$ are parallel

$$\begin{aligned} l_2\,k_1 - l_1\,k_2 &= 0 \\ l_3\,k_1 - l_1\,k_3 &= 0 \end{aligned} \tag{4.5}$$

which is equivalent to

$$\bar{\mathbf{B}}\,\bar{\mathbf{p}} = 0$$

with $\bar{\mathbf{p}} = {}^{\mathrm{t}}(p_{11}, p_{21}, p_{31}, ..., p_{33})$ and

$$\bar{\mathbf{B}} = \begin{bmatrix} l_2l'_1 & l_2l'_2 & l_2l'_3 & -l_1l'_1 & -l_1l'_2 & -l_1l'_3 & 0 & 0 & 0 \\ l_3l'_1 & l_3l'_2 & l_3l'_3 & 0 & 0 & 0 & -l_1l'_1 & -l_1l'_2 & -l_1l'_3 \end{bmatrix}.$$

Note that $\bar{\mathbf{p}}$ includes the elements of the transponated matrix ${}^{\mathrm{t}}\mathbf{P}$, while in the derivation in section 4.3.1 vector $\mathbf{p}$ was used. In order to combine the computation for lines and points, a matrix $\mathbf{B}$ which is compatible to $\mathbf{p}$ instead of $\bar{\mathbf{p}}$ is needed. By rearranging the elements of $\bar{\mathbf{B}}$, the matrix

$$\mathbf{B} = \begin{bmatrix} l_2l'_1 & -l_1l'_1 & 0 & l_2l'_2 & -l_1l'_2 & 0 & l_2l'_3 & -l_1l'_3 & 0 \\ l_3l'_1 & 0 & -l_1l'_1 & l_3l'_2 & 0 & -l_1l'_2 & l_3l'_3 & 0 & -l_1l'_3 \end{bmatrix} \tag{4.6}$$

is obtained. Now, (4.3) and (4.6) can be used to define one single system of equations to compute $\mathbf{P}$.

4.3.3 Solving the Equation

To get $\mathbf{p}$ and by that the transformation matrix $\mathbf{P}$, a solution for all equations

$$\mathbf{B}_i \, \mathbf{p} = 0 \tag{4.7}$$

for line and point correspondences needs to be found. Let $\mathbf{A}$ be the matrix with dimension $2n \times 9$ that consists of all $\mathbf{B}_i$ stacked on top of each other. If exactly 4 point/line correspondences (thus 8 equations) are given, $\mathbf{A}$ has rank 8 and thus has a one dimensional null-space.[1] One eigenvector provides the solution for $\mathbf{p}$ up to a scaling factor.

This does not change if the system gets overdetermined which means more than 4 correspondences are available. However, if those correspondences are inexact due to noise, this is not true anymore and no exact solution can be found. In this case it is necessary to find an approximate solution for $\mathbf{p}$ that minimizes the cost function given by

$$\sum_{i=1}^{n} (\mathbf{B}_i \mathbf{p})^2 = \sum_{i=1}^{n} {}^{\mathrm{t}}\mathbf{p} \, {}^{\mathrm{t}}\mathbf{B}_i \mathbf{B}_i \mathbf{p} = {}^{\mathrm{t}}\mathbf{p} \underbrace{\left(\sum_{i=1}^{n} {}^{\mathrm{t}}\mathbf{B}_i \mathbf{B}_i \right)}_{\mathbf{C}} \mathbf{p} \; .$$

The minimal solution is given by the eigenvector that corresponds to the smallest eigenvalue of matrix $\mathbf{C}$.

Since each $\mathbf{B}_i$ can origin either from lines or points, it might be useful to add a factor that depends on the kind of correspondence:

$$\alpha_i = \begin{cases} a & \text{if } i \text{ is point correspondence} \\ 1 & \text{else} \end{cases}$$

Then, $\mathbf{C}$ is given by

$$\mathbf{C} = \sum_{i=1}^{n} \alpha_i \, {}^{\mathrm{t}}\mathbf{B}_i \mathbf{B}_i \; .$$

If a is set to a value smaller than 1, line correspondences become more important in the computation compared to point correspondences. Since a line consists of a large number of pixels, it makes sense to give it a many times larger weight compared to a single point. For simplicity, all points are assigned the same weight $a = 5 \cdot 10^{-3}$.

Another reason for choosing a small weight for points is that optical flow is a lot more imprecise than finding lines in the segmented image. In other words, line correspondences are far more trustable.

[1] It has to be emphasized that the set of correspondences has to be non-trivial, e.g. no three points in one line.

4.3.4 Normalization

It is well known from the literature, that the accuracy of the homography estimation depends on the coordinate system in which the points and lines are expressed, more precisely its origin and scaling [17]. That is why a normalization is highly recommended in order to get best results for any image size and any region in the image. In this work, the simple *isotropic scaling* is used which rescales all axes by an equal factor. Since the higher effort of more complex methods does not lead to better results in most cases [17], they are not regarded here. Fig. 4.7 shows qualitative results achieved with and without normalization.

(a) Segmented image (b) Without normalization (c) With normalization

Figure 4.7: An example of the effect of normalization: (a) shows frame n of a rotating cube, while the other images show the estimation of segment 1 from the previous frame $n-1$ without (b) and with (c) normalization. It is easy to see that result (c) comes far closer to ground truth (a).

Points

The normalization process of point vectors consists of the following steps. At first, the homogeneous coordinates of the points are divided by its third entry, so that it becomes 1:

$$(x, y, z) \mapsto \left(\frac{x}{z}, \frac{y}{z}, 1\right)$$

Then, the x- and y-coordinates of all points are translated in a way that their center of weight reaches the origin. Furthermore, the coordinates are scaled so that the average distance of the points to the origin equals to $\sqrt{2}$. That way, the vectors $\mathbf{x}_i$ and $\mathbf{x}'_i$ are mapped to a new set of vectors $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$.

This procedure, consisting of a translation and a scaling, can be expressed by the matrix

$$\mathbf{N}_{\mathrm{P}} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\langle x \rangle \\ 0 & 1 & -\langle y \rangle \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s & 0 & t_{\mathrm{x}} \\ 0 & s & t_{\mathrm{y}} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

where $s := \langle\sqrt{x^2+y^2}\rangle$ is the mean value of the point's distances to the origin, $\langle x\rangle$ and $\langle y\rangle$ are the mean values of their x- and y-coordinates and $t_\mathrm{x} := -s\langle x\rangle$ and $t_\mathrm{y} := -s\langle y\rangle$. The same way, the matrix $\mathbf{N}'_\mathrm{P}$ for the points $\mathbf{x}'_i$ is computed.

After normalizing the point vectors as described, the homography matrix $\hat{\mathbf{P}}$ is computed with these values. The corresponding matrix $\mathbf{P}$ can then be obtained by

$$\mathbf{P} = \mathbf{N}_\mathrm{P}^{-1}\hat{\mathbf{P}}\mathbf{N}_\mathrm{P} \ .$$

Lines

To normalize a line vector $\mathbf{l}$, two points $\mathbf{a} = (a_1, a_2, a_3)$ and $\mathbf{b} = (b_1, b_2, b_3)$ of the corresponding line are considered. It is obvious that the normalized line goes through the normalized points. The homogeneous line vector $\mathbf{l}$ can be retrieved by computing the cross product of those points

$$\mathbf{l} = \mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2\,b_3 - a_3\,b_2 \\ a_3\,b_1 - a_1\,a_3 \\ a_1\,b_2 - a_2\,b_1 \end{pmatrix} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} . \tag{4.9}$$

Replacing $\mathbf{a}$ and $\mathbf{b}$ by their normalized vectors leads to

$$\hat{\mathbf{l}} = \hat{\mathbf{a}} \times \hat{\mathbf{b}} = \mathbf{N}\,\mathbf{a} \times \mathbf{N}\,\mathbf{b} \ ,$$

where $\mathbf{N}$ is defined in (4.8). It follows that

$$\begin{aligned} \hat{\mathbf{l}} &= \begin{pmatrix} s\,a_1 + t_\mathrm{x}\,a_3 \\ s\,a_2 + t_\mathrm{y}\,a_3 \\ a_3 \end{pmatrix} \times \begin{pmatrix} s\,b_1 + t_\mathrm{x}\,b_3 \\ s\,b_2 + t_\mathrm{y}\,b_3 \\ b_3 \end{pmatrix} \\ &= s \begin{pmatrix} a_2\,b_3 - a_3\,b_2 \\ a_3\,b_1 - a_1\,b_3 \\ s(a_1\,b_2 - a_2\,b_1) + t_\mathrm{x}(a_3\,b_2 - a_2\,b_3) + t_\mathrm{y}(a_1\,b_3 - a_3\,b_1) \end{pmatrix} . \end{aligned}$$

Inserting (4.9) eliminates the arbitrary point vectors so that the equation only depends on $\mathbf{l}$ and the normalization parameters t_x, t_y and s:

$$\hat{\mathbf{l}} = s \begin{pmatrix} l_1 \\ l_2 \\ sl_3 - t_\mathrm{x}l_1 - t_\mathrm{y}l_2 \end{pmatrix} = \mathbf{N}_\mathrm{L}\,\mathbf{l} \ ,$$

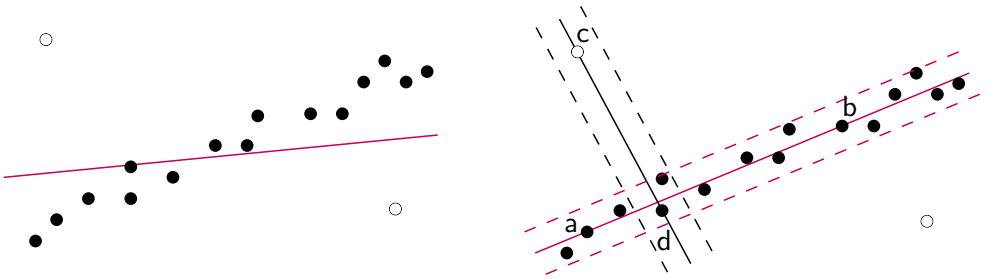
where

$$\mathbf{N}_\mathrm{L} = s \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_\mathrm{x} & -t_\mathrm{y} & s \end{bmatrix} .$$

4.3.5 Recognizing Outliers Using RANSAC

As already described, at least 4 correspondences are needed to compute the homography matrix. Since each correspondence obtained by e.g. optical flow usually has a certain error, the computation is done with far more values than technically necessary. By that, the matrix that fits best to all input values is obtained.

Until here, it has been assumed that the errors originate from the measurement of the point's position. This error is assumed to be Gaussian distributed and for all points equal. However, when using methods that are based on matching pixels between frames, one has to deal with mismatchings that can distort the estimated homography matrix a lot. It is a non-trivial task to find those *outliers* in order to ignore them in the computation. A commonly used algorithm for that kind of *robust estimations* is *RANSAC* standing for *Random Sample Consensus* [13].



(a) Best fit line of 2D points (b) RANSAC algorithm

Figure 4.8: Robust estimation of a line. (a) Outliers (○) can influence the best fit line a lot. In order to recognize outliers, the RANSAC algorithm (b) chooses two points randomly. Then, the *consensus set* that consists of all points within a range around the guessed line, is determined. This procedure is repeated several times until a line with a large *support* was found.

The algorithm has a very simple principle which will be explained regarding the easy example in Fig. 4.8(a). The task is to find a line that fits the correct 2D data points best while the outliers are ignored. Note that the number of outliers is unknown in general and could be larger than two.

The minimum number of points needed to define a line is 2. The algorithm selects consequently 2 points randomly as a *sample*. By them, a line is defined that might fit to the data points well – or poorly. The accuracy of this line is given by its *support* which is the number of data points lying within a threshold around it. Those matching data points are called *consensus set*. In order to

find the maximum consensus set the random selection is repeated several times. Regarding Fig. 4.8(b), it is obvious that a sample including outliers like c and d gets less support than a sample of solely inliers like a and b.

After some iterations the largest consensus set is deemed as inliers. The best fit line is then computed by using only those values while ignoring the outliers.

Computing a homography matrix using RANSAC is very similar. Since at least 4 correspondences are needed, the sample has to be of that size. As point and line correspondences are to be used together in this work, the random selector has to choose from both. This is done by first selecting a random value between 0 and 4 to define the number of lines within the sample. Then, that many lines are selected from the available line correspondences while the remaining number is chosen out of the points.

After selecting a random sample, a (temporary) homography matrix $\check{\mathbf{P}}$ is computed using (4.7) out of that 4 correspondences. Then, the distance of each available correspondence is computed using

$$d(\mathbf{x}'_i, \check{\mathbf{P}}\,\mathbf{x}_i) + d(\mathbf{x}_i, \check{\mathbf{P}}^{-1}\,\mathbf{x}'_i) \tag{4.10}$$

for points and

$$d(\mathbf{l}'_i, {}^{\mathrm{t}}\check{\mathbf{P}}^{-1}\,\mathbf{l}_i) + d(\mathbf{l}_i, {}^{\mathrm{t}}\check{\mathbf{P}}\,\mathbf{l}'_i) \tag{4.11}$$

for lines, which is called the *symmetric transfer error*. It is important to note that the distances have to be computed in inhomogeneous coordinates because two points in homogeneous coordinates can have a large euclidean distance although their corresponding image coordinates are close. Depending on the computed distance and the thresholds τ_{points} and τ_{lines}, each correspondence is classified as inlier or outlier in order to obtain the support of the sample.

As described in section 4.3.3, lines are given a larger weight than points when computing the homography matrix since they are firstly more precise and secondly representing a large number of points. This has also to be considered when computing the support of a sample. For that reason, a line that is recognized as inlier counts as much as 200 point inliers which each count 1. As a consequence, a sample that matches a certain number of line correspondences gets much higher support than a sample that matches the same number of points.

The whole algorithm can be summarized by the following steps:

1. Randomly select a number l between 0 and 4.

2. Randomly select l line correspondences and $4 - l$ point correspondences. Those four values form the sample.

3. Compute from the sample the homography matrix $\check{\mathbf{P}}$ using (4.7).

4. Compute the distance of each correspondence using (4.10) and (4.11). Count the number of point correspondences c_{points} having a distance smaller than the threshold τ_{points}. Also count the number of line correspondences c_{lines} with a smaller distance than τ_{lines}. The support is then given by $c_{\text{points}} + 200 \cdot c_{\text{lines}}$.

5. Repeat the above steps several times.

6. Use the largest consensus set to compute the final homography matrix $\mathbf{P}$.

There are some issues concerning the optimization of RANSAC. The number of necessary iterations can be estimated during the process in order to abort it as early as possible. Note that it is not necessary to try every possible sample what would cause a computational overhead. Instead of that, a minimum of iterations can be computed so that the probability of finding at least one sample without any outliers is very high.

Let w be the relative number of inliers. Then, the probability of choosing a sample (consisting of 4 correspondences) with at least one outlier is $1 - w^4$. If p is the probability of finding one sample without any outlier during N iterations, then

$$(1 - w^4)^N = 1 - p.$$

Let the desired value for p be 0.99, then the number N of necessary iterations is given by

$$N = \frac{\log(1 - 0.99)}{\log(1 - w^4)}.$$

The number of inliers w can be guessed or obtained by experiments. Another option is to update the value in each iteration when a larger consensus set was found. This temporarily largest support grows during the iterations and converges to the total number of inliers. Consequently, it is at each step a lower bound of the number of inliers. For that reason, w can be set to this value divided by the total number of correspondences.

There are some other modifications of the standard algorithm, namely *LO-RANSAC* [9] and *MSAC* [43]. Since this work does not intend to describe a highly optimized system, they are not considered further.

4.4 Extracting Boundary Lines from Segments

As mentioned in the previous sections, line correspondences can be used to estimate the homography matrix next to point correspondences. While the latter

are already given by optical flow, the lines have to be determined first in the segmented image. This is done using the *Hough transformation* [19, 41, 1].

This method is a feature extracting technique that allows finding imperfect lines in an image. There are also extensions of the classic algorithm that can be used to find arbitrary shapes like circles, ellipses and even predefined patterns. However, only the line extraction is used in the proposed framework and hence, those other methods will not be considered.

The following three sections briefly introduce the Hough transform for grayscaled and colored images. Then, some modifications are applied to optimize it for finding the boundary lines of segments, hence binary images.

4.4.1 Determination of Border Pixels

In a first step, the image is analyzed to find the pixels that belong to a border. This is usually done by computing its first derivate. High values indicate a large change of the color value which is regarded as a sign of a border, while close-to-zero values indicate a homogeneous region. Well-known implementations of this algorithm are the Roberts, Prewitt and Sobel operators [41]. Note that these operators only create edge maps that show where borders are. They do not provide any parametrization of the borders. It is often non-trivial to find the pixels that form a straight line, especially if those lines are corrupted due to noise, which is why the Hough transform was developed.

4.4.2 The Parameter Space

The Hough transform is based on a voting procedure that takes place not in the image itself but in a parameter space. A line is defined by a line equation $y = mx + b$, hence completely determined by the parameters m (the slope) and b (the interception of the line with the y axis). Since m and b equal infinity at lines parallel to the y axis, which is badly manageable, it is more beneficial to chose a different pair of parameters. It is very common to use the distance r of the line to the origin and its angle $\theta \in [0, \pi)$ as shown in Fig. 4.9. The line equation is then given by

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right) x + \left(\frac{r}{\sin\theta}\right) \ .$$

It is also an option to choose both vector components of the line's normal vector with length r instead of θ and r to avoid the computation of angles. However, the big disadvantage of this is that the accuracy of the line's orientation would depend on the line's distance to the origin.

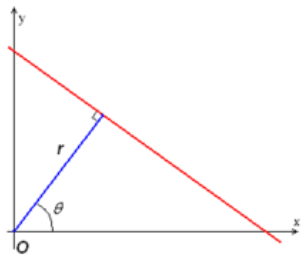


Figure 4.9: Parameters θ and r that are used to represent a line in the parameter space [1].

Now, considering a border pixel $\mathbf{x}$ which has been found by a border detector as described above, this pixel can possibly belong to lines with angles

$$\theta \in \Theta = \{\theta < \pi \,|\, \theta = n\Delta_\theta, n = 1, 2, ...\} \ ,$$

where Δ_θ is the angle step size that has to be chosen regarding accuracy and computational cost. The corresponding distance value is then given by

$$r_{\mathbf{x}}(\theta) = x \cos\theta + y \sin\theta \ . \tag{4.12}$$

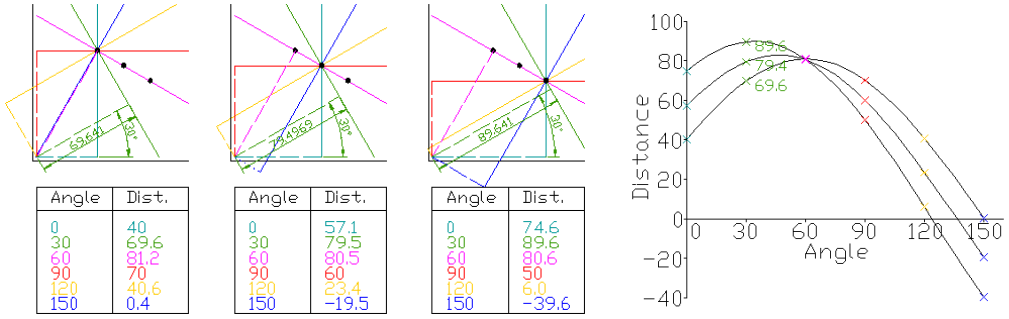
Then, each border pixel $\mathbf{x}$ gives a set of votes

$$\{(\theta, r_{\mathbf{x}}(\theta)) \,|\, \theta \in \Theta\} \ .$$

To store these votes, a two dimensional parameter space called *accumulator* is created. It is usually implemented as an array of size $2\sqrt{s_{\mathrm{x}}^2 + s_{\mathrm{y}}^2} \times |\Theta|$, thus twice the length of the image diagonal which corresponds to the maximum range of distances to the origin, times the number of angles.

For each vote, the corresponding entry in the accumulator is increased by one. The votes of one point correspond to a sinusoidal function as can be seen from (4.12). The corresponding curves of a set of collinear points will intersect in the parameter space at a point (θ, r), which holds the searched line parameters. Fig. 4.10 shows an example of the described procedure.

It is obvious that the computational time increases by decreasing Δ_θ, but this is necessary to get more precise lines. As an optimization, the gradient angle of the image can be used as θ, which dramatically reduces the number of votes that have to be processed down to 1 per border pixel. Unfortunately, this method cannot be applied to binary images like image segments, as can be seen in Fig. 4.11. The occurring gradient angles are $0, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \pm\pi, ...$, which is too imprecise. It is possible to perform a low-pass filtering as an upstream process to smooth the steps, but this would also lead to a loss of information. Since the



(a) Processing the border pixels (b) Accumulator

Figure 4.10: An example of filling the accumulator [1]. The black dots in (a) are border pixels. For each, votes in $30°$ steps are created and added to the accumulator (b). At $\theta \approx 60°$, there is a maximum (3 votes, one of each pixel) which indicates that a line of that angle and $r = 80\text{px}$ is present.

images processed in this work are of small size, the performance of the described standard algorithm is good enough and, for that reason, no optimization concerning computational time is done here.

4.4.3 Finding Peaks

After filling the accumulator, the next task is finding the peaks. A simple method for that is scanning the accumulator to find local maxima which mean elements that have the highest value in their neighborhood. Since the accumulator is periodic on the x-axis (angle), the neighborhood of a point at the border of the accumulator has to be continued at the opposite side.

Problems occur if the maxima are not point-shaped but form plateaus. This can be the case when the angle step size Δ_θ is small and many votes are found also in the neighbor elements. Furthermore, when coping with binary images, even for a perfect line, multiple peaks corresponding to angles of multiples of $\frac{\pi}{2}$ will be found as already stated in the previous section.

For that reason, the window size for searching the local maxima and merging close sub-peaks should be large enough to cover widespread peaks. On the other hand, this makes it hard to distinguish closely located peaks like in Fig. 4.14(b).

One heuristic solution for this is to make the window size dependent on the value of the current element. If an accumulator element is large, it might be part of a sharp maximum, so a small window size is chosen. At low values, the window size is increased to cover more neighbored elements. Even with only two

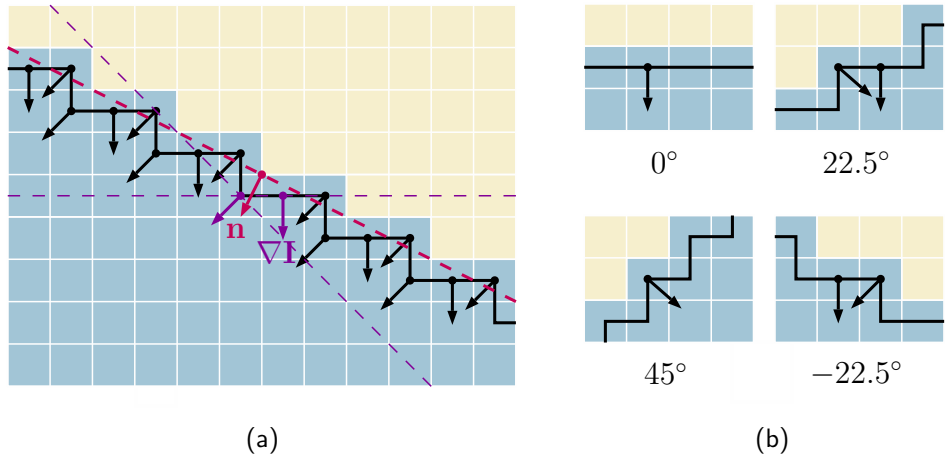


Figure 4.11: Directions of gradient in a binary image. In (a), $\mathbf{n}$ is the normal vector of the optimal approximation of the border, which is not fitting to any of the gradient vectors $\nabla\mathbf{I}$. Two gradient vectors are highlighted for comparison. (b) shows the occurring gradient directions for different line angles.

(static) window sizes, usable results can be achieved, but of course only to a certain extend.

4.4.4 Introducing Lines with Orientation

A better approach for this problem is to prevent the occurrence of narrow peaks in the first place. Although the red and the green line in Fig. 4.14(a) are close and by that, their θ and r are similar, they can be distinguished by the following property: the segment is in one case on the left side and in the other case on the right-hand side. One way to include that kind of additional information to the accumulator is extending it by a third dimension.

Instead of that, the angle range is expanded. In general, it is sufficient to regard angles between 0 and π, since lines corresponding to larger angles are equivalent. By introducing an orientation of the lines, $\theta = 0$ can be regarded as a line with the segment on the right-hand side, while $\theta = \pi$ has it on the left side. Consequently, $\theta = \frac{\pi}{2}$ indicates the line being at the top and $\theta = \frac{3\pi}{2}$ at the bottom (see Fig. 4.12).

When creating the votes, the relative position of the border pixel to the segment has to be considered. For that, the gradient vector's property of always pointing to the inside of the segment is used. As can be seen in Fig. 4.11(b), there is only a very limited range of gradient directions in one line. It is easy to see that the inversion is also true: The gradient vector constrains the range of

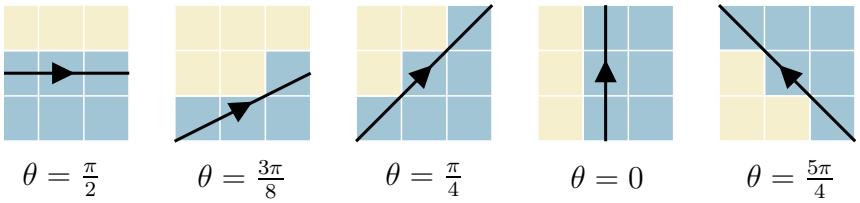
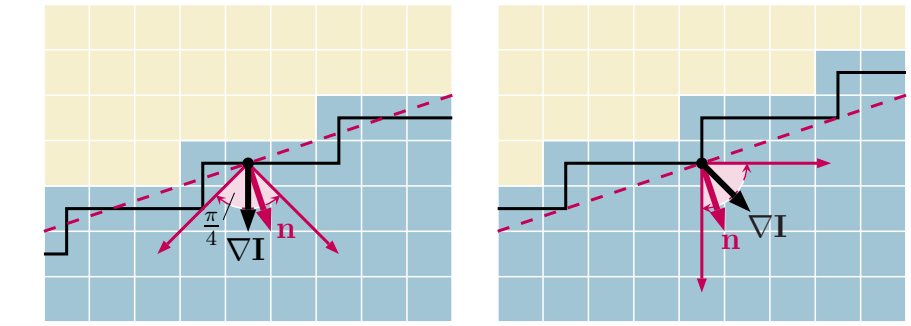


Figure 4.12: Lines with orientation for some angles.

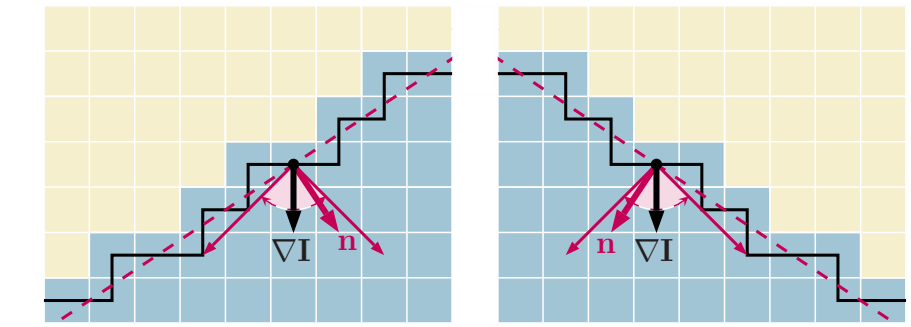
possible lines. E.g. a gradient vector pointing to the right will never be found in a horizontal line. It is important to note, that this is only true in "perfect" images like those shown. The segmented image of a real camera frame showing a horizontal border may have that kind of errors. But this is not a problem, since the Hough transform is stable against such isolated faults.

Fig. 4.13 shows, that the range of votes reduces to even only 90 degrees. Given the gradient vector at a border pixel, the normal vectors of all lines that the pixel could possibly belong to, fall into the range of $\pm\frac{\pi}{4}$ around it.

Hence, the proposed method reduces the number of votes for each border point by half, since the standard method creates votes in the range of 0 to π. On top of that, it uses orientated lines to enable the peak finding algorithm to distinguish close lines, as shown in Fig. 4.14(c).



(a) Vote range of two border pixels of the same line



(b) Identical gradient direction (and thus same vote range) in two different lines

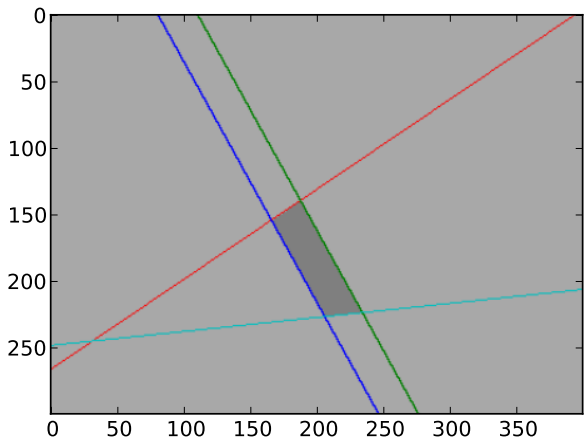
Figure 4.13: Votes depending on gradient direction $\nabla \mathbf{I}$. $\mathbf{n}$ is the normal vector of the optimal line. It is always in a range of $\pm\frac{\pi}{4}$ around the gradient.

4.4.5 Removing Distracting Holes in Segments

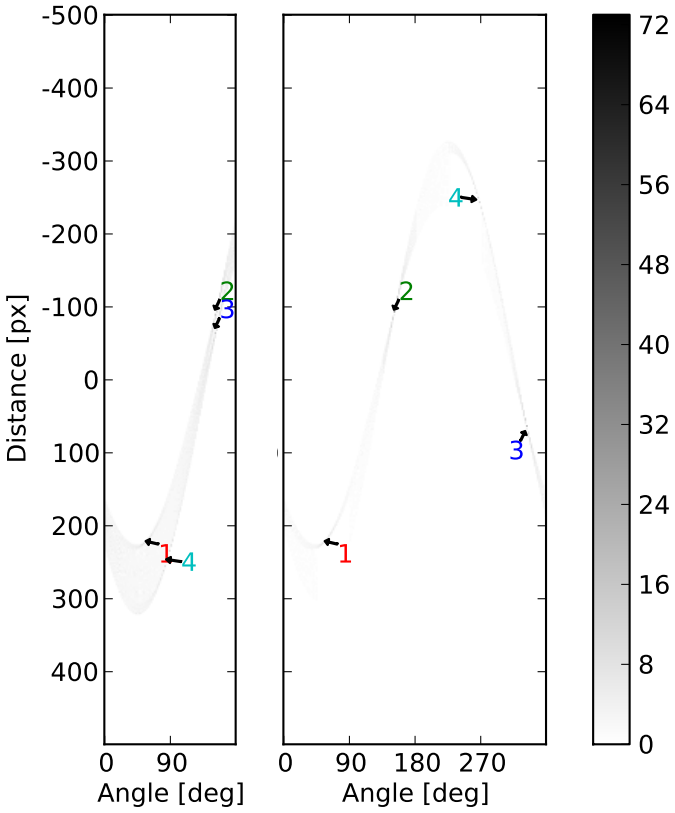
The strength of the Hough transform is the ability to find lines even in noisy images where lines might be for example disconnected. However, the algorithm cannot distinguish between wanted lines and those which arose from noise. Fig. 4.15(a) shows a segment that is not a homogeneous area but includes holes. As a result, the Hough transform finds an additional line (see Fig. 4.15(b)).

To solve this problem, a topological operation called *binary closing* is preprocessed [41]. It is a very efficient method to remove noise introduced by small holes. Fig. 4.15(c) shows the same segment after the operation. The inner noise has been completely removed while the boundary is almost unchanged.

The result of the Hough transform of the closed segment can be seen in Fig. 4.16. Note, that the original segment is shown instead of the closed version which is used only internally.

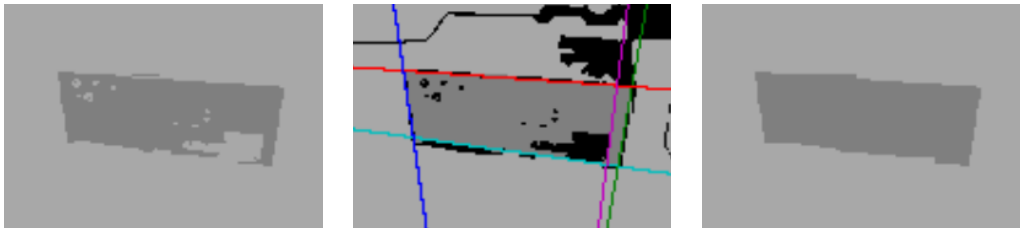


(a) Recognized lines



(b) Standard method (c) Proposed method

Figure 4.14: The peaks created by the standard Hough transform can be very close for parallel lines, as can be seen in (b). The modified algorithm solves this problem, the peaks in (c) are well separated.



(a) Original segment (b) Recognized boundary lines (c) Segment after "Closing"

Figure 4.15: Applying the binary closing operation to a segment in order to improve the Hough transform. See Fig. 4.16 for the final result.

4.4.6 Line Correspondences in Subsequent Frames

For each frame and each segment, the boundary lines are established as described in the previous sections. Since those lines are to be used to compute homography matrices between frames, line correspondences between subsequent frames are needed. For that reason, each new line is assigned a label which is then managed to be kept consistent over the whole sequence.

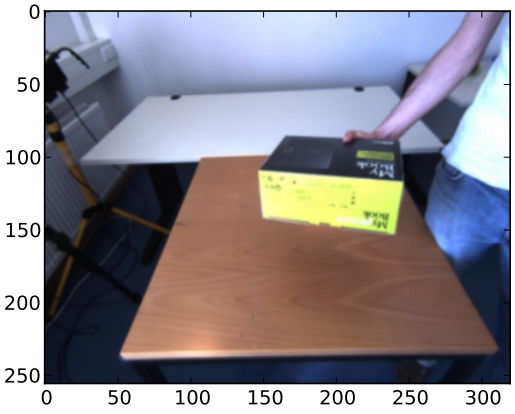
Instead of comparing the lines in the image space, this is done in the accumulator space. Obviously, similar lines have neighboring peaks. Given the peaks of one frame and those in the other, the set of peak-correspondences, which minimizes the sum of their distances, needs to be found. This is simply done by computing the distances of all combinations and choosing the smallest. Although this computation has a squared complexity in the number of lines, it is fast enough for the purpose of this work since in most cases the number of lines is less than 6.

Besides the matching, a maximum distance, that line peaks are allowed to have, is defined. This is important for cases when a new line appears while another line gets lost. Without a limitation of the distance, those lines might get wrongly matched.

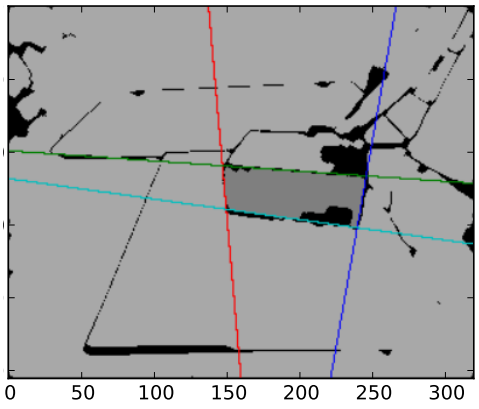
4.5 Transformation of Pixel Maps

Given the matrix $\mathbf{M}_n$ the data of the layer L can be transformed to the perspective of the current frame n. Theoretically, this is done by multiplying each pixel of L with $\mathbf{M}_n$ so that the set of transformed pixels is given by

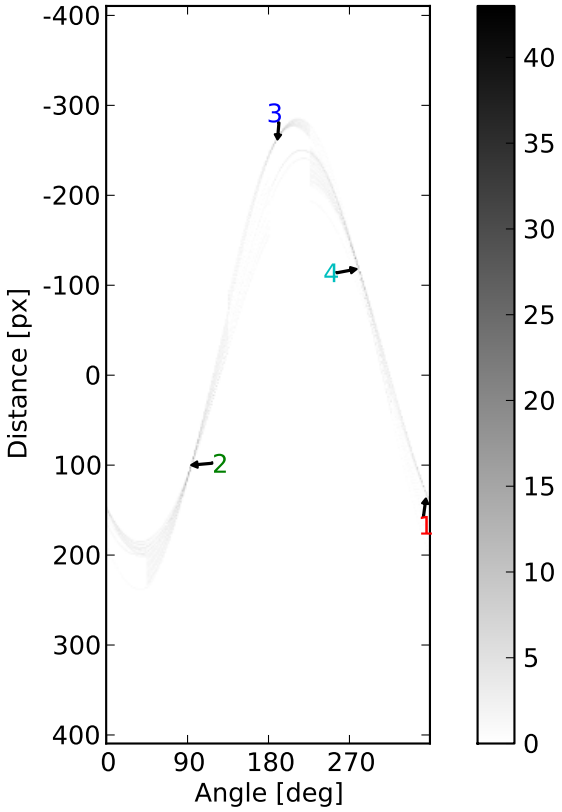
$$\{\mathbf{M}_n\mathbf{x} \,|\, \mathbf{x} \in L\} \ . \tag{4.13}$$



(a) Original image



(b) Recognized lines of segment



(c) Accumulator with peaks

Figure 4.16: An example of finding the boundary lines of a segment. The colors of the lines correspond to those of the peaks in the accumulator.

It is obvious that the total number of pixels before and after the transformation will not change. If $\mathbf{M}_n$ is e.g. a scaling, this fact will lead to holes in the transformed image since more pixels would be necessary to fill the complete shape.

For that reason, the transformation is done backwards. For each pixel in the target coordinate system it is checked, whether a pixel of the layer is mapped to it. In mathematical terms, this is given by

$$\{\mathbf{y} \,|\, \mathbf{M}_n^{-1}\mathbf{y} \in L\} \equiv \mathbf{M}_n[L] \ . \tag{4.14}$$

Note that more than one $\mathbf{x} \in L$ fulfill $\mathbf{M}_n^{-1}\mathbf{y} = \mathbf{x}$ since all variables are rounded to integer values. For simplicity, the notation $\mathbf{M}_n[L]$ will be used for this method.

When mapping the layer to the image, it is enough to check (4.14) only for the image coordinates, since layer pixels outside the image are not considered. So, in this case the range of putative values of $\mathbf{y}$ are known. In other tasks that will follow later in this work this range is unknown. Hence, it needs to be determined by first using (4.13) in order to find out the range of $\mathbf{y}$, and then using (4.14) within that range.

4.6 Keeping Layers Up-To-Date

Until here, the framework creates layers from the first frame and updates matrices that map the layers to the current frame. However, the layers' contents are not getting updated yet.

4.6.1 Adding Data

Obviously, the assumption that all segments are completely visible in the initial frame needs to be dropped. Fig. 4.17 shows the following example: Segment 1 is occluded in the first frame and, as a consequence, the layer contains only the visible parts. As the segment moves out, more parts become visible that need to be added to the layer.

The coordinates of the pixels, that newly appeared in the frame, have to be mapped to layer coordinates. This is done by the inverse mapping

$$\mathbf{M}^{-1} : \mathrm{Image} \to L \ .$$

The addition of new data to the layer can be seen as a union of sets. Let L be the data of the layer and S be the segment visible in the frame. Then, all pixels need to be transformed to the layers coordinate system, and then merged to the layer data. The new layer data is consequently given by

$$\{\mathbf{M}^{-1}\mathbf{x} \,|\, \mathbf{x} \in S\} \cup L \ .$$

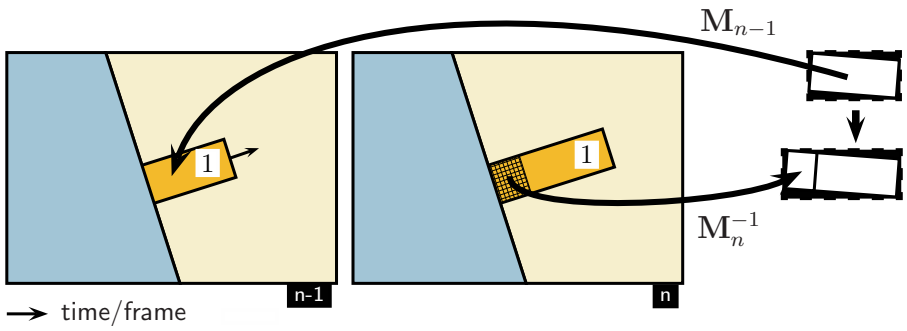


Figure 4.17: Adding new data to the layer using inverse mapping.

By that, new pixels are added to the layer, while already stored values are kept unchanged. If the segment in Fig. 4.17 moved back, the vanishing parts would remain in the layer.

Considering a big object like a wall or the ground captured by a moving camera, the corresponding layer can grow bigger than the frame size.

4.6.2 Removing Noise

A more complex task related to updating the layer data is the removal of noise. First, the term "noise" needs to be clarified. The motivation for using layers is to be able to remember parts of segments which are currently invisible. By mapping a layer to the current frame the original boundaries (or at least all parts which have already been visible at some time) of the object can be restored. Noise refers in this context to layer pixels that are mapped to parts of the image that do not represent the actual object (this could be called a "wrong memory").

There are two reasons that can lead to such erroneous pixels in the layers:

- Segments are noisy especially at the borders. The boundaries of even static objects are never perfect but contain flickering pixels.
- The mapping of a layer to the frame is not perfect in most cases since the homography estimation is erroneous. For that reason, new data might get falsely added to the layer.

Unfortunately, there is no chance to get an unfailing algorithm that prevents those errors. Instead, a method is needed that enables the layers to "forget". One approach is deleting parts of the layer after a period of time. For each layer pixel a counter would have to be created that is increased if the pixel is invisible in the

current frame. As soon as a counter reaches a maximum value the corresponding pixel of the layer is deleted. However, the counters of visible pixels are reset back to zero. Using that method, visible pixels are kept in the layers while occluded parts are stored only for a limited amount of time.

It is obvious that the time threshold is a crucial value. The quicker the noise is supposed to be deleted the shorter becomes the memory of the system. Or in other words: The system deletes wrong values as fast as correct values (invisible pixels) since it cannot distinguish between them.

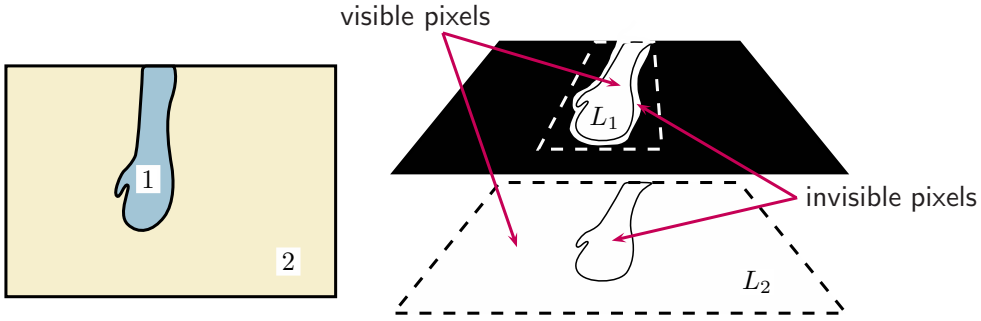


Figure 4.18: Parts of the layers can be invisible. This can be due to occlusion or due to noise and distiguishing between those two cases can be done using depth information.

A better approach is solving that last issue. Fig. 4.18 shows a segmented frame and the corresponding layers. Layer 1 contains noise around the boundary of the segment which is marked by a black line. Layer 2 contains pixels even at parts where segment 1 is currently located. It is obvious that the hidden parts of layer 2 should be kept while the noisy parts in layer 1 should be deleted.

This can be done by taking the depth order of the segments into account. As a first step, the layer is mapped to the current frame. Some pixels of the layer are invisible which means that they do not match the corresponding segment. Instead, those pixels are mapped to a different segment. Now, there are two options: either this segment is above and occludes everything beneath, or this segment is below. In the last case the pixel must obviously be a false pixel since being occluded by a underlying object does not make sense.

Regarding the example in Fig. 4.18, the invisible pixels of layer 1 are mapped to segment 2. Given the information that segment 1 is above 2, those pixels are identified as noise. The invisible pixels of layer 2 are mapped to segment 1 which is a real occluder since it is above. Hence, those values are kept.

The depth information can either be taken from stereo vision, a 3d camera or from the method described in section 4.7. In order to keep the layer framework

applicable for monocular image sequences and independent from special sensors, the last option is used.

4.7 Detecting Depth Relations of Segments

For some parts of the framework it is necessary to know the depth order of segments. Regarding Fig. 4.19 it is obvious that one single (segmented) frame does not contain enough information to recognize that segment 1 that might be a hand is above 2, e.g. a table. Of course, based on the knowledge of how hands and tables look like, a human is in most cases able to figure out the correct depth determination even in single images (e.g. photos). But this prior experience is additional information that is not contained in the image itself. Next to recognition of objects, humans also use other hints like lightening/shadows in order to get an idea of depth. There have been many investigations of implementing methods like those on a robot, namely *object recognition* and *shape from shading*.

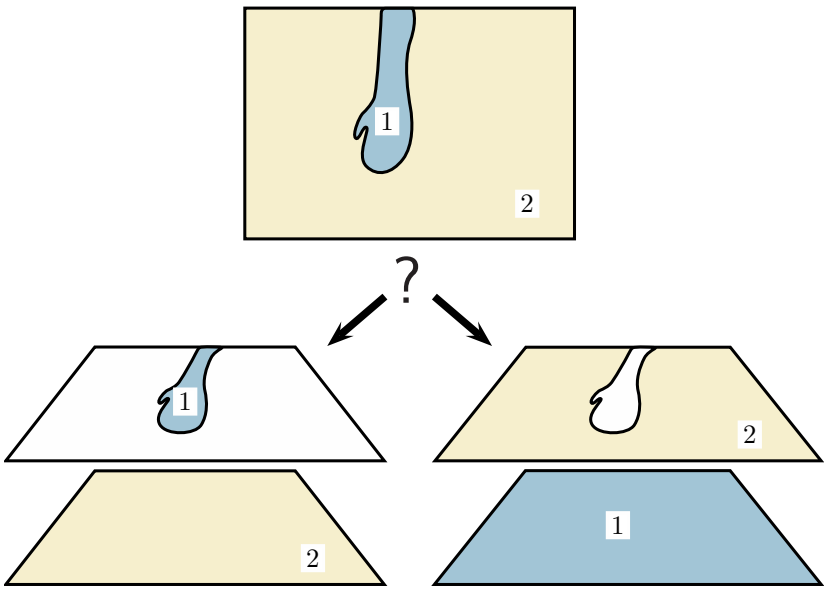
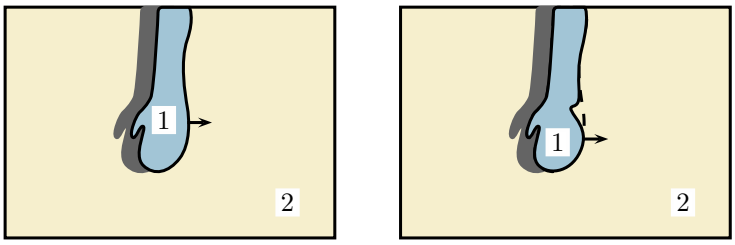


Figure 4.19: Regarding one single frame the depth order of the segments cannot be extracted. Both situations are a-priori possible: The hand (1) is above the table (2) or the table has a (hand-shaped) hole that allows the view to an underlying object (1).

Since the framework developed in this work is intended to be model-free, any method based on object recognition is not an option. Taking the shading of the scene into account is a complex task and often too erroneous. That's why an alternative hint on depth is used: motion. Since the framework tracks the segment's movements it can be extended to derive depth information. This is related to *structure from motion* [17] that obtains the 3D shape of an object by observing it while it rotates. This is used e.g. in some 3D scanners. The essential difference to our situation is that the real-world movement is unknown here while a scanner defines the objects' rotation itself. However, since no 3D shape but only the depth order between segments (which one of two is above the other) is intended to be derived, this task is a lot simpler.

4.7.1 Basic Idea

Since it is assumed that segments correspond to solid objects, changes of the shape can only happen either due to the object's movement/rotation in the 3D space, or due to occlusion. Using that assumption, the depth order of segments can be derived from their movement.



(a) Ideal example (b) Unstable boundary

Figure 4.20: (a) As the hand 1 moves, parts of the table 2 become visible while others become occluded (b). If the segment's boundary of the occluding object is unstable, it is more complex to decide about the depth order.

Consider two objects that move relative to each other. If the first object is above the second, the shape of the corresponding segment should not change. The segment of the second, underlying object however does change, since new parts of the object become visible and others become invisible as the occluder moves. Let Fig. 4.20(a) be the next frame of the example in Fig. 4.19. Segment 1 has moved to the right and parts of the table that were previously occluded are visible now, while others are now occluded. This observation can only be explained, if segment 1 is above 2.

The basic algorithm can be described as follows: Between two subsequent frames, it is checked for each segment if there are regions that were occluded by another segment before. Additionally, regions, that are occluded now by another segment but were visible before, are determined. This observation will be called *change of visibility*. If such regions are found, the object belonging to the segment seems to be beneath the other. Unfortunately, this simple algorithm gets into trouble if the boundaries of the occluding segment are unstable, as can be seen in Fig. 4.20(b). A small part of segment 1 gets lost and changes the label to 2. This noise effect must not be mixed up with the case where a part of the table occludes the hand. Instead of that, the algorithm has to choose the most probable constellation: Although there could be indeed a small part of the table occluding the hand, there are much more hints (=regions with changes of visibility) that the hand is above the table. Note that an important assumption is made here: One segment can be either above or beneath another, but not both. If the hand is above one part of the table and another part is beneath, it is assumed that those two parts of the table would be represented by two different segments. Although there might be cases violating this assumption they are improbable since the lightning condition will cause the color-based segmentation to separate parts/surfaces of objects that are very different in position and orientation.

As an additional way of sorting out changes of visibility caused by noise, the proposed algorithm checks whether the newly occluded/newly invisible regions can be "explained" by the movement of the occluder. The whole algorithm will be described in the following section.

4.7.2 Implementation

Until now, the layers are used only to store the segment's shape. In order to find regions with changes of visibility, the layers have to be extended to store parts of the layer which are occluded. This is illustrated in Fig. 4.21. To each layer L, two more sublayers of the same size are added. The first sublayer (a) holds the layer data as described before. The second sublayer (b) marks those parts which are occluded by another segment. More precisely, the pixels of sublayer (b) include the label of the occluding segment. If there is no occlusion, the corresponding pixels have value 0. The sublayers are named $L_{s\mathrm{a}}$, $L_{s\mathrm{b}}$ and L_{sc} as an example for a segment s.

In the figure, the layer $L_{1\mathrm{a}}$ of segment 1 has parts which are invisible in frame $n-1$ since segment 2 is located in that area. According to that, $L_{1\mathrm{b}}$ gets assigned the number 2 in that area. Note that the sublayer (b) is not binary contrary to (a) since each pixel needs to store the label of the segment that causes the occlusion. Also note that the sublayer $L_{2\mathrm{b}}$ of segment 2 is empty because no occlusion is present.

The third sublayer (c) contains the data of sublayer (b) at the previous frame which means that the data is moved from the second to the third sublayer in each time step. The data of the second sublayer is computed by mapping the first sublayer to the image and determining the areas that are not covered by the segment which means occlusion.

The changes of visibility can then be derived using the data of sublayer (b) and (c). At first, a new notation has to be introduced in order to use set syntax. As defined in section 4.1, the first sublayer can be seen as a set of coordinates of elements with value *True*. Since the other sublayers are not of type boolean but integer, it is required to specify the element value in set notation. This is done using the "|" symbol, similar to the notation of conditional probability. As an example, all elements of sublayer (b) of segment s having the value v will be annotated as

$$L_{s\mathrm{b}|v} \ .$$

The determination of the changes of visibility is done segment-wise. For each segment s in the current frame n, the corresponding sublayer (b) is searched through in order to find segments currently occluding the layer. For each occluder k of s the change of visibility from *invisible* to *visible* is obtained by

$$C_s^{k*} = \overline{L_{s\mathrm{b}|k}} \cap L_{sc|k} \ . \tag{4.15}$$

Furthermore, the parts that changed from *visible* to *invisible* are given by

$$C_s^{k\dagger} = L_{s\mathrm{b}|k} \cap \overline{L_{sc|k}} \ . \tag{4.16}$$

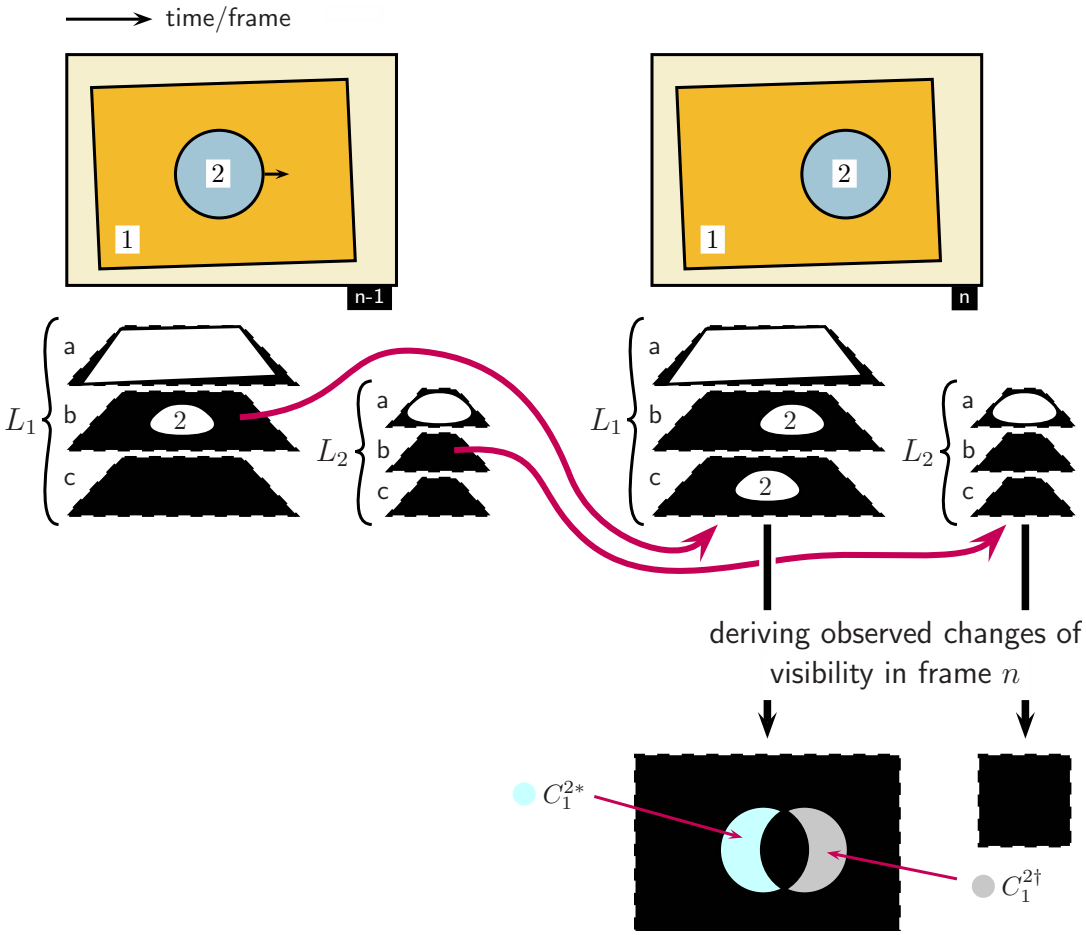


Figure 4.21: The layers are extended into three sublayers. While the first sublayer stores the original layer data, the second holds information about occlusions in the current frame and the third in the previous frame. The observed changes of visibility can then be derived. Colors: ● from *visible* to *invisible*, ● from *invisible* to *visible*.

In noise-free cases it would be enough to find those changes of visibility to decide, that segment s is below k. Since the segment's shapes are noisy in most cases, those visibility changes can also occur without any relation to the movement of an occluding segment as shown in Fig. 4.20(b).

For that reason a control mechanism is introduced in the next section that finds putative areas where a change of visibility can occure. Changes in other areas are then ignored and the algorithm becomes more stable.

The Occluding Segment is Moving

The control mechanism is illustrated in Fig. 4.22 as continuation of Fig. 4.21. The layer of the occluding segment k, which is 2 in the example, is transformed using the current and the previous transformation matrix $\mathbf{M}_n^k$ respectively $\mathbf{M}_{n-1}^k$ which gives its appearance in the current and the previous frame. This simulates the movement of the occluder based on the available layer data. The theoretical change of visibility is then given by

$$\begin{aligned} D_s^{k*} &= \overline{\mathbf{M}_{n-1}^k[L_{k\mathrm{a}}]} \cap \mathbf{M}_n^k[L_{k\mathrm{a}}] \\ D_s^{k\dagger} &= \mathbf{M}_{n-1}^k[L_{k\mathrm{a}}] \cap \overline{\mathbf{M}_n^k[L_{k\mathrm{a}}]} \ . \end{aligned} \tag{4.17}$$

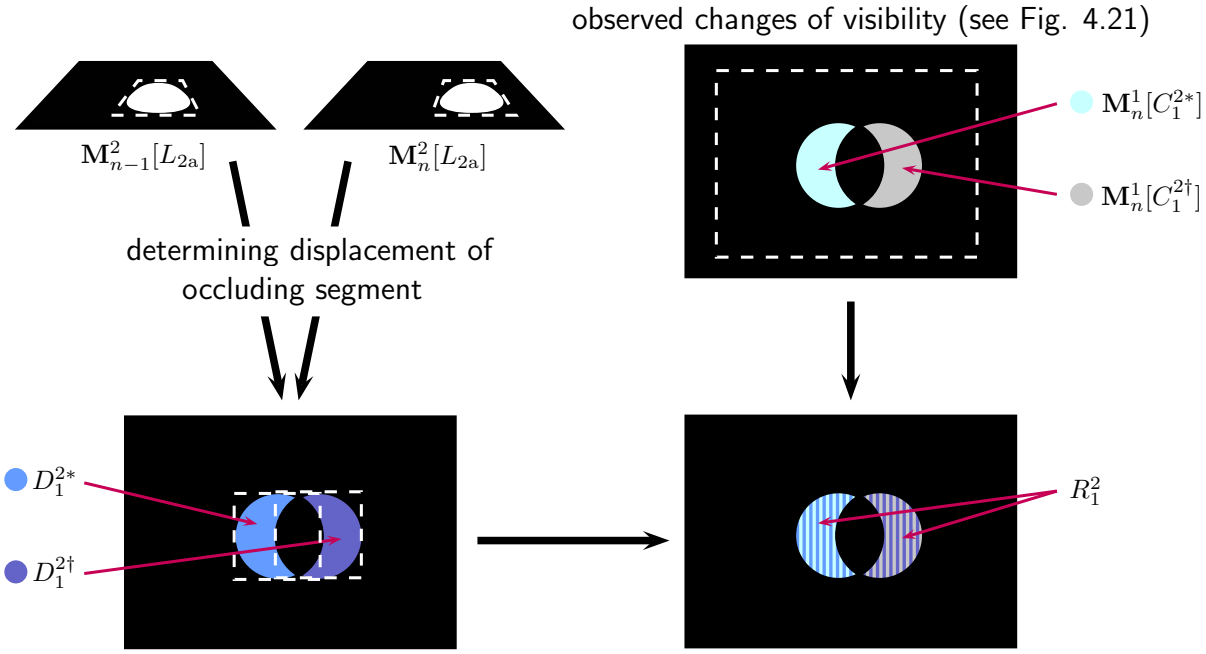
After that, the observed values that match the theoretical values can be seen as "resolved". Since D_s^{k*} and $D_s^{k\dagger}$ are in image coordinates due to the transformation, the observed values C_s^{k*} and $C_s^{k\dagger}$ also have to be transformed to image coordinates. The set of resolved pixels for k being above s is then given by

$$R_s^k = \left(\mathbf{M}_n^s[C_s^{k*}] \cap D_s^{k*}\right) \cup \left(\mathbf{M}_n^s[C_s^{k\dagger}] \cap D_s^{k\dagger}\right) \ .$$

This computation is done for all segments s in the image.

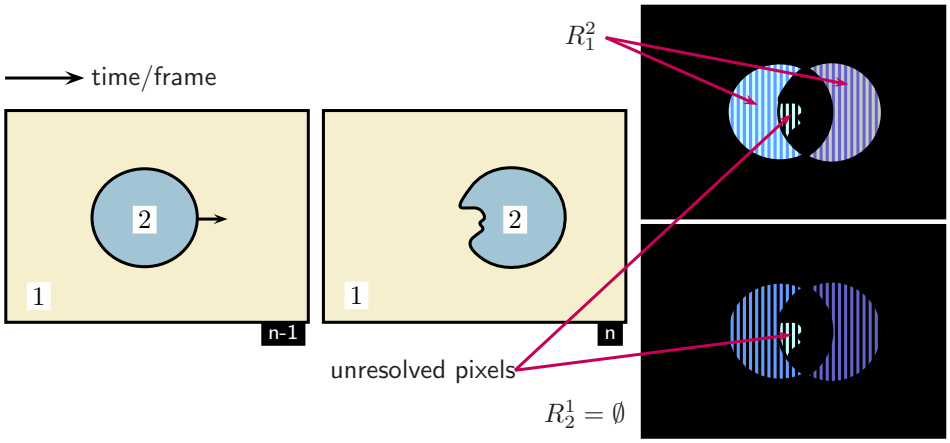
Fig. 4.23 shows the effect of the control mechanism. Segment 2 moves to the right. Due to some noise, the label of background (namely 1) is assigned to a little part of segment 2. Consequently, changes of visibility are observed on both segments 1 and 2. On the right, the observed changes of visibility are shown in light blue and light gray while the theoretical areas are shown in darker blue tones (see image caption for a color description). The areas where theoretical and observed values match are visualized blue/white-striped. Note that the visibility change caused by the noise is outside of the allowed area. As a result, the set of resolved pixels R_2^1 for 1 above 2 is empty, while R_1^2 standing for 2 above 1 is large. Consequently, the latter depth relation is recognized as being correct.

The described control mechanism can only filter out changes of visibility that conflict with the segment's movements. In Fig. 4.24 an example is shown where noise occurs within the allowed area. In that case, resolved pixels are found for 1 being above 2 as well as for 2 being above 1. However, the right choice can be made by taking the number of pixels $|R_2^1|$ and $|R_1^2|$ into account. Obviously, the depth relation, which could be verified by more resolved pixels, is correct.



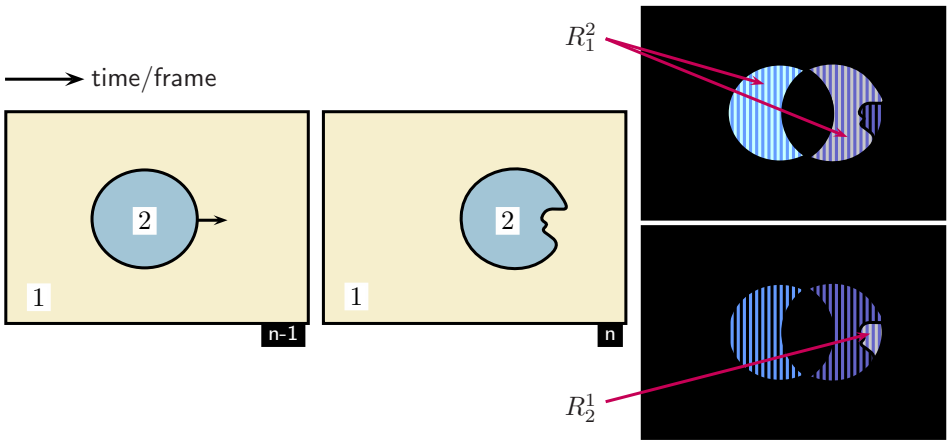
(a) Theoretical changes of visibility (b) Verification of observed values

Figure 4.22: (a) Determination of areas where a change of visibility is allowed to happen. (b) The observed changes are then compared to them and ignored if they do not match. If they do match (overlap of ●/●, ●/●, visualized as stripes) the observed changes are called "resolved". In this case, segment 2 seems to be above 1. Colors: ● from *visible* to *invisible* (● theoretical), ● from *invisible* to *visible* (● theoretical).



(a) Frames (b) Verification

Figure 4.23: Example where some of the observed changes of visibility conflict with the movement and hence can be filtered out by the control mechanism. The verification part (according to Fig. 4.22(b)) is shown in (b) for 2 being above 1 and 1 being above 2. Colors: ● from *visible* to *invisible* (● theoretical), ● from *invisible* to *visible* (● theoretical), stripes indicate the presence of both colors.



(a) Frames (b) Verification

Figure 4.24: Example where the changes of visibility caused by noise do not conflict with the movement, hence lie in the allowed area. The verification part is shown in (b) for 2 being above 1 and 1 being above 2. Since the number of resolved pixels is higher for 2 being above 1 ($|R_1^2| > |R_2^1|$), the depth relation is still correctly recognized. Colors: (see Fig. 4.23)

The Occluded Segment is Moving

Equation (4.17) does not depend on s. The reason for that is that only the movement of the occluder was taken into account. Fig. 4.25 shows an example where the occluded segment moves while the occluder is static. The resulting changes of visibility on segment 1 would all be ignored since the occluder 2 does not move and consequently $D_1^{2*} = D_1^{2\dagger} = \emptyset$. That is why (4.17) needs to be extended so that it also checks the movement of the occluded segment s.

This computation is very similar to the case with the moving occluder. As described, the sublayer (a) of the occluder is transformed both to the current and back to the last frame to simulate its movement. For the case, where the occluded segment is moving, the same is done, but for sublayer (c) of the occluded segment itself (see Fig. 4.25). By that, the movement of the occluded area is simulated and then compared to the actual observation.

The area of allowed visibility changes caused by the movement of s is added to (4.17) which leads to

$$\begin{aligned} \bar{D}_s^{k*} &= \left(\overline{\mathbf{M}_{n-1}^k[L_{k\mathrm{a}}]} \cap \mathbf{M}_n^k[L_{k\mathrm{a}}]\right) \cup \left(\overline{\mathbf{M}_{n-1}^s[L_{sc|k}]} \cap \mathbf{M}_n^s[L_{sc|k}]\right) \\ \bar{D}_s^{k\dagger} &= \left(\mathbf{M}_{n-1}^k[L_{k\mathrm{a}}] \cap \overline{\mathbf{M}_n^k[L_{k\mathrm{a}}]}\right) \cup \left(\mathbf{M}_{n-1}^s[L_{sc|k}] \cap \overline{\mathbf{M}_n^s[L_{sc|k}]}\right) . \end{aligned} \tag{4.18}$$

It is obvious that $\bar{D}_s^{k*} \cap \bar{D}_s^{k\dagger} = \emptyset$ should be fulfilled since the visibility can only change from visible to invisible or vice-versa, but not both. However, this is not always fulfilled when using (4.18). For that reason, those wrong values are filtered out by

$$\begin{aligned} D_s^{k*} &= \bar{D}_s^{k*} \cap \overline{\bar{D}_s^{k\dagger}} \\ D_s^{k\dagger} &= \bar{D}_s^{k\dagger} \cap \overline{\bar{D}_s^{k*}} . \end{aligned} \tag{4.19}$$

An example, where this is necessary, are two segments that move synchronously. In this case, no changes of visibility will be observed since the upper segment always occludes the same parts of the lower segment. Consequently, there should be no theoretical changes either. However, (4.18) will allow visibility changes in some areas. If noise occurs in exactly those areas, the algorithm might recognize a (wrong) depth relation, although no such relation can be obtained.

Fortunately, those wrong theoretical changes of visibility in that kind of situations can easily be prevented using (4.19). Unfortunately, this leads to inaccuracy in other situations, as shown in the next chapter.

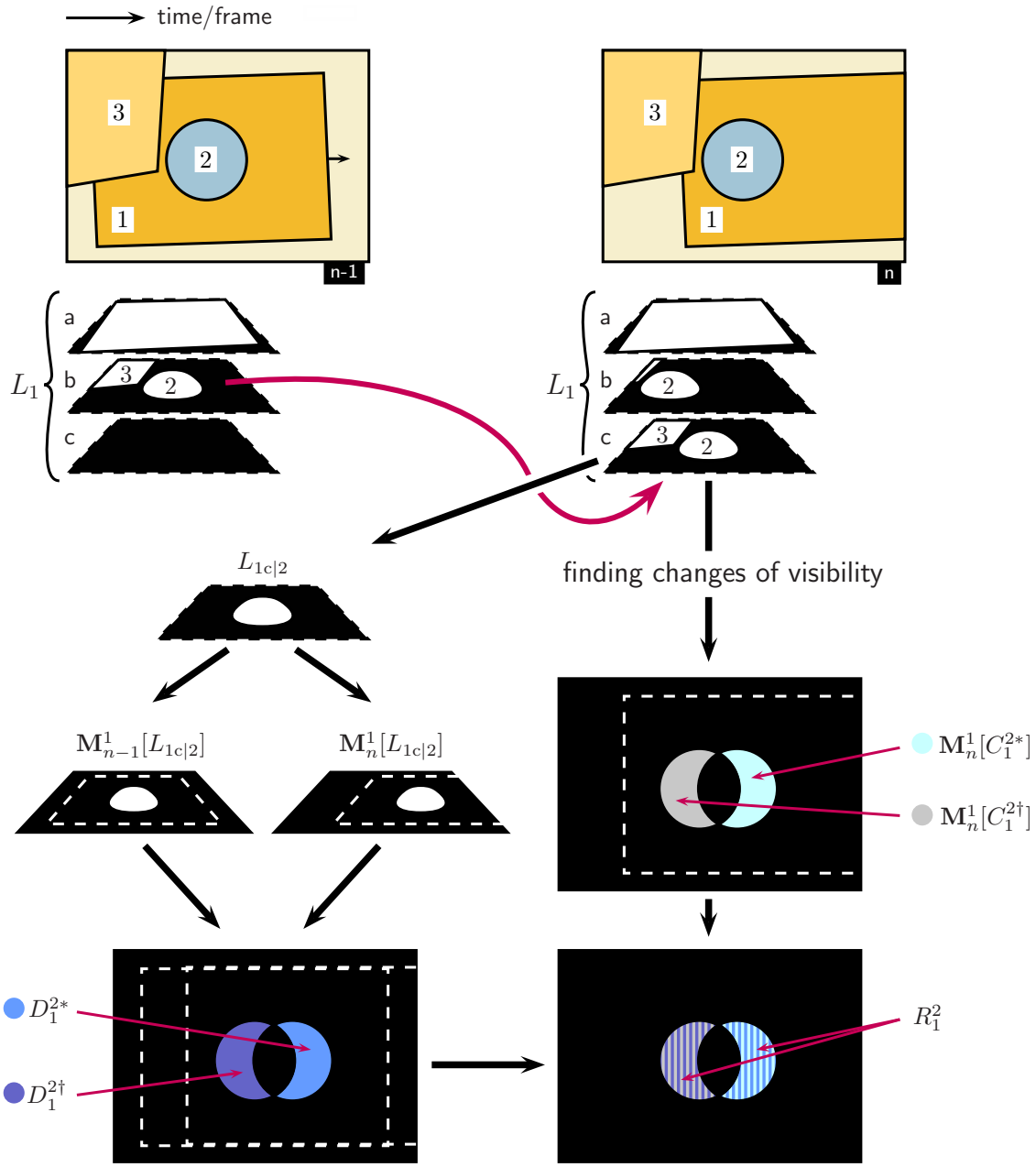


Figure 4.25: Computation of theoretical changes of visibility when the occluded segment is moving while the occluder is static. $L_{1c|2}$ is transformed to the current and to the previous view to simulate the movement of the occluded area of segment 1. Segment 3 was added in order to illustrate that sublayer (b) can also include more than one occluder. Colors: ● from *visible* to *invisible* (● theoretical), ● from *invisible* to *visible* (● theoretical), stripes indicate the presence of both colors.

The Occluded and the Occluding Segment Are Moving

In the previous method it is assumed that either the occluding or the occluded segment is moving relative to the camera. Nevertheless, the concurrent movement of both segments is in most cases also covered. The advantage is that the computations are very simple and, furthermore, can be cached. Note that

$$\mathbf{M}_n^s[L_{sc|k}] = \mathbf{M}_n^s[L_{sc}]|k$$

which means that a sublayer can be transformed first and then be converted to a binary array that marks all pixels of value k. Because of that, (4.18) can be programmed in a way that all transformations through $\mathbf{M}_n^s$ and $\mathbf{M}_{n-1}^s$ have to be computed only once for each segment, but not for all combinations of two segments. More precisely, the left part of (4.18) has to be computed for each k but is same for each s. The same is true for the right part concerning s and k, apart from the simple $|k$-operation. By that, a squared complexity can be prevented at least concerning the transformation, which is the most time consuming part.

The problem of the previous method is that the movements of the occluding and the occluded segment are processed separately, and then merged by a simple union, given by (4.18). This works in many situations, at least when applying (4.19), but is not an exact simulation of what really happens. Instead of tracking the absolute movements (which means relative to the camera) of both segments, their relative movement needs to be regarded in order to get exact results. However, this implies a squared complexity. If computational time is not important and precise results are needed, the now proposed method can be used.

It goes back to the sole movement of the occluder. The corresponding theoretical changes of visibility are given by (4.17). The right term (which is $\mathbf{M}_n^k[L_{k\mathrm{a}}]$ respectively $\overline{\mathbf{M}_n^k[L_{k\mathrm{a}}]}$) represents the position of the occluder k in the current frame and, by that, the currently occluded parts of segment s.

The left part (which is $\overline{\mathbf{M}_{n-1}^k[L_{k\mathrm{a}}]}$ respectively $\mathbf{M}_{n-1}^k[L_{k\mathrm{a}}]$) represents the position of the occluder k in the previous frame. This is equivalent to the formerly occluded parts of the occluded layer only if the latter did not move between the previous and the current frame. On the contrary, if it did move, the occluder has to be moved the same way. This transformation between frame $n-1$ and n is given by $(\mathbf{M}_{n-1}^s)^{-1} \cdot \mathbf{M}_n^s$. The area of allowed visibility changes is then given by

$$\begin{aligned} D_s^{k*} &= \overline{((\mathbf{M}_{n-1}^s)^{-1} \cdot \mathbf{M}_n^s \cdot \mathbf{M}_{n-1}^k)[L_{k\mathrm{a}}]} \cap \mathbf{M}_n^k[L_{k\mathrm{a}}] \\ D_s^{k\dagger} &= \underbrace{((\mathbf{M}_{n-1}^s)^{-1} \cdot \mathbf{M}_n^s \cdot \mathbf{M}_{n-1}^k)[L_{k\mathrm{a}}]}_{\text{previously occluded pixels}} \cap \overline{\mathbf{M}_n^k[L_{k\mathrm{a}}]} \ . \end{aligned}$$

In order to get precise results for all kinds of relative movements, this method is used in the framework.

Obtaining Depth Relations with Incomplete Layer Data

The described method assumes that the occluded parts of one layer are already known when determining the changes of visibility. In other words, the sublayers b and c that hold the needed information about hidden parts, have to be filled so that (4.15) and (4.16) can be used. Unfortunately, this is not the case if the occluded parts of one segment are still unknown to the system. An example for that situation is Fig. 4.17 where new data is added to the layer. Until here, the system does not know where those new parts come from. However, due to the assumption that segments belong to solid objects, it is obvious that the newly visible part of the segment had been occluded before.

In order to enable the framework to come to this conclusion, the layer system is modified. While layers have been kept as small as possible till here, they are now enlarged so that the sublayers are able to store not only the currently occluded segments but also the segment configuration in a small neighborhood. If new parts of a segment become visible in this neighborhood, the sublayer already contains which segment had been at this place before. That way, this segment is identified as a former occluder. The increase of the layer's size has to be adjusted to the expected speeds in the image sequence. If only slow speeds are expected, only a small enlargement is needed.

Since this modification is only technical, it is not described in further detail.

4.7.3 Gaining Further Depth Relations Using Logic

The up to here described method allows for gaining depth relations between segments that move relative to each other. However, there are situations where a depth order between two segments can be derived although they are static. Fig. 4.26 shows an example for that.

Assume that segment 3 is moving so that its depth relations with 1 and 2 have been established to "3 above 1" and "3 below 2". Now, the question is whether this is already all that one can get out of these observations. Regarding segment 3 moving between 1 and 2, it is obvious that 2 has to be above 1. Else, nothing could move between them and, at the same time, be in front of the one and behind the other. In mathematical terms, the depth relation is a *transitive relation*:

$$a \xrightarrow{\text{above}} b,\ b \xrightarrow{\text{above}} c \implies a \xrightarrow{\text{above}} c$$

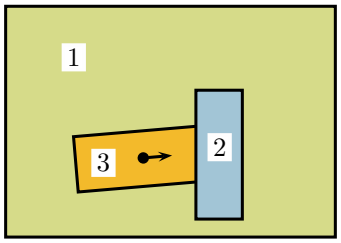


Figure 4.26: With the previously proposed method, the depth relation between 1 and 3, and 2 and 3 can be derived because 3 is moving. However, since there is no relative movement between 1 and 2, their depth relation stays unknown.

Deriving a relation as shown in the above example, is called a *chain inference*. Problems like that can be solved using logic frameworks. Given a set of depth relations obtained by the observation method described in the previous chapters, the goal is to extend them using logical conclusions.

Deriving the Absolute Depth Order

As a first step, a method called *constraint programming* is used to derive an absolute depth order of the segments, based on the relative depth relations (see Fig. 4.27). Constraint programming allows for solving a problem only by defining the properties of the desired solution. The necessary steps to obtain the solution do not have to be specified by the programmer. (In contrast, the very common *iterative programming* requires that the approach is already known.)

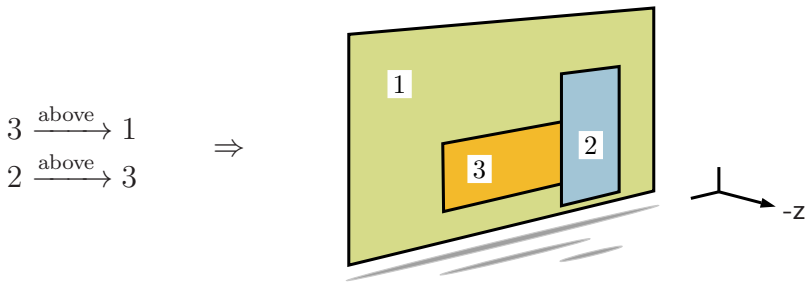


Figure 4.27: By the use of constraint programming, an absolute depth order is derived from the observed relative depth information. Note that, by definition, the depth index z increases in the direction towards the background.

The properties of the solution are described using *variables*, their possible values (*domain*) and *constraints*. Obtaining an absolute depth order of all segments

means finding a consistent set of depth indices for each segment. Consequently, given the segments $1, 2, ..., N$ in the frame, the variables $v_1, v_2, ..., v_N$ are created that represent these depth indices. The allowed range of the values is given by $1, 2, ..., N$ because each segment may need a unique depth index (this is not always the case). By definition, the smaller the number is, the closer the segment is to the foreground. Since the allowed values are a finite set, this is called a *finite domain*. (Constraint programming in general can also handle other types like *boolean* or *integer domains*.)

The most essential step is defining constraints. A solution for $v_1, v_2, ..., v_N$ has to fulfill the depth relations that have been observed. Consequently, the number of possible solutions decreases when adding further constraints. Finally, the solutions are computed using the constraint solver provided by the logic programming library[2].

constraints	**domain**	**solutions**
$3 \xrightarrow{\text{above}} 1 \;\widehat{=}\; v_3 < v_1$ $2 \xrightarrow{\text{above}} 3 \;\widehat{=}\; v_2 < v_3$	$v_1, v_2, v_3 \in \{1, 2, 3\}$	$\Rightarrow$ $v_1 = 3$ $v_3 = 2$ $v_2 = 1$

Figure 4.28: Deriving the absolute depth order of segments from depth relations which are reformulated as constraints for the desired solution. In this case, there is only one solution. Since v_2 has the minimum value, segment 2 has to be in the foreground, while, due to v_1 is maximal, 1 is in the background.

Fig. 4.28 shows the result of this derivation for the given example of Fig. 4.26. In that case there is only one possible solution. This, however, is not always the case, especially when many segments are present but only few relations/constraints are known. To show this, the example is slightly extended by two additional, static segments 4 and 5 (see Fig. 4.29).

Since segments 4 and 5 are static relatively to any other segment in their neighborhood, no depth relation can be observed. Consequently, no further constraints can be applied although the number of variables and allowed values has increased. As a consequence, numerous solutions are possible and, hence, returned by the constraint solver (see Fig. 4.30).

It can be seen that the given constraints are always fulfilled as expected, but the "free" variables can take any value. Using one of that solutions, e.g. (a), could lead to the wrong conclusion that 4 was behind 1. Or, using (c), "4 is above 3" could be derived, which is possible but not proved. Since only conclusions that

[2]logilab-constraint for Python available at Logilab.org

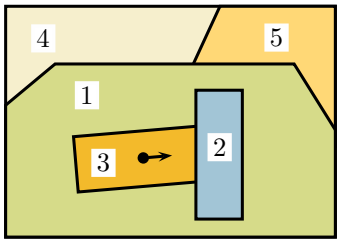


Figure 4.29: The example of Fig. 4.26 is extended by two more segments 4 and 5. However, since they do not move they do not offer additional constraints for the depth order.

constraints	**domain**		**solutions**			
			$v_5 = 5$	$v_5 = 3$	$v_5 = 2$	
			$v_4 = 4$	$v_4 = 3$	$v_4 = 2$	
$v_3 < v_1$	$v_1, ..., v_5 \in$	$\Rightarrow$	$v_1 = 3$	$v_1 = 3$	$v_1 = 4$	...
$v_2 < v_3$	$\{1, ..., 5\}$		$v_3 = 2$	$v_3 = 2$	$v_3 = 3$	
			$v_2 = 1$	$v_2 = 1$	$v_2 = 1$	
			(a)	(b)	(c)	

Figure 4.30: Since the number of variables increased while the number of constraints remained the same, more than one solution is found. Only some of them are listed (a-c).

are definitely true should be made, there is no point in including segments into the computation that do not appear in any constraint. For that reason, variables are only created for segments for which one or more depth relations have been found. By that, the number of allowed values is also decreased.

Deriving Depth Relations

In spite of taking only constrained segments into account, there is usually still more than one solution for their depth order. Fig. 4.31 shows a further extension of the example. Segment 6 might be the right part of the object corresponding to 3 but this fact is unimportant. However, since it moves, it adds two more constraints: "6 above 1" and "2 above 6".

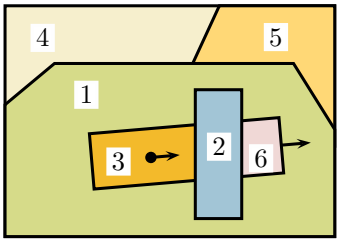


Figure 4.31: The example of Fig. 4.29 is extended by one more segment 6. Since it moves relative to 1 and 2, two more constraints are added, namely "6 above 1" and "2 above 6".

Fig. 4.32 shows the result. Note that only four variables are created since v_1, v_2, v_3, v_6 are the only ones that appear in the four constraints. Besides that, only values between 1 and 4 are needed to be allowed.

constraints	**domain**		**solutions**			
$v_3 < v_1$ $v_2 < v_3$ $v_6 < v_1$ $v_2 < v_6$	$v_1, v_2, v_3, v_6 \in \{1, ..., 4\}$	$\Rightarrow$	$v_1 = 3$ $v_6 = 2$ $v_3 = 2$ $v_2 = 1$	$v_1 = 4$ $v_6 = 3$ $v_3 = 2$ $v_2 = 1$	$v_1 = 4$ $v_6 = 2$ $v_3 = 3$ $v_2 = 1$	...
			(a)	(b)	(c)	

Figure 4.32: Constraints, domain and a selection of solutions (a-c) for the example in Fig. 4.31. See text for a detailed description.

Given one solution for the absolute depth order, the relative depth orders can easily be obtained by checking all pairs of segments/variables. Regarding solution (a), the new relation "2 above 1" can be derived correctly. Solution (b) however implies "3 above 6" in addition, while (c) indicates the opposite "6 above 3".

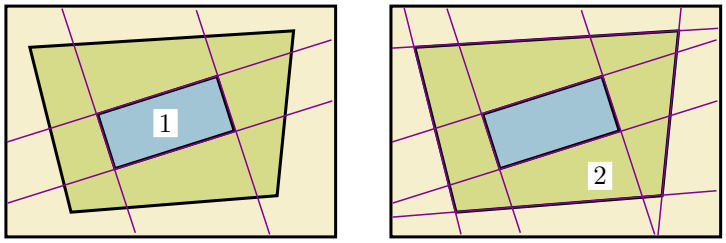
This shows that all possible solutions have to be taken into account in order to obtain reliable results. In particular, only relations that are never equal in any

solution are extracted. Since $v_3 = v_6$ in (a), no depth relation can be obtained for 3 and 6. Note that $v_a = v_b$ for two arbitrary segments a and b does not imply that they are on the same depth level. It implies that their depth relation cannot be derived from the available data.

4.8 Improvements of the Layer Method

4.8.1 Detecting Wrong Lines

As described in section 4.4, the boundary lines of each segment are obtained using the Hough transform. The changes of those lines between frames is used, next to optical flow, to compute a homography matrix. It was stated in section 4.3 that this improves the accuracy of this estimation process.



(a) Lines of segment 1 (b) Lines of segment 2

Figure 4.33: When finding the boundary lines of a segment (—), some of them might not be a real border of the corresponding object but of another object being in the foreground.

However, results can be very bad, if lines are found that do not really belong to the segment but to an occluder. This is illustrated in Fig. 4.33. The foreground object represented by segment 1 has 4 lines that really belong to the segment. Segment 2 however has 8 lines but only 4 of them belong to the object. If segment 1 was moving and the transformation matrix for segment 2 was computed using all 8 lines, the result would be distorted.

If segment 2 was the foreground object with a rectangular hole, no correct lines would be available for segment 1. By computing the transformation matrix regardless of that fact, wrong results would be obtained.

It is obvious that lines cannot be used to track the transformation of segments unless a way to ignore wrong lines is found. The proposed framework has up to five mechanisms to deal with this problem:

1. **Occlusion of Segments.** The depth order computed as described in section 4.7 is used to sort out lines that adjoin to higher segments. This segment is likely to be an occluder causing the wrong line. The advantage of this method is that it takes the real cause of the problem into account: the depth. However, a big drawback is that the segments already have to have moved in order to obtain their depth relation. Hence, this is a *chicken or the egg dilemma* in the first frame and cannot be used without other methods like those described below.

2. **3D Data.** If depth information obtained by e.g. stereo vision is available, this data can be used in the previous method instead of 4.7 which solves the mentioned problems.

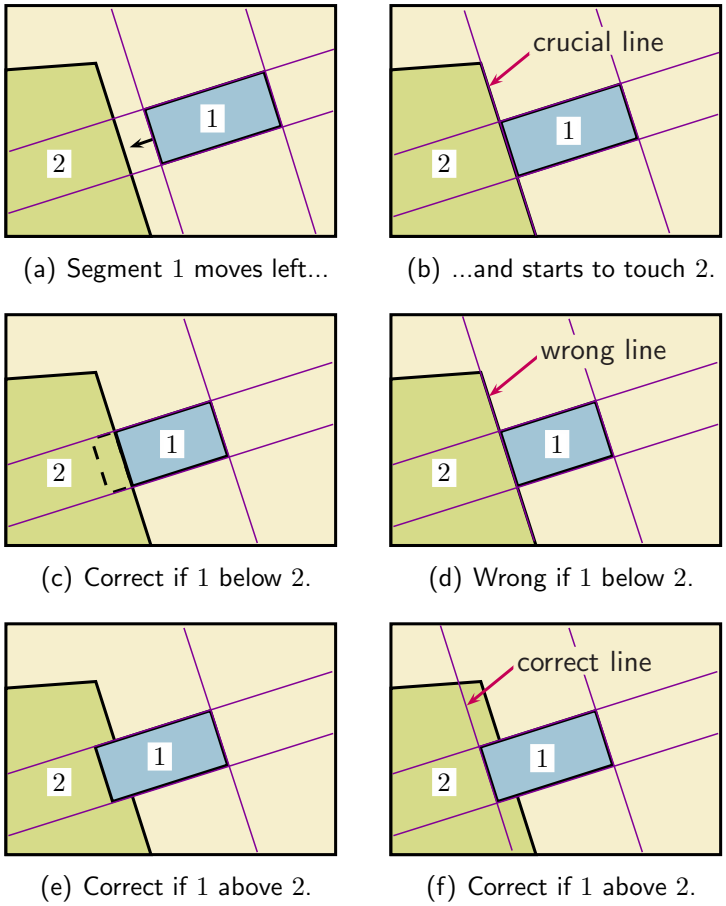
3. **Touching of Segments.** The problem described in method 1 is essential in the case shown in Fig. 4.34. One segment moves towards another segment (a) and starts to touch it (b). The four lines of segment 1 are sufficient to compute its transformation. Until here, no depth order using method 1 could be obtained since the segments did not occlude each other yet. For that reason, using the line at the connection for the further transformation of segment 1 is crucial, since it may belong to segment 2.

   Assume that 1 is below 2. Then, the correct transformation of layer 1 would be as shown in (c). The layer extends into segment 2 but is invisible (visualized with a dashed line). This can only be achieved by ignoring the line a the connection. The transformation is then computed using the remaining three lines and optical flow. If the line is not ignored (d), layer 1 would be compressed by the computed transformation, instead of moved towards 2.

   If 1 is above 2, the transformation of 1 is always correct, no matter if the line is ignored (e) or not (f). Since lines in general provide better results, it is even better not to ignore it. However, since the proposed framework needs to perform best in both situations, it will always ignore the line between to segments that just started to touch. That way, the transformation is computed correctly although the depth order is unknown.

   In the next frame, method 1 will be able to find out the depth order since one of the segment is occluding the other.

4. **Convex Hull.** If no depth relation is available for two segments at all, it is assumed that segments lying within another segment correspond to foreground objects. Regarding Fig. 4.33, the system will guess that segment 1 is above 2, as long as no confident information using the first method



(a) Segment 1 moves left... (b) ...and starts to touch 2.

(c) Correct if 1 below 2. (d) Wrong if 1 below 2.

(e) Correct if 1 above 2. (f) Correct if 1 above 2.

Figure 4.34: (a) The transformation of segment 1 is computed using its boundary lines (—). (b) When segment 1 starts to touch 2, using the line at the connection can lead to a wrong transformation if 1 is below 2, see (c) and (d). For that reason, the line is ignored, see (e) and (f). See text "Touching of Segments" for details.

could be gained. Note that the assumption is wrong if segment 2 has a hole so that 1 is visible although it is below 1.

Whether this assumption can be made or not depends on the application of the system. If many objects with holes are present, this should be avoided. However, since the scenarios considered in this work are rather simple, this fall-back system performs well.

In principe, the algorithm computes the convex hull of the segment and ignores all lines lying within. By that, only the outer lines that belong most

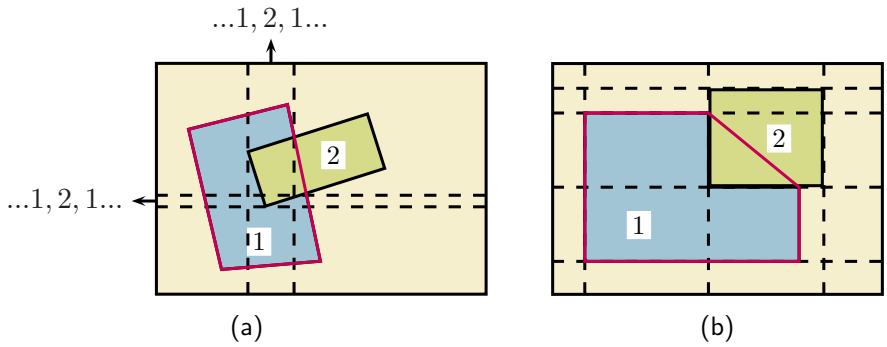


Figure 4.35: The convex hull (—) of a segment (in this case 1) can be used to figure out if another segment (here 2) extends into it. Instead of computing the convex hull to check this, the segment sequences (see 3.3) are used, which gives in most cases equivalent results (a). However, in some special cases, the results are not equivalent (b): Two edges of segment 2 lie within the convex hull of 1, but no sequence $(..., 1, ..., 2, ...1, ...)$ can be found.

probably to the corresponding object are used. In order to avoid the extra cost of computing the convex hull, the counter $\mathbf{C}^{\mathrm{o}}_{i,j}$ that is created during the computation of the segment's relations as described in section 3.3 is (re)used. If its value for a pair of segments i, j is greater than 0, a segment sequence $(...j, ...i, ..., j, ...)$ seems to exist. In this case, j is inside of the convex hull of i. This is the case if i, j are *Overlapping* but also if j only extends into i as in Fig. 4.35(a). Note that this procedure is not completely equivalent to computing the convex hull since the segment sequences are only computed along the x and y axis. As a consequence, cases like in Fig. 4.35(b) are not recognized as segment is 2 extending into 1. However, one might even prefer this behavior. Furthermore, since the method, contrary to the first two, is only used to guess if a line is wrong most probably, the result is not expected to be always right.

5. **RANSAC.** If a wrong line could not be recognized by any of the previous methods, the last instance in the framework that might prevent a bad homography matrix is RANSAC which is described in section 4.3.5. A wrong line can be seen as an outlier since is does not fit to the other line's movement or the optical flow data.

With the first four methods it is determined which boundary lines should be ignored. However, this computation does not require that the lines itself have already been found by the Hough transform. As described in section 4.4 the first

step of the Hough transform is computing the first derivate of the segment in order to find its border pixels. Then, for each border pixel, votes are created.

If one of the above methods found out that the line between segment i and j has to be ignored, the corresponding border pixels can be deleted from the border pixel map. As a consequence, no votes will be created for that pixels and the Hough transform will not find the lines in the first place. This saves computational time since no lines that would get deleted later anyway need to be computed. On top of that, there is another advantage concerning method 3.

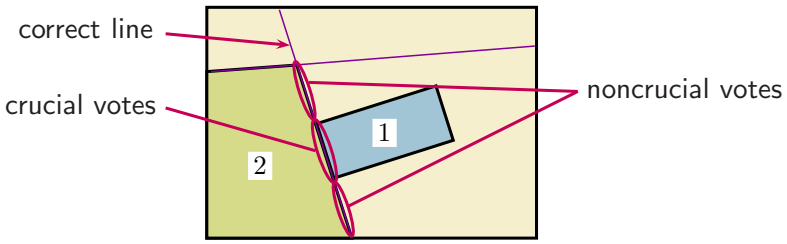


Figure 4.36: Not crucial *lines* but crucial *votes* are deleted. This prevents a correct line to be deleted if only parts of it are crucial.

Note that there is no conceptual difference between segment 1 and 3 in Fig. 4.34. As a consequence, the crucial line would be ignored when computing the transformation of segment 2. However, this line remains correct no matter if 2 is below or above 1 since many pixels of that line stay visible. Hence, deleting the line, based on the fact that some parts of it touched another segment, would ignore many correct pixels.

Since not the lines are deleted but the Hough votes as described above, fortunately, this problem does not occur. Only the votes in the region where segment 2 is touched by 1 are deleted (see Fig. 4.36). As there remain many other uncritical votes, the putatively crucial line is correct.

4.8.2 Preventing Error Propagation

A big drawback of the update process described in section 4.2 is that errors are propagated. If the computation of $\mathbf{P}_n$ is very imprecise because e.g. the optical flow data is poor, this does not only lead to an erroneous $\mathbf{M}_n$ but also to errors in $\mathbf{M}_{n+1}, \mathbf{M}_{n+2}$ etc. This can be prevented by using a different update process as described now.

As described in section 4.3.2, the computation of $\mathbf{P}_n$ can be done using both points and lines. The disadvantage of point correspondences derived from optical flow is that optical flow is only available for subsequent frames. For that reason, the update of the mapping can only be done successively.

When using lines on the contrary, a transformation between any frames can be computed which means that erroneous frames can be skipped. The only requirement is that, for each line in one frame, the corresponding line in the other frame needs to be known.

Fig. 4.37 shows the whole process. For the first two frames, there is no difference compared to the original procedure (see Fig. 4.5 for comparison): In the initial frame ($n = 0$), four lines were found and the transformation to the next frame is computed just like before, but note that the process relies completely on line correspondences and no optical flow is used here.

However, when processing the next frame ($n = 2$), the algorithm checks whether the same lines could be found as in the previous frame. If yes, then the transformation is not computed refering to the previous frame what would give $\mathbf{P}_2$. Instead, the initial frame is considered and the matrix $\mathbf{P}_{0,2}$ is obtained. It is obvious that frame 1 does not affect that computation, especially no error is propagated. For every following frame that contains the same lines as the initial frame, the transformation is computed that way. Note that the lines are still tracked concerning all frames in between in order to guarantee a consistent line labeling. Finding corresponding lines between distant frames would lead to mismatchings.

The described algorithm only works as long as lines are constant over a long period of time. Luckily, this can be assumed in most cases since the shapes of rigid objects usually are stable. Another requirement is that at least four lines can be found, thus line correspondences are sufficient for the computation and optical flow can be completely ignored. If this is not possible e.g. because the object has less or no edges, the framework has to "fall back" to the original method. Besides that, as soon as at least four stable lines arise again, the method should be switched again (see Fig. 4.38).

The algorithm, including this switching-feature, is now described in a more general way. In the normal case the transformation between two subsequent frames is computed as described in section 4.2 using points and lines. This mode will now be called *successive mode*. The mapping of the layer to the current frame n is then given by

$$\mathbf{M}_n = \mathbf{P}_n \cdot \mathbf{M}_{n-1} \ .$$

If at least 4 lines in frame n match to lines in the previous frame, frame $s := n - 1$ is then marked as the start of a new sequence. $\mathbf{M}_s$ is the mapping of the layer to frame s. Both s and $\mathbf{M}_s$ need to be stored when a new sequence is entered. The algorithm now switches to the *skipping mode*.

For each subsequent frame it needs to be checked whether all lines could still be established. If this is the case, the matrix $\mathbf{P}_{s,n}$ is computed. The mapping of

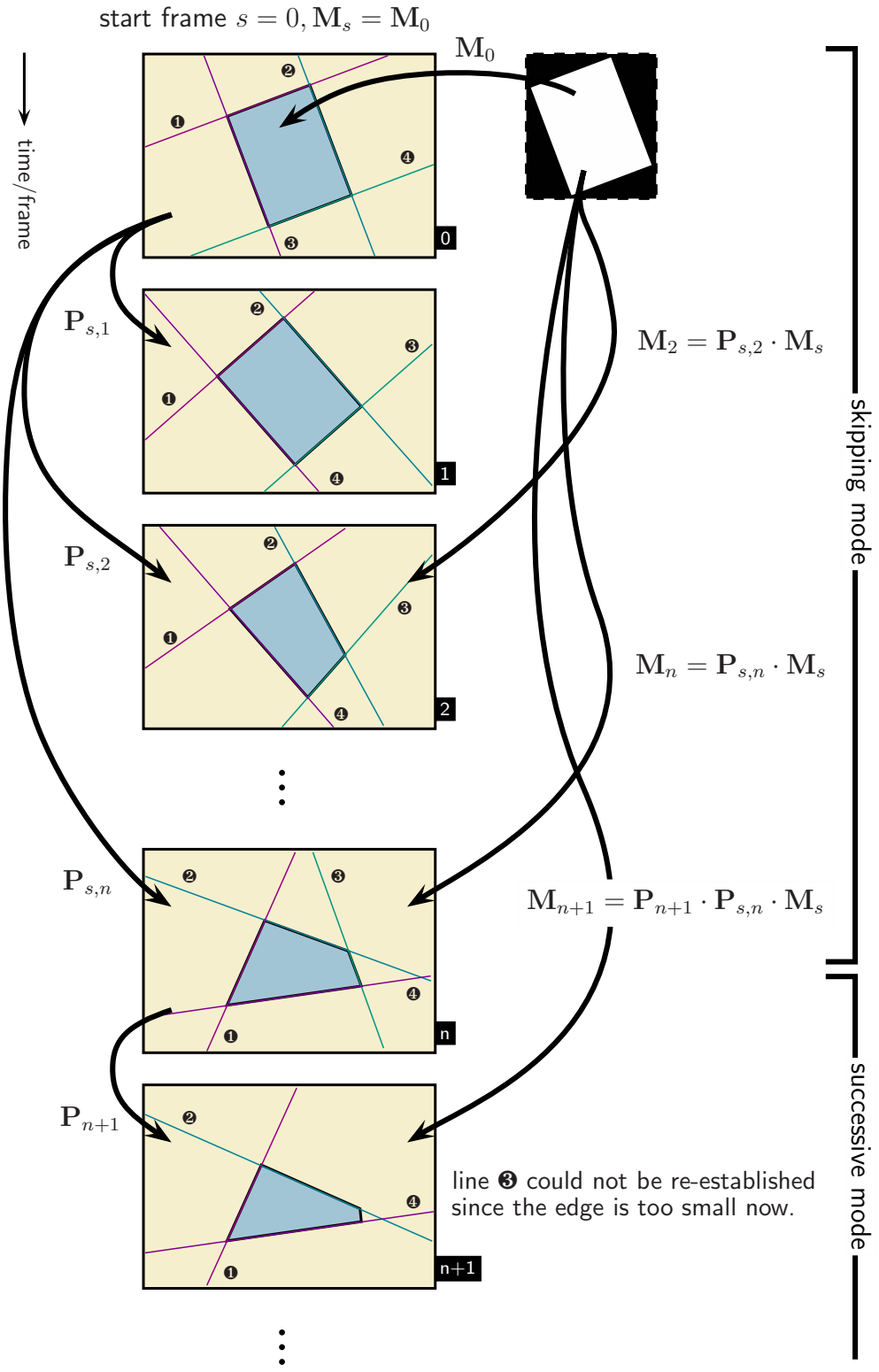


Figure 4.37: Extension of the mapping update procedure (see text).

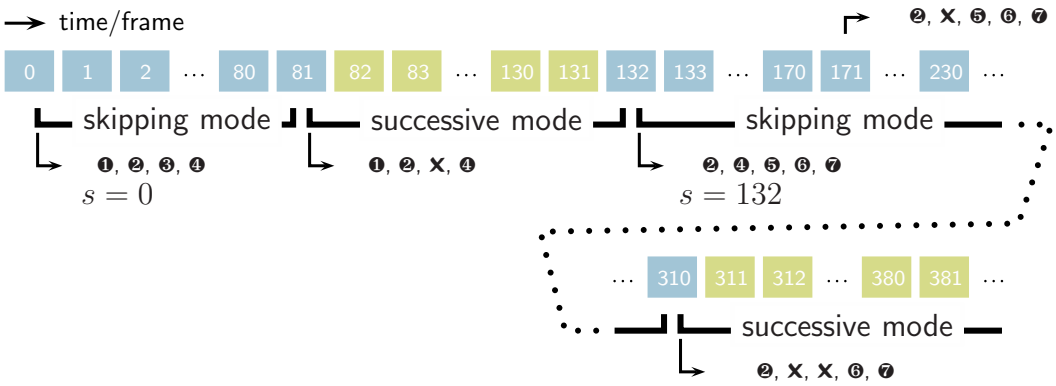


Figure 4.38: Example of how *skipping* and *successive mode* alternate depending on the appearance/disappearance of lines ❶ ... ❼.

the layer to the frame is then given by

$$\mathbf{M}_n = \mathbf{P}_{s,n} \cdot \mathbf{M}_s \ .$$

This is obvious because, by that, the layer gets at first transformed so that it matches frame s and after that the transformation between s and n is applied.

If one line could not be found in the current frame and the total number of line correspondences shrinks to less than 4, there is not enough data to compute a matrix $\mathbf{P}_{s,n}$. From now on, optical flow data is necessary and the framework switches to *successive mode*. Note that, as long as the number of available line correspondences is greater or equal to 4, *skipping mode* can be continued, since there is still a sufficient amount of data.

The framework proceeds in *successive mode* as long as the start of a new sequence could be found as described above.

4.8.3 RANSAC for Skipping Mode

The method proposed in section 4.8.2 exclusively relies on line correspondences. However, in section 4.8.1 has been discussed that lines can sometimes lead to very bad results if one line does not really belong to the segment. Then, the movement of another segment causes distortion of the homography matrix.

For that reason, some methods have been proposed that recognize or guess those wrong lines. The RANSAC algorithm is one of them. It treats those wrong lines as outliers that can be recognized because there are most often enough correct point and line correspondences.

Since the skipping mode only uses lines, the ratio of correct and wrong lines can become crucial. In the worst case there might be more outliers than inliers

and RANSAC would sort out values that would have led to the right result. For that reason, the standard RANSAC cannot be used in skipping mode. With the now proposed algorithm, the optical flow data can be included to the computation of the consensus set so that far more data is available.

In skipping mode, the homography matrix $\mathbf{P}_{s,n}$ between start frame s and current frame n is computed. It is obvious that optical flow data cannot be included in this computation since it only contains correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$ between $n-1$ and n but not between s and n. Thus, only homography matrices $\mathbf{P}_{n-1,n} \equiv \mathbf{P}_n$ between adjacent frames can be computed that way.

However, this matrix includes the transformation between $\mathbf{P}_{s,n}$ and the matrix $\mathbf{P}_{s,n-1}$ which was computed in the previous time step:

$$\mathbf{P}_{s,n} = \mathbf{P}_{n-1,n} \cdot \mathbf{P}_{s,n-1} \tag{4.20}$$

It is not the goal to compute $\mathbf{P}_{s,n}$ by using data of the previous frame because this would again lead to error propagation that is intended to be prevented. Instead of that, the optical flow data can be used to check the accuracy of $\mathbf{P}_{s,n}$. Given (4.20) and $\mathbf{x}' = \mathbf{P}_{n-1,n}\,\mathbf{x}$, it is obvious that each point correspondence from $n-1$ to n should fulfill the equation

$$\mathbf{x}' = (\mathbf{P}_{s,n} \cdot \mathbf{P}_{s,n-1}^{-1})\,\mathbf{x}\ . \tag{4.21}$$

With that result, RANSAC can be modified the following way: The sample set is selected from lines only. In order to compute the support of the temporary matrix $\check{\mathbf{P}}_{s,n}$, the distances of all available lines is computed using (4.11). For points, a new distance obtained using (4.21) is used:

$$d(\mathbf{x}_i', (\mathbf{P}_{s,n-1} \cdot \check{\mathbf{P}}_{s,n})\,\mathbf{x}_i) + d(\mathbf{x}_i, (\mathbf{P}_{s,n-1} \cdot \check{\mathbf{P}}_{s,n})^{-1}\,\mathbf{x}_i') \tag{4.22}$$

After finding the sample with the highest support (including the point correspondences), the final homography matrix is computed using only the inlying lines. Note that the inlying point correspondences are not included to this computation. If the number of inlying lines is less than 4, the skipping mode is aborted.

The whole procedure can be summarized by the following steps:

1. Randomly select 4 line correspondences as sample.
2. Compute out of the sample the homography matrix $\check{\mathbf{P}}_{s,n}$ using (4.7).
3. Compute the distance of each line correspondence (between frame s and n) using (4.11) and the distance of each point correspondence (between frame $n-1$ and n) using 4.22. Count the number of point correspondences c_{points} having a distance smaller than the threshold t_{points}. Also count the number of line correspondences c_{lines} with a smaller distance than t_{lines}. The support is then given by $c_{\text{points}} + 200 \cdot c_{\text{lines}}$.

4. Repeat the above steps several times.

5. Use only the lines in largest consensus set to compute the final homography matrix $\mathbf{P}_{s,n}$. If less than 4 lines are available, skip to successive mode.

With the proposed steps, wrong lines are recognized in skipping mode by taking the optical flow data into account.

4.8.4 Changing the Layer's Perspective

By now, the framework is capable of adding and removing pixels. Hence, the layers are not static anymore but are getting updated permanently. The only thing that stays unchanged is the perspective of the layer. This is illustrated in Fig. 4.39.

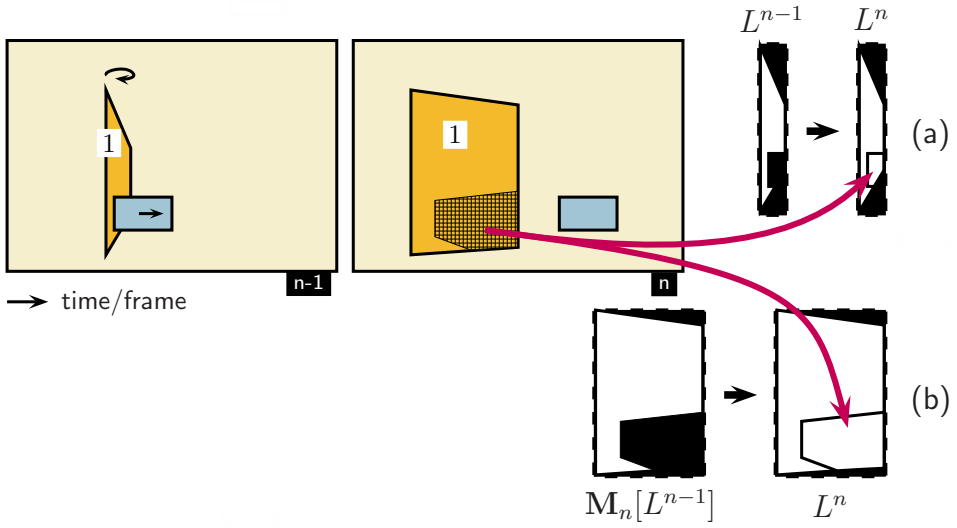


Figure 4.39: New data of segment 1 is mapped to the layers perspective (a). Since the perspective of the object related to segment 1 improves, the layer's perspective is updated (b) so that new data is stored with the best accuracy.

In the initial frame, object 1 is visible edgewise. According to that, the segment is very narrow and so is the now created layer. Since the object rotates, the camera gets a better view of the object. However, when adding new parts of the segment to the layer, the segment is transformed back to the original perspective of the initial frame (see 4.39(a)). It is obvious that this procedure can lead to inaccuracy.

In order to improve the method, layers should always store their data from the best perspective. This can easily be achieved by checking

$$|\mathbf{M}_n[L^{n-1}]| \ ,$$

which is the number of pixels of the layer after transformation to the current frame (see 4.5). If it is greater than the layer's size $|L^{n-1}|$, the current frame's perspective seems to be better. In this case, the layer data is replaced by this transformed data, which is denoted as $\mathbf{M}_n[L^{n-1}]$ in Fig. 4.39(b). Note that this notation includes the cropping of the layer to the minimum size. According to that, the mapping M_n is reset to

$$\mathbf{M}_n^* = \begin{bmatrix} 1 & 0 & t_\mathrm{x} \\ 0 & 1 & t_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.23}$$

where t_x and t_y define the new offset of the layer, similar to the initialization of layers that was described in section 4.1. Note, that the described procedure is not an ordinary re-initialization of the layer. The difference is that no invisible pixels of the layer will get lost. After this change of perspective, the new data of the frame is added.

4.9 Summary of the Standard Layer Framework

The (standard) layer framework which has been described until this point consists of several modules (see Fig. 4.40). The main module can be summarized as "Layer Management" containing the initialization of new layers, the computation of their transformation matrices (homographies) between frames, and the updating of the layer data, which covers adding and removing pixels.

These procedures require data which is provided by external modules for Image Segmentation and Optical Flow and internal modules which compute the boundary lines of segments and their depth relations. It is important to note that some modules can be interchanged by other methods, or new modules can be added to improve the system's performance. This section describes some of these possibilities.

The boundary lines and the optical flow values are used in combination to compute the homography matrices. However, the homography estimation also works with one of the both methods, or could be improved by implementing a third method using e.g. a feature point detection. The tracking of feature points would allow the "skipping mode" also to work with non-angular segments (see section 7.2), while the current implementation only works when lines are available.

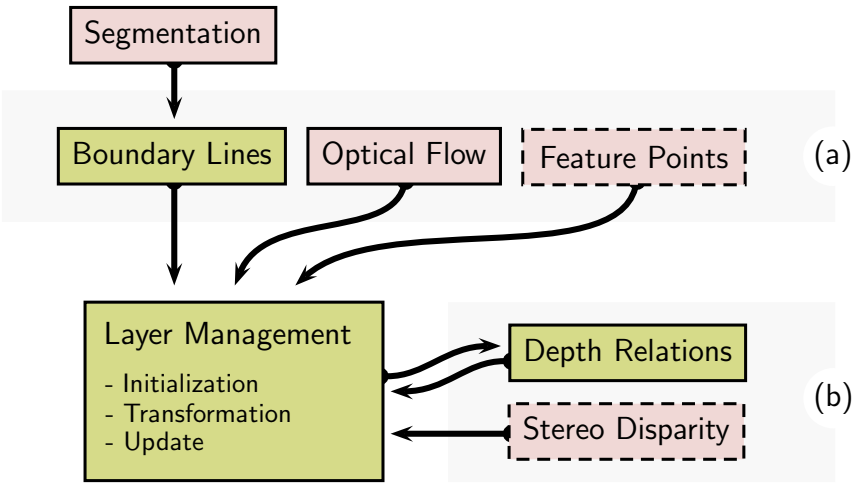


Figure 4.40: Block diagram of the standard layer framework. The transformations of layers are computed using the data produced by (a). The updating of the layers requires depth relations obtained by (b). Colors: external algorithms, layer framework. Dashed lines indicate optional and future modules.

The computation of the depth relations can also be replaced. Since the current work is intended to be based on monocular images, the derivation of depth order is computed by analyzing the segment's movements. Instead of this, a module using *Stereo Vision*[3] can be added in order to provide a depth value for each pixel. This way, not only depth relations but real distances between segments can be obtained. This was used in Aksoy et al. [3] to remove *Touching* and *Overlapping* relations for segment pairs which are not connected in the 3d domain.

It is easy to see that the proposed framework is highly modular and can be reduced or extended according to the needs implied by the application. Nevertheless, the next section covers a more fundamental extension of the system.

[3]Stereo Vision derives the depth of an image point by establishing its position in the image from a second camera. The pixel shift (*disparity*) has an inverse relation to the distance.

4.10 Estimating Errors of Layers

In the previous chapters, a framework has been proposed that stores each image segment in its own layer while tracking its transformation during the image sequence. By that, parts of segments that get occluded by other segments are preserved so that segment permanence can be established (Fig. 4.41). The tracking of the transformations is done by evaluating point correspondences (obtained by optical flow) and line correspondences (obtained by the Hough transform).

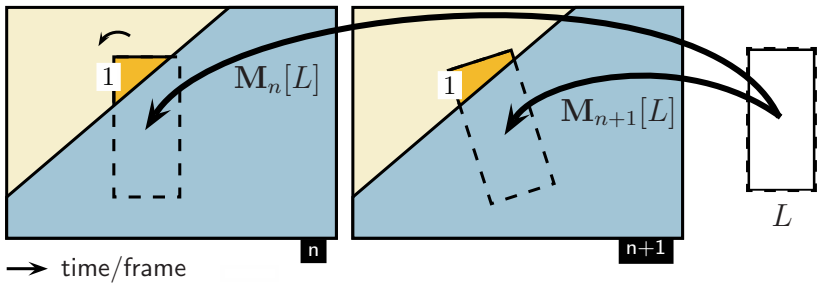


Figure 4.41: The transformation matrix $\mathbf{M}_n$ is used to map the layer L to the n-th frame. Its computation can obviously use only the visible data, which may lead to inaccuracies.

An important fact is that the transformation of a layer is obtained by evaluating only the appearance of the corresponding segment. In other words, since a segment can be seen as the currently visible part of its layer, it is an important question how precise the segment's transformation is for the rest of the layer. For that reason, the framework is extended so that it computes the expected deviation based on the data that was used to obtain the transformation matrix.

4.10.1 Covariance of the Homography Matrix

A detailed description of the theory behind the error estimation of a homography matrix can be found in [17]. For that reason, the focus of this chapter is on how the given results are used to extend the proposed framework. However, a brief description of the mathematics is given here as well.

An estimation problem like obtaining a homography from measured point correspondences can be considered in a more general way as follows. One defines a "parameter space" $\mathbb{R}^M$ and a "measurement space" $\mathbb{R}^N$. A function $f : \mathbb{R}^M \rightarrow \mathbb{R}^N$ maps a parameter vector to the corresponding measurement vector. This defines a subspace S_{M} with a dimension equal to the number of

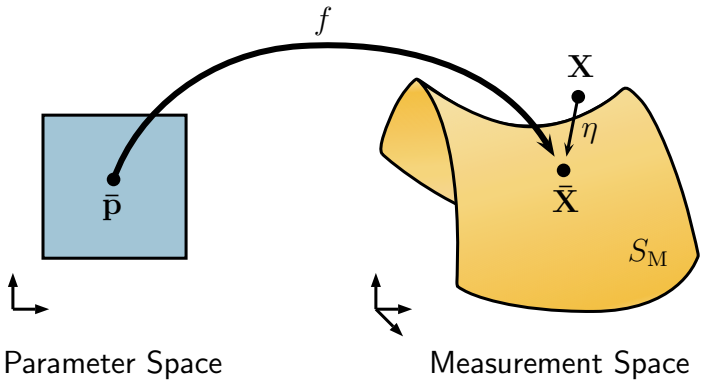


Figure 4.42: Function f maps the parameter vector $\bar{\mathbf{p}}$ to the (higher dimensional) measurement vector $\bar{\mathbf{X}}$. By varying $\bar{\mathbf{p}}$, $\bar{\mathbf{X}}$ moves in the subspace S_{M} and its dimension is equal to $\bar{\mathbf{p}}$'s degrees of freedom. A measurement with noise $\mathbf{X}$ does not lie on S_{M} but is mapped by a function η to the closest point on it.

degrees of freedom of the parameters. Fig. 4.42 shows a schematic for this with $M = 2$, $N = 3$ and a 2-dimensional subspace S_{M}. The dimension of S_{M} is not always equal to M, which will be important later.

On S_{M}, f is invertible so that the corresponding parameter vector $\bar{\mathbf{p}}$ of a measurement $\bar{\mathbf{X}} \in S_{\mathrm{M}}$ can be obtained by $\bar{\mathbf{p}} = f^{-1}(\bar{\mathbf{X}})$. The overbar indicates that the entities are noise-free. However, a measurement does not lie on S_{M} in most cases since noise is present. For that reason, the function $\eta : \mathbb{R}^N \rightarrow S_{\mathrm{M}}$ is defined that maps a measurement vector to the closest point $\tilde{\mathbf{X}}$ on S_{M}. Finding this mapping is the essential part of the estimation process. Given an optimal estimator, this estimated vector is equal to the true values ($\tilde{\mathbf{X}} = \bar{\mathbf{X}}$), while in realistic implementations the distance $d(\tilde{\mathbf{X}} - \bar{\mathbf{X}})$ is called the *estimation error*. This error usually decreases when including more measurements, thus expanding the measurement vector, and the estimated parameter $\tilde{\mathbf{p}}$ comes closer to the correct value $\bar{\mathbf{p}}$.

In the case of homography estimation, the parameter vector has dimension 9 and contains the entries of the transformation matrix $\mathbf{P}$. It is denoted as $\mathbf{p} = {}^{\mathrm{t}}(p_{11}, p_{12}, ..., p_{33})$ according to section 4.3. The estimated parameter vector $\tilde{\mathbf{p}}$ is obtained using n point correspondences (note section 4.10.4 about the use of line correspondences) that are gained from optical flow. Since optical flow finds for each pixel $\mathbf{x}_i$ the best matching correspondence $\mathbf{x}'_i$ in the next image, only the $\mathbf{x}'_i$ are treated as measurements subject to a Gaussian error distribution with variances σ_{x}^2 and σ_{y}^2, while the $\mathbf{x}_i = \bar{\mathbf{x}}_i$ are considered as being fixed. Consequently, the measurement vector is built by stacking the coordinates of

each target point: $\mathbf{X}' = {}^{\mathrm{t}}(x'_0, y'_0, x'_1, y'_1, ..., x'_n, y'_n)$, and its dimension is $N = 2n$.

The accuracy of the estimated parameter vector depends on the number of points used for the computation, the precision of the given point correspondences, and the configuration of the points. It is conveniently captured in the *covariance matrix* $\Sigma_{\mathbf{p}}$ of the transformation. Since $\mathbf{P}$ has 9 entries, $\Sigma_{\mathbf{p}}$ is a 9×9 matrix and its diagonal elements are the variances of the corresponding elements of $\mathbf{P}$.

The goal is to obtain the covariance of the estimated $\tilde{\mathbf{p}}$ from the covariance of the measurement vector $\mathbf{X}'$. For that purpose, the following three theorems are needed (cited from [17]):

Theorem 1: Forward propagation of covariance. *Let $\mathbf{v}$ be a random vector in $\mathbb{R}^M$ with mean $\bar{\mathbf{v}}$ and covariance matrix $\Sigma_{\mathbf{v}}$, and let $f : \mathbb{R}^M \to \mathbb{R}^N$ be differentiable in a neighborhood of $\bar{\mathbf{v}}$. Then up to a first-order approximation, $f(\mathbf{v})$ is a random variable with mean $f(\bar{\mathbf{v}})$ and covariance*

$$\Sigma_f = \mathbf{J}_f \, \Sigma_{\mathbf{v}} \, {}^{\mathrm{t}}\mathbf{J}_f \tag{4.24}$$

with the Jacobian matrix of f, evaluated at $\bar{\mathbf{v}}$.

The Jacobian matrix includes the partial derivates of a vector-valued function $\mathbf{y} = f(\mathbf{p})$ with respect to a parameter vector $\mathbf{p}$. It has the form

$$\mathbf{J}_f = \mathbf{J}_f(\mathbf{p}) = \begin{bmatrix} \frac{\partial y_1}{\partial p_1} & \cdots & \frac{\partial y_1}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial p_1} & \cdots & \frac{\partial y_m}{\partial p_n} \end{bmatrix}$$

and may also be denoted as $\frac{\partial \mathbf{y}}{\partial \mathbf{p}}$.

Theorem 2: Backward propagation of covariance. *Let $f : \mathbb{R}^M \to \mathbb{R}^N$ be a differentiable mapping and let $\mathbf{J}_f$ be its Jacobian matrix evaluated at a point $\bar{\mathbf{p}}$. Suppose that $\mathbf{J}_f$ has rank M. Then f is one-to-one in a neighborhood of $\bar{\mathbf{p}}$. Let $\mathbf{X}$ be a random variable in $\mathbb{R}^N$ with mean $\bar{\mathbf{X}} = f(\bar{\mathbf{p}})$ and covariance matrix $\Sigma_{\mathbf{X}}$. Let $f^{-1} \circ \eta : \mathbb{R}^N \to \mathbb{R}^M$ be the mapping that maps a measurement $\mathbf{X}$ to the set of parameters corresponding to the maximum likelihood estimate $\tilde{\mathbf{X}}$. Then to first-order, $\tilde{\mathbf{p}} = (f^{-1} \circ \eta)(\mathbf{X})$ is a random variable with mean $\bar{\mathbf{p}}$ and covariance matrix*

$$\Sigma_{\mathbf{p}} = ({}^{\mathrm{t}}\mathbf{J}_f \Sigma_{\mathbf{X}}^{-1} \mathbf{J}_f)^{-1}.$$

In the context of homographies arises the problem that the set of parameters is redundant. Although the matrix has 9 entries, it has only 8 degrees of freedom. Such case is called *over-parametrized*. In particular, the matrix $k\mathbf{P}$ represents the

same map for any k. For that reason, the mapping f is not one-to-one (injective) in the neighborhood of $\bar{\mathbf{p}}$. As a consequence, $\mathbf{J}_f$ has rank $d < M$ where d is the number of *essential parameters*, thus 8. In that case, the inverse $({}^{\mathrm{t}}\mathbf{J}_f \Sigma_{\mathbf{X}}^{-1} \mathbf{J}_f)^{-1}$ cannot be computed since full rank would be required.

Furthermore, since the homography can be multiplied by an arbitrary constant k without changing the mapping, the entries of the matrix can vary without bound and hence have infinite variance. For that reason, the estimated parameter set has to be restricted to lie on an 8-dimensional surface S_{P}. This is usually done by demanding $\|\mathbf{p}\| = 1$ with the *Frobenius norm*

$$\|\mathbf{p}\| = \sqrt{\sum_{i,j} p_{i,j}^2} \ .$$

With that restriction, the parameter vectors lie on a unit sphere and every tangential plane is perpendicular to the parameter vector. Furthermore, the function f is invariant to changes of the scale: $f(\mathbf{p}) = f(k\mathbf{p})$. Consequently, its Jacobian matrix $\mathbf{J}_f$ has a null-vector in the radial direction. It can be found that in cases like this, where the parameter vector is constrained to a surface locally orthogonal to the nullspace of $\mathbf{J}_f$, the inversion can be replaced by the pseudo-inversion. This is the content of the following theorem (cited from [17]):

Theorem 3: Backward propagation of covariance - over-parametrized case. *Let $f : \mathbb{R}^M \to \mathbb{R}^N$ be a differentiable mapping taking $\bar{\mathbf{p}}$ to $\bar{\mathbf{X}}$ and let $\mathbf{J}_f$ be its Jacobian matrix. Let a Gaussian distribution on $\mathbb{R}^M$ be defined at $\bar{\mathbf{X}}$ with covariance matrix $\Sigma_{\mathbf{X}}$ and let $\tilde{\mathbf{p}} = (f^{-1} \circ \eta)(\mathbf{X})$ be the mapping taking a measurement $\mathbf{X}$ to the maximum likelihood estimate parameter vector $\mathbf{p}$ constrained to lie on a surface S_{P} orthogonal to the null-space of $\mathbf{J}$. Then $f^{-1} \circ \eta$ induces a distribution on $\mathbb{R}^M$ with covariance matrix, to first-order equal to*

$$\Sigma_{\mathbf{p}} = ({}^{\mathrm{t}}\mathbf{J}_f \Sigma_{\mathbf{X}}^{-1} \mathbf{J}_f)^{+}. \tag{4.25}$$

The last theorem can now be used to compute the covariance of the homography matrix. Firstly, the Jacobian matrix $\mathbf{J} = \partial \mathbf{X}' \,/\, \partial \mathbf{p}$ has to be considered. As $\mathbf{X}'$ is a composed vector of the image points $\mathbf{x}_i'$ in the second image, its Jacobian matrix has the form

$$\mathbf{J}_{\mathbf{X}'} = \begin{bmatrix} {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{x}}_1'} \\ {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{x}}_2'} \\ \vdots \\ {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{x}}_n'} \end{bmatrix} .$$

with the Jacobians $\mathbf{J}_{\hat{\mathbf{x}}'_i}$ of each point with size 2×9. The dependence of $\mathbf{x}'$ of $\mathbf{p}$ is given by $\mathbf{x}' = \mathbf{P}\mathbf{x}$. Note that this equation is in homogeneous coordinates and needs to be transformed back to euclidean coordinates so that the third component of $\hat{\mathbf{x}}'$ equals 1. Consequently, the Jacobian is obtained by computing the derivate of $\mathbf{P}\mathbf{x}/[\mathbf{P}\mathbf{x}]_3$ with respect to all entries of $\mathbf{P}$. The result is

$$\mathbf{J}_i := \mathbf{J}_{\hat{\mathbf{x}}'_i}(\mathbf{p}) = \partial \hat{\mathbf{x}}'_i \,/\, \partial \mathbf{p} = \frac{1}{w'_i} \begin{pmatrix} {}^{\mathrm{t}}\mathbf{x}_i & {}^{\mathrm{t}}\mathbf{0} & -\hat{x}'_i\,{}^{\mathrm{t}}\mathbf{x}_i \\ {}^{\mathrm{t}}\mathbf{0} & {}^{\mathrm{t}}\mathbf{x}_i & -\hat{y}'_i\,{}^{\mathrm{t}}\mathbf{x}_i \end{pmatrix}, \tag{4.26}$$

where $\mathbf{x}' = \mathbf{P}\mathbf{x} = {}^{\mathrm{t}}(x', y', w')$ and $\hat{\mathbf{x}}' = {}^{\mathrm{t}}(\hat{x}', \hat{y}') = {}^{\mathrm{t}}(x'/w', y'/w')$.

The covariance matrix $\Sigma_{\mathbf{X}'}$ also has a decomposition. Since the image vectors $\mathbf{x}'_i$ are measured independently, is has the form

$$\Sigma_{\mathbf{X}'} = \begin{bmatrix} \Sigma_{\hat{\mathbf{x}}'_1} & \cdots & 0 \\ & \Sigma_{\hat{\mathbf{x}}'_2} & \\ \vdots & & \vdots \\ & & \ddots \\ 0 & \cdots & \Sigma_{\hat{\mathbf{x}}'_n} \end{bmatrix},$$

where $\Sigma_{\hat{\mathbf{x}}'_i}$ is the 2×2 covariance matrix of the i-th point. If the measurements of the x- and y-component are independent, this is a diagonal matrix and the diagonal elements are set to the expected variances σ_{x}^2 and σ_{y}^2. Given $\Sigma_{\mathbf{X}'}$ and $\mathbf{J}_{\mathbf{X}'}$, the computation of the homography's covariance is given by (4.25). However, due to their decompositions, the computation can be expressed using a sum instead of building the large matrices, which is of advantage when dealing with a large number of measurements:

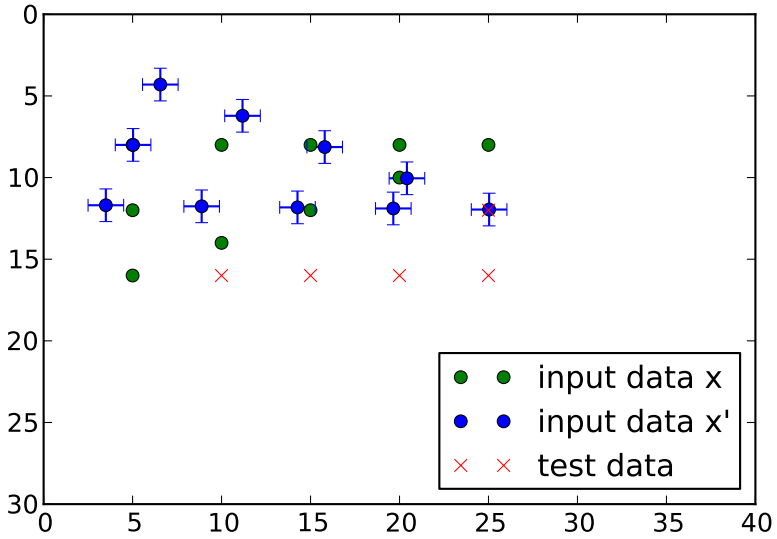
$$\Sigma_{\mathbf{p}} = \left({}^{\mathrm{t}}\mathbf{J}_{\mathbf{X}'}(\mathbf{p})\, \Sigma_{\mathbf{X}'}^{-1}\, \mathbf{J}_{\mathbf{X}'}(\mathbf{p}) \right)^{+} = \left(\sum_i {}^{\mathrm{t}}\mathbf{J}_i\, \Sigma_{\hat{\mathbf{x}}'_i}^{-1}\, \mathbf{J}_i \right)^{+} \tag{4.27}$$

By that, the framework is extended not only to estimate the homography but also to compute the covariance matrix. Consequently, the confidence of the estimation is no longer unknown to the system.

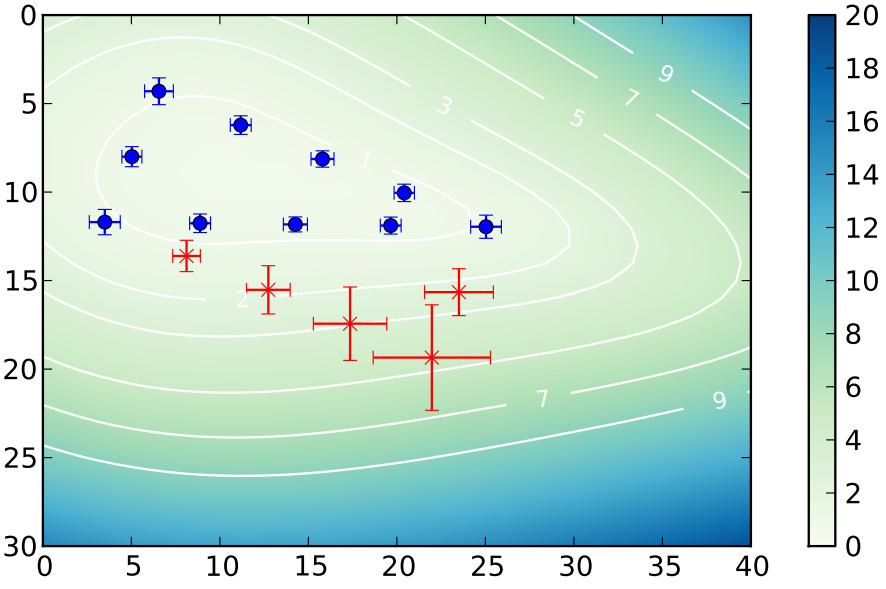
4.10.2 Covariance of a Transformed Point

Now that the covariance of the homography matrix is known, the following question states how confident the transformation of an arbitrary point $\mathbf{x}$ to $\mathbf{x}'$ is. Given the covariance $\Sigma_{\mathbf{p}}$ of the homography, the covariance of $\mathbf{x}'$ can be computed using (4.26) and (4.24):

$$\Sigma_{\hat{\mathbf{x}}'} = \mathbf{J}_{\hat{\mathbf{x}}'}(\mathbf{p}) \cdot \Sigma_{\mathbf{p}} \cdot {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{x}}'}(\mathbf{p}) \tag{4.28}$$



(a) Input data used to compute the transformation from green to blue pixels.



(b) Deviation σ_x and σ_y of transformed pixels. $\sqrt{\sigma_x^2 + \sigma_y^2}$ is color-coded. Additionally, some contour lines are given for better comparison to other images in this section.

Figure 4.43: The points marked as "input data x" and "input data x'" in (a) were used to compute the transformation matrix $\mathbf{P}_1$. While the points $\mathbf{x}_i$ are assumed to be precise, the target points $\mathbf{x}'_i$ have a standard deviation of 1px. Using $\mathbf{P}_1$, any pixel can be transformed which is shown for some pixels marked as "test data". However, their expected deviation varies dramatically with their position relatively to the input data as can be seen in (b).

Fig. 4.43 shows an example. The parameters of a homography were estimated using 10 point correspondences $\mathbf{x}_i \mapsto \mathbf{x}'_i$. The $\mathbf{x}_i$ are considered noise-free while the $\mathbf{x}'_i$ have a standard deviation of $\sigma_\mathrm{x} = \sigma_\mathrm{y} = 1\mathrm{px}$. When applying the obtained transformation to the input points, their deviation is more or less as specified. However, when transforming some test points that were not included in the estimation, their deviation increases dramatically with higher distance from the input points.

As the proposed framework computes the transformation of a layer using its visible pixels, the use of (4.28) allows for estimating its uncertainty with respect to invisible pixels. A case in which large deviations are to be expected was shown in Fig. 4.41.

4.10.3 Covariance of a Matrix Product

In the previous sections it was described how the covariance of a homography can be obtained using the point correspondences. Since the proposed framework computes the transformations of layers by chaining frame-to-frame homographies, the propagation of the covariance needs to be concerned.

Uncertainty in both Matrices

As described in section 4.2, the transformation matrices obtained for subsequent frames are multiplied in order to update the mapping of the corresponding layer to the current frame:

$$\begin{aligned}\mathbf{M}_n &= \mathbf{P}_n \cdot \mathbf{P}_{n-1} \cdot ... \cdot \mathbf{P}_1 \cdot \mathbf{M}_0 \\ &= \mathbf{P}_n \cdot \mathbf{M}_{n-1}\end{aligned}$$

Given the covariance $\Sigma_{\mathbf{m}_{n-1}}$ of the layer's transformation to the last frame and the covariance $\Sigma_{\mathbf{p}_n}$ of its transformation between the last and the current frame, the covariance $\Sigma_{\mathbf{m}_n}$ of the multiplied matrix needs to be computed. For simplicity, this multiplication is now denoted as $\mathbf{M} = \mathbf{P}_1\mathbf{P}_2$. The parameter vectors of these matrices are denoted as $\mathbf{m} = {}^\mathrm{t}(m_{11}, m_{12}, ..., m_{33})$, $\mathbf{p}_1 = {}^\mathrm{t}(p^1_{11}, p^1_{12}, ..., p^1_{33})$ and $\mathbf{p}_2 = {}^\mathrm{t}(p^2_{11}, p^2_{12}, ..., p^2_{33})$.

Since $\mathbf{M}$ is a homography matrix and thus only defined up to a constant, the normalized matrix

$$\hat{\mathbf{M}} = \frac{\mathbf{P}_1\,\mathbf{P}_2}{\|\mathbf{P}_1\,\mathbf{P}_2\|}$$

is considered in the following, just as in section 4.10.1. The covariance of the corresponding parameter vector $\hat{\mathbf{m}} = {}^\mathrm{t}(\hat{m}_{11}, \hat{m}_{12}, ..., \hat{m}_{33})$ is computed by applying

(4.24). For that, the dependence of $\hat{\mathbf{m}}$ with respect to $p^1_{11}, p^1_{12}, ..., p^1_{22}, p^2_{11}, p^2_{12}, ..., p^2_{22}$ has to be established – it is given by the Jacobian matrix $\mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_1, \mathbf{p}_1)$ which is a 9×18 matrix with the decomposition

$$\begin{bmatrix} \mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_1) & \mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_2) \end{bmatrix}.$$

The normalization of $\hat{\mathbf{M}}$ respectively $\hat{\mathbf{m}}$ requires the use of the product and quotient rule when computing the derivates:

$$\begin{aligned} \mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_1, \mathbf{p}_2) &= \frac{1}{\|\mathbf{m}\|} \frac{\partial \mathbf{m}}{\partial [\mathbf{p}_1 \ \mathbf{p}_2]} + \mathbf{m} \frac{\partial \|\mathbf{m}\|^{-1}}{\partial [\mathbf{p}_1 \ \mathbf{p}_2]} \\ &= \frac{1}{\|\mathbf{m}\|} \frac{\partial \mathbf{m}}{\partial [\mathbf{p}_1 \ \mathbf{p}_2]} - \frac{\mathbf{m}}{\|\mathbf{m}\|^2} \frac{\partial \|\mathbf{m}\|}{\partial [\mathbf{p}_1 \ \mathbf{p}_2]} \\ &= \frac{\mathbf{J}_{\mathbf{m}}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{m}\|} - \hat{\mathbf{m}} \frac{\mathbf{J}_{\|\mathbf{m}\|}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{m}\|} \end{aligned}$$

The Jacobian matrix of the unnormalized $\mathbf{m}$ is given by

$$\mathbf{J}_{\mathbf{m}}(\mathbf{p}_1, \mathbf{p}_2) = \begin{bmatrix} {}^{\mathrm{t}}\mathbf{P}_2 & & & \mathbb{1}\, p^1_{00} & \mathbb{1}\, p^1_{01} & \mathbb{1}\, p^1_{02} \\ & {}^{\mathrm{t}}\mathbf{P}_2 & & \mathbb{1}\, p^1_{10} & \mathbb{1}\, p^1_{11} & \mathbb{1}\, p^1_{12} \\ & & {}^{\mathrm{t}}\mathbf{P}_2 & \mathbb{1}\, p^1_{20} & \mathbb{1}\, p^1_{21} & \mathbb{1}\, p^1_{22} \end{bmatrix}$$

with the 3×3 identity matrix $\mathbb{1}$. The 1×18 Jacobian matrix of the Frobenius norm of vector $\mathbf{m}$ is

$$\mathbf{J}_{\|\mathbf{m}\|}(\mathbf{p}_1, \mathbf{p}_2) = \begin{bmatrix} \hat{\mathbf{M}}^1 \, {}^{\mathrm{t}}\mathbf{P}_2 & \hat{\mathbf{M}}^2 \, {}^{\mathrm{t}}\mathbf{P}_2 & \hat{\mathbf{M}}^3 \, {}^{\mathrm{t}}\mathbf{P}_2 & \hat{\mathbf{M}}^1 \, {}^{\mathrm{t}}\mathbf{P}_1 & \hat{\mathbf{M}}^2 \, {}^{\mathrm{t}}\mathbf{P}_1 & \hat{\mathbf{M}}^3 \, {}^{\mathrm{t}}\mathbf{P}_1 \end{bmatrix},$$

where $\hat{\mathbf{M}}^i$ denotes the i-th row of $\hat{\mathbf{M}}$.

Finally, the covariance matrix of the parameters is given by

$$\Sigma_{\mathbf{p}_1, \mathbf{p}_2} = \begin{bmatrix} \mathbf{\Sigma}_{\mathbf{p}_1} & \\ & \mathbf{\Sigma}_{\mathbf{p}_2} \end{bmatrix}$$

since zero correlation between $\mathbf{p}_1$ and $\mathbf{p}_2$ is assumed. The covariance of $\hat{\mathbf{M}}$, respectively $\hat{\mathbf{m}}$, is then given by

$$\Sigma_{\hat{\mathbf{m}}} = \mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_1, \mathbf{p}_2) \cdot \Sigma_{\mathbf{p}_1, \mathbf{p}_2} \cdot {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}_1, \mathbf{p}_2) \tag{4.29}$$

according to (4.24).

Continuing the example from Fig. 4.43, another transformation is computed in Fig. 4.44. Chaining both transformations maps the pixels directly, and the covariance is computed as given in (4.29). Fig. 4.45 shows the respective plot. It can be compared to Fig. 4.46 showing the result for a transformation that was

directly computed with the points $\mathbf{x}_i$ from Fig. 4.43 and $\mathbf{x}'_i$ from Fig. 4.44. It is obvious that the expected deviation is lower. This fact is the justification for the proposed skipping mode (see section 4.8.2) that computes the transformation not successively from frame to frame but between distant frames. However, this is only possible if the points can be matched between any frames which is not the case when using optical flow. It only offers point correspondences for subsequent frames. (For that reason, the skipping mode currently only works if line correspondences are available.)

Consequently, a successive computation of the transformation is still necessary so that (4.29) for estimation the propagation of covariance is an important result.

Uncertainty in one Matrix

A simpler case is a matrix product were only one matrix is assumed to have an uncertainty. Regarding the update process

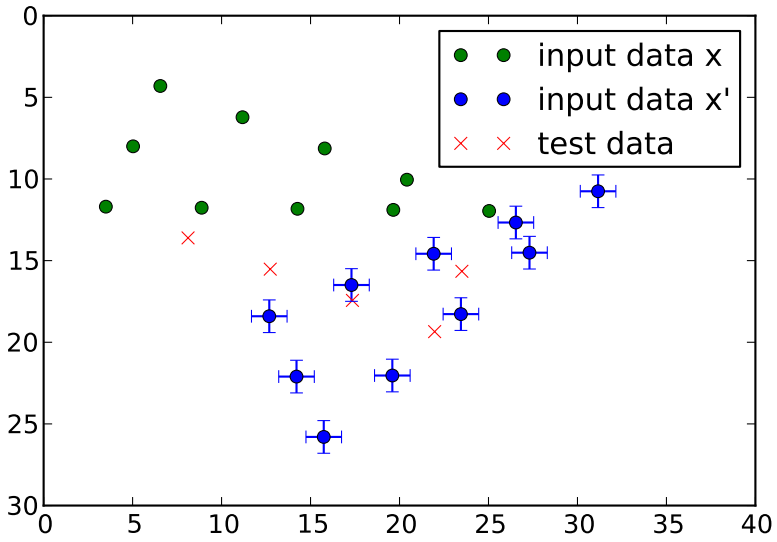
$$\mathbf{M}_n = \mathbf{P}_{0,n} \cdot \mathbf{M}_0 = \underbrace{\mathbf{P}_n \cdot \mathbf{P}_{n-1} \cdot \ldots \cdot \mathbf{P}_1}_{\text{covariance } \Sigma_{\mathbf{P}_{1,n}}} \cdot \underbrace{\mathbf{M}_0}_{\text{fixed}}$$

of the transformation, the matrices $\mathbf{P}_i$ are obtained using optical flow while $\mathbf{M}_0$ ("placement matrix") represents the position of the layer in the first frame. As described in section 4.1, this is done in order to minimize the layer's size. During this layer initialization, the segment is just copied to a new array which shift relative to the image origin is stored in $\mathbf{M}_0$. This matrix is assumed to have no uncertainty.

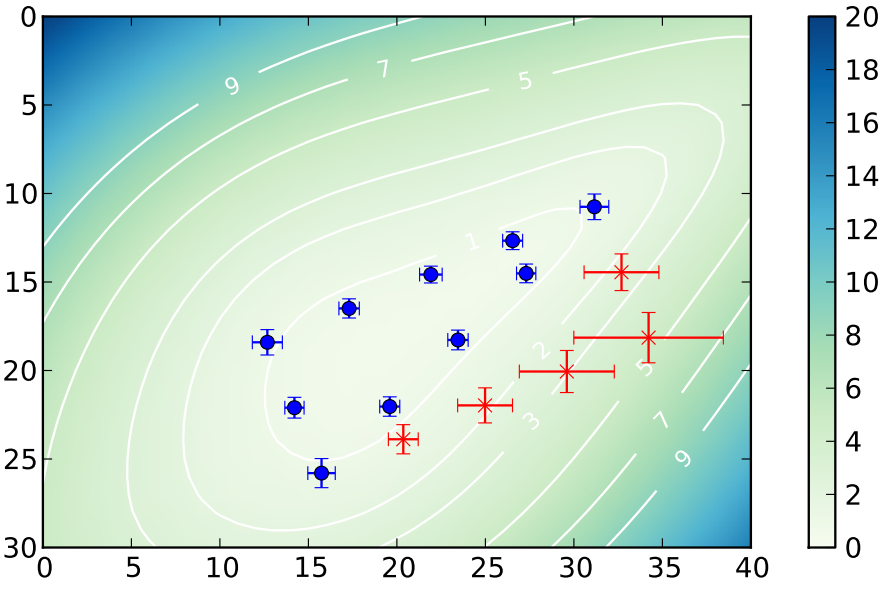
In the next frames, given the transformation $\mathbf{P}_{0,n}$ and its covariance matrix $\Sigma_{\mathbf{P}_{0,n}}$, the covariance of $\mathbf{M}_n = \mathbf{P}_{0,n} \cdot \mathbf{M}_0$ needs to be computed where only the left matrix has an uncertainty

Another case in which this is needed is the change of a layer's perspective described in section 4.8.4: The transformed layer is stored into a new array and the transformation matrix $\mathbf{M}_n$ is reset to a placement matrix (4.23) which is denoted as $\mathbf{M}^*_n$. Since a placement matrix of a layer is always treated as a fixed matrix without uncertainty, the covariance information of the transformation that had been computed until this frame, would get lost. As a consequence, the covariance of the transformed layer pixels would be zero, which is wrong since the change of perspective did not reduce or compensate the uncertainty of the layer but only altered the data representation.

In order to preserve the estimation of the uncertainty, the transformation after the change of perspective is denoted as $\mathbf{M}'_n = \mathbf{P}^*_n \mathbf{M}^*_n$ with the identity matrix $\mathbf{P}^*_n$. For this identity matrix, a covariance matrix $\Sigma_{\mathbf{P}^*_\mathbf{n}}$ has to be found so that $\mathbf{M}'_n$ and its $\Sigma_{\mathbf{M}'_\mathbf{n}}$ produce the same point uncertainties (using (4.28)) as $\mathbf{M}_n$ and



(a) Input data used to compute the transformation from green to blue pixels.



(b) Deviation σ_x and σ_y of transformed pixels. $\sqrt{\sigma_x^2 + \sigma_y^2}$ is color-coded. The contour lines are at same levels as those in Fig. 4.44.

Figure 4.44: The target pixels $\mathbf{x}'_i$ from Fig. 4.43 are used to define a new set of input data $\mathbf{x}_i$ in order to compute a second transform $\mathbf{P}_2$ that moves them to the $\mathbf{x}'_i$. As before, an uncertainty of again $1\mathrm{px}$ is assumed in the measurement of $\mathbf{x}'_i$.

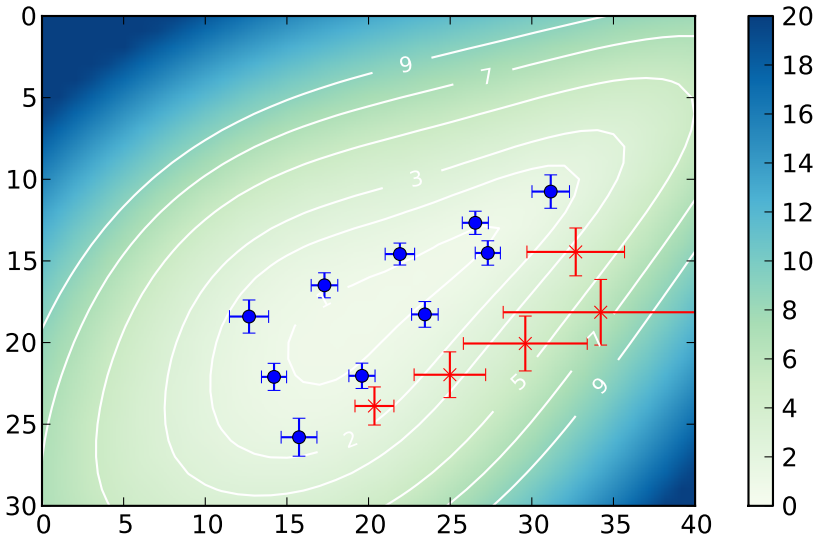
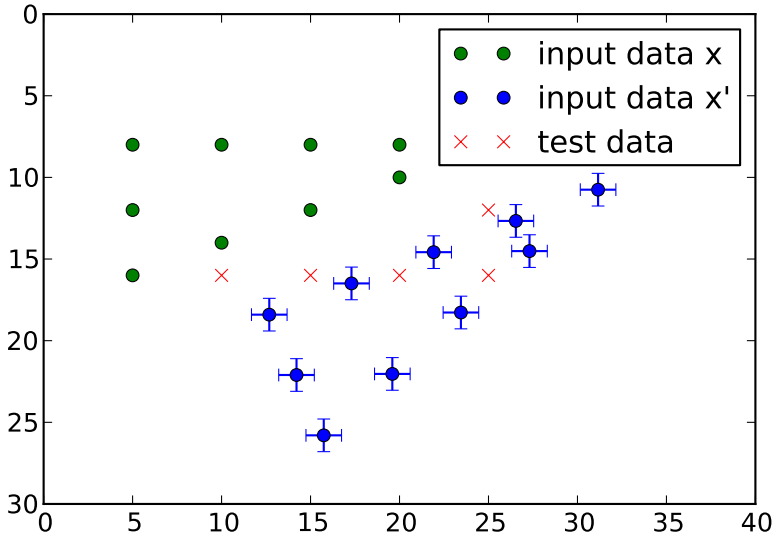
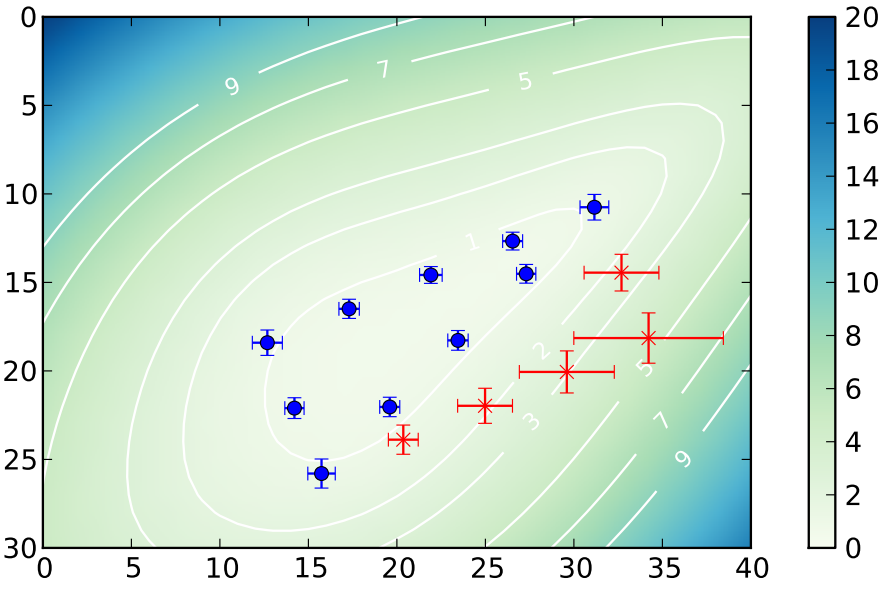


Figure 4.45: After the partial transformations $\mathbf{P}_1$ and $\mathbf{P}_2$ have been computed together with their covariances, the combined transformation $\mathbf{M} = \mathbf{P}_2\mathbf{P}_1$ can be applied to the initial pixels $\mathbf{x}_i$ shown in Fig. 4.43(a) so that they are directly transformed to the $\mathbf{x}'_i$ shown in Fig. 4.44(a). As the covariance of $\mathbf{M}$ is given by (4.29), the covariance of each pixel can be obtained by (4.28), the result is shown here for the input and the test data, as well as color-coded for all other pixels.



(a) Input data used to compute the transformation matrix



(b) Deviation σ_x and σ_y of transformed pixels. $\sqrt{\sigma_x^2 + \sigma_y^2}$ is color-coded.

Figure 4.46: For comparison, a transformation $\mathbf{P}_{12}$ is computed directly from the $\mathbf{x}_i$ in Fig. 4.43 to the $\mathbf{x}'_i$ in Fig. 4.44. It can be seen that the covariance of the transformed pixels is lower than for $\mathbf{P}_2\mathbf{P}_1$ which is shown in Fig. 4.45.

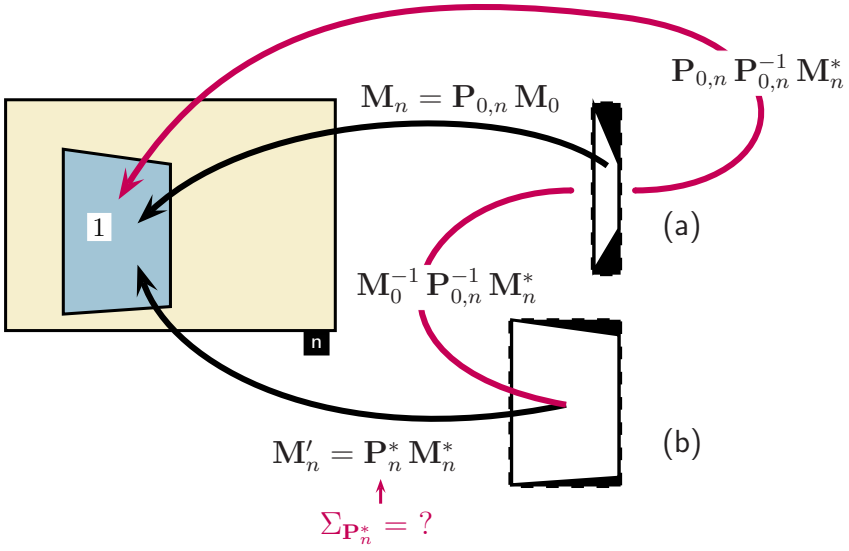


Figure 4.47: Derivation of the covariance after a change of perspective, which was described in (4.23). (a) shows the old layer, (b) the new. See text for further details.

its $\Sigma_{\mathbf{M_n}}$ did before. The solution for this is derived as follows, see Fig. 4.47 for a visualization. A transformation from the new and the old layer is given by $\mathbf{M}_0^{-1}\,\mathbf{P}_{0,n}^{-1}\,\mathbf{M}_n^*$. By chaining the transformation of the old layer to the current frame $\mathbf{M}_n = \mathbf{P}_{0,n}\,\mathbf{M}_0$, one obtains $\mathbf{P}_{0,n}\,\cancel{\mathbf{M}_0}\,\cancel{\mathbf{M}_0^{-1}}\,\mathbf{P}_{0,n}^{-1}\,\mathbf{M}_n^*$, which has to be equal to $\mathbf{M}'_n = \mathbf{P}_n^*\,\mathbf{M}_n^*$. Consequently, the solution is to implicitly transform the new layer back to its appearance in the first frame using $\mathbf{P}_{0,n}^{-1}$ and then transform it back again with respect to the old covariance matrix $\Sigma_{\mathbf{P}_{0,n}}$:

$$\mathbf{P}_n^* = \underbrace{\mathbf{P}_{0,n}}_{\text{cov. } \Sigma_{\mathbf{P}_{0,n}}} \cdot \underbrace{\mathbf{P}_{0,n}^{-1}}_{\text{fixed}}$$

Again, only the left matrix has a covariance. For simplicity, this matrix product is now denoted as $\mathbf{M} = \mathbf{PT}$ where $\mathbf{T}$ is fixed and $\mathbf{P}$ has an uncertainty given by the covariance $\Sigma_{\mathbf{p}}$. The normalized matrix $\hat{\mathbf{M}} = \frac{\mathbf{PT}}{\|\mathbf{PT}\|}$ has the parameter vector $\hat{\mathbf{m}}$ and the computation of its covariance matrix is very similar to the one in the previous section. It is

$$\Sigma_{\hat{\mathbf{m}}} = \mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}) \cdot \Sigma_{\mathbf{p}} \cdot {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}) \tag{4.30}$$

with

$$\mathbf{J}_{\hat{\mathbf{m}}}(\mathbf{p}) = \frac{\mathbf{J}_{\mathbf{m}}(\mathbf{p})}{\|\mathbf{m}\|} - \hat{\mathbf{m}}\,\frac{\mathbf{J}_{\|\mathbf{m}\|}(\mathbf{p})}{\|\mathbf{m}\|}$$

according to (4.24). However, the Jacobian matrices

$$\mathbf{J}_{\mathbf{m}}(\mathbf{p}) = \begin{bmatrix} {}^{\mathrm{t}}\mathbf{T} & & \\ & {}^{\mathrm{t}}\mathbf{T} & \\ & & {}^{\mathrm{t}}\mathbf{T} \end{bmatrix}$$

and

$$\mathbf{J}_{\|\mathbf{m}\|}(\mathbf{p}) = \begin{bmatrix} \hat{\mathbf{M}}^1\,{}^{\mathrm{t}}\mathbf{T} & \hat{\mathbf{M}}^2\,{}^{\mathrm{t}}\mathbf{T} & \hat{\mathbf{M}}^3\,{}^{\mathrm{t}}\mathbf{T} \end{bmatrix},$$

where $\hat{\mathbf{M}}^i$ denotes the i-th row of $\hat{\mathbf{M}}$, only depend on $\mathbf{p}$ and not on $\mathbf{t}$ since the latter has no uncertainty.

4.10.4 Computation of the Covariance Using Lines

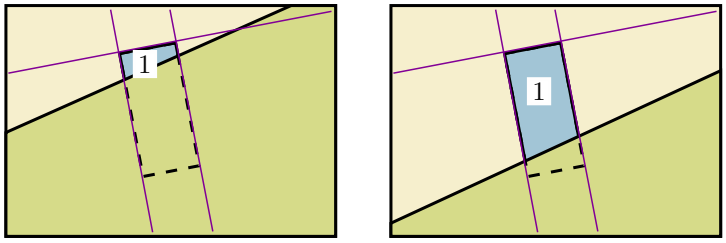
In section 4.10.1 was described, how the covariance matrix of a homography is computed. However, result (4.27) only takes the point correspondences $\mathbf{x}' = \mathbf{P}\mathbf{x}$ into account. Since the framework also uses line correspondences $\mathbf{l}' = {}^{\mathrm{t}}\mathbf{P}^{-1}\mathbf{l}$ to compute the homographies, this computation has to be extended.

On way is to extend (4.27) by a second sum over the line vectors

$$\Sigma_{\mathbf{p}} = \left(\sum_i {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{x}}'_i}\, \Sigma^{-1}_{\hat{\mathbf{x}}'_i}\, \mathbf{J}_{\hat{\mathbf{x}}'_i} + \sum_j {}^{\mathrm{t}}\mathbf{J}_{\hat{\mathbf{l}}'_j}\, \Sigma^{-1}_{\hat{\mathbf{l}}'_j}\, \mathbf{J}_{\hat{\mathbf{l}}'_j} \right)^{+},$$

where $\Sigma_{\hat{\mathbf{l}}'_j}$ denotes the covariance of each line vector and $\mathbf{J}_{\hat{\mathbf{l}}'_j}$ is the Jacobian matrix of $\hat{\mathbf{l}} = \frac{{}^{\mathrm{t}}\mathbf{P}^{-1}\mathbf{l}}{\|{}^{\mathrm{t}}\mathbf{P}^{-1}\mathbf{l}\|}$. The problem of this approach lies in obtaining $\Sigma_{\hat{\mathbf{l}}'_j}$. For point correspondences, the expected variance of the point coordinates can be set to a constant value. This is not that easy with line correspondences, since the accuracy of a line depends on the distribution of the belonging points. Fig. 4.48 shows an example for that. During the computation of the Hough transform, the distinct pixel coordinates get lost (by design) when creating the accumulator. In other words, they they do not have a representation in the parameter space which only holds a line's angle and distance from the origin. One could take the level of the peak in the accumulator into account, but the number of border pixels, which an accumulator entry stands for, does not say anything about their distribution in the image. Hence, re-establishing the pixel coordinates would be an additional effort but is necessary for a correct covariance estimation of each line.

In the proposed framework, this is left out because using the available point correspondences has turned out to be sufficient for the error estimation of the homography. In skipping mode, where no point correspondences are available, those are artificially created by $\mathbf{x}_i = \mathbf{P}^{-1}\,\mathbf{s}_i$ where $\mathbf{s}_i$ denotes all pixels of the



(a) (b)

Figure 4.48: Although the lines found by the Hough transform are identical in (a) and (b), the accuracy of the two vertical lines lines in (b) is higher since the border pixels are better distributed. The Hough transform itself cannot recognize this.

segment in the current frame. Note that (4.27) only contains the points $\mathbf{x}_i$ in the first image and the expected variance of their correspondences in the second image, but not any measurement in the second image – the vector $\mathbf{x}'_i$ is given by $\mathbf{P}\mathbf{x}_i$. For that reason, no real point correspondences need to be available. With that "trick", a simple though approximative estimation of the covariance in skipping mode is achieved (see Fig. 4.49 together with Fig. 4.37).

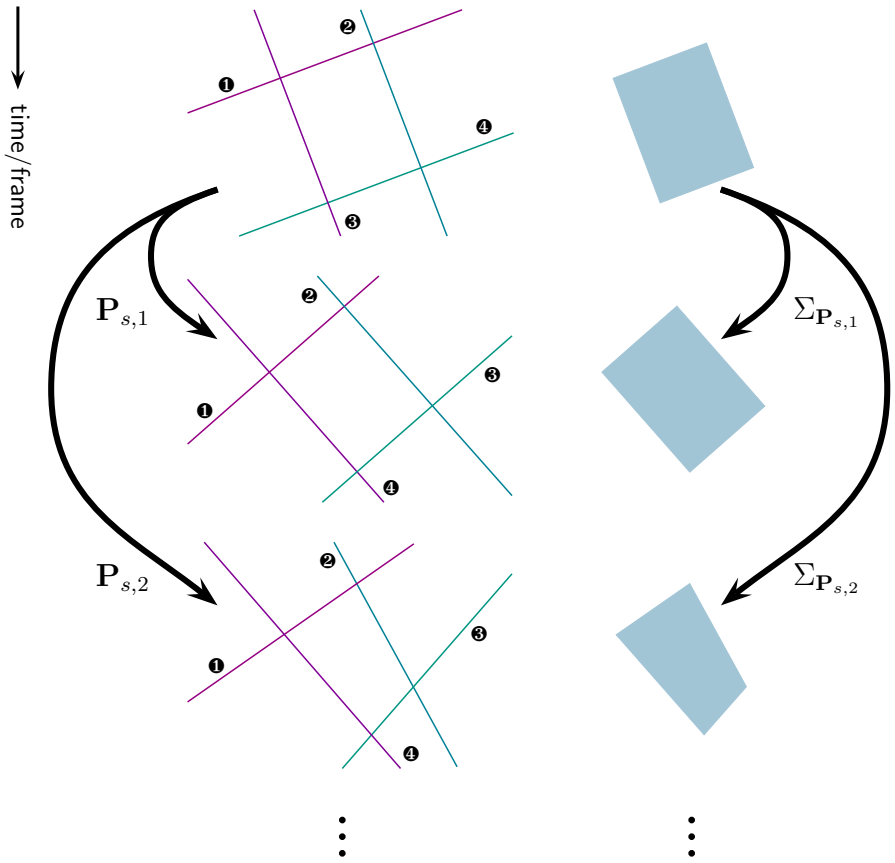


Figure 4.49: In skipping mode (see also Fig. 4.37), the transformations $\mathbf{P}_{s,n}$ are computed using line correspondences, while the belonging covariance matrices $\Sigma_{\mathbf{P}_{s,n}}$ are obtained using the segment pixels. Although this is not exact, this is used because, with this implementation, it is not necessary to compute individual line variances (see text).

Chapter 5

Ascertaining Object Permanence

The layer framework described in the previous chapter can be seen as a toolkit for higher level analyses. It will be used now to recognize segment splits and to track occluded segments.

5.1 Re-merging Split Segments

One of the most important problems of building semantic scene graphs is that the output of the image segmentation is is not stable against occlusions. The most essential issue is the "splitting problem" which is addressed in this section.

By tracking the segments and mapping the layers to the frame, hidden parts of the segments can be determined. The key aspect of this procedure is that new segments can be checked whether they belong to an already known segment. This is illustrated in Fig. 5.1. Segment 2 moves behind segment 1 and gets partially occluded. The layer framework can track the hidden parts of 2. As it comes out on the other side of 1, the segmentation algorithm will create a new segment with a new label. Due to the tracking of layer 2 it can be established that the new segment is a part of it.

To do this, each time a new segment is found, the framework checks all available layers whether the new segment fits into it. Note that the mapping of the layers has an expected deviation as described in section 4.10. Hence, a layer is not only transformed to the frame – an expected deviation of each transformed pixel is also obtained. Regarding the outer pixels of a layer, their expected deviations σ_x and σ_y define an area around the layer. According to *Chebyshev's inequality*, with a probability of 95%, a transformed pixel lies within a range of 2σ around the mean value. Hence, when mapping a layer to the frame, this is not done pixel-wise – instead of that, each pixel of the layer is mapped as an rectangle with side lengths $2\sigma_x$ and $2\sigma_y$ to the frame.

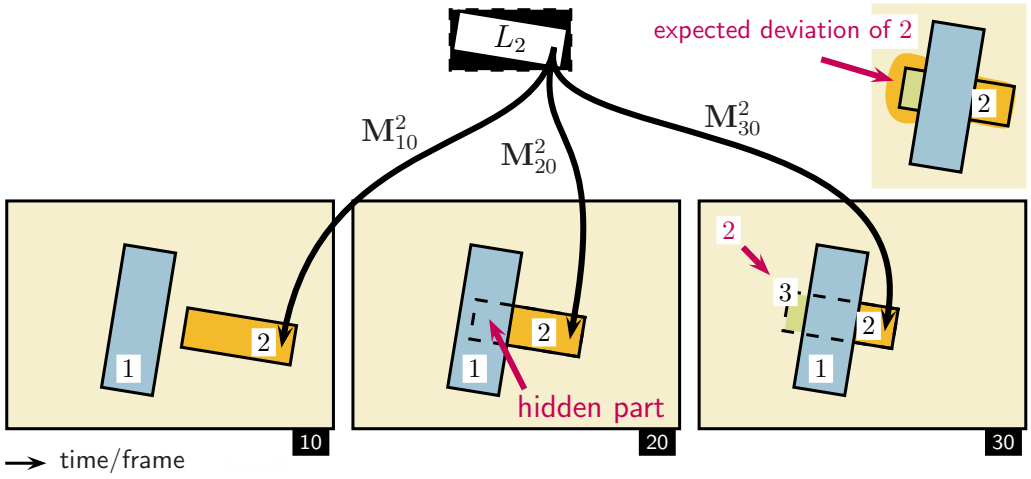


Figure 5.1: Segment 2 moves behind segment 1. By tracking layer 2, the position of its hidden parts is always known. That way, a new segment appearing when 2 comes out at the other side of 1 can be relabeled to 2. The method takes the expected deviation of 2 into account (see text).

The resulting area covers all pixels in the image that might belong to the layer with at least 95% probability. The area exceeds the original layer mostly in areas that are lateral to visible pixels of the layer as discussed in 4.10. Since the accuracy of the transformed pixels in these regions is low, the matching of a new segment with the original layer could be bad in extreme situations. Since the area increases with larger deviations, this is not the case with the proposed method. Naturally, the error estimate has to be correct. Else, recognition of splitting can fail.

An additional condition to recognize that a new segment belongs to an already known one, is that both move coherently. In terms of homographies, the transformation matrices have to be same. If this is not the case, both parts cannot belong together. Hence, the distance of each point transformed by $\mathbf{P}^2_{n-1,n}$ and $\mathbf{P}^3_{n-1,n}$, which determine the segment's transformations from the last to the current frame, is computed. Then, the average distance of all points is checked. In case of a perfect match, it should be zero. However, since the matrices where estimated with some error captured by their covariance matrices, an error threshold is accepted given by the sum of expected deviations of the transformed points.

5.2 Tracking Hidden Segments

The layer framework, that was described up to here, only works with segments that are always visible apart from partial occlusions. If a segment is completely occluded, no transformation matrix can be computed and, as a consequence, no further tracking is possible. However, humans are able to do this, at least to a certain extend: Consider a hand holding an object. If the hand moves behind another object, the rest of the arm still allows us to estimate where the object is. A more advanced example is the "shell game" where a small ball becomes occluded by one of three shells. After shuffling the shells around by the operator, the player has to guess under which shell the ball is. On the one hand these examples show that a human is able to track hidden objects even when they are moving. On the other hand, it proves that even the brain can be trapped in that complicated task which is why the shell game operator usually makes a lot of money with it.

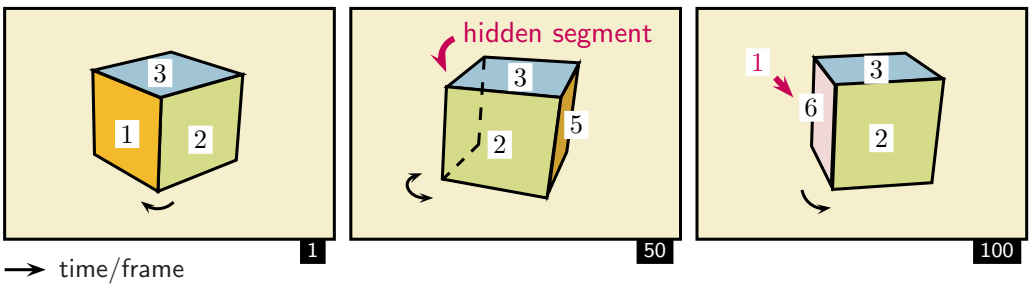


Figure 5.2: The surfaces of a rotating box vanish and reappear. In the latter case, they are assigned a new label by a standard pixel-based segmentation algorithm. Only a high-level method can recognize already known segments and restore their old label.

In the current work, this problem is approached using a far more basic example. Consider a box (see Fig. 5.2) that is rotating so that one of its surfaces becomes invisible while another appears. If the box rotates further or if it rotates back, a hidden but already known surface will become visible again. Considering the surfaces as segments, a standard pixel-based segmentation algorithm is not able to recognize this and will assign the segment a new label as if the surface was unknown.

Obviously, this problem is related to the "splitting problem" since one could regard the box are being split so that one of its parts (the reappearing side) is assigned a new label. However, since the sides of the box are not lying on

a common plane, this problem cannot be solved with the method proposed in section 5.1. This section deals with extending the proposed framework by mechanisms to track invisible segments based on visible segments which have previously been observed to move in a coherent way. This will be referred as the "binding problem". Two methods with different advantages/drawbacks are proposed.

5.2.1 Line-based Correlation Determination

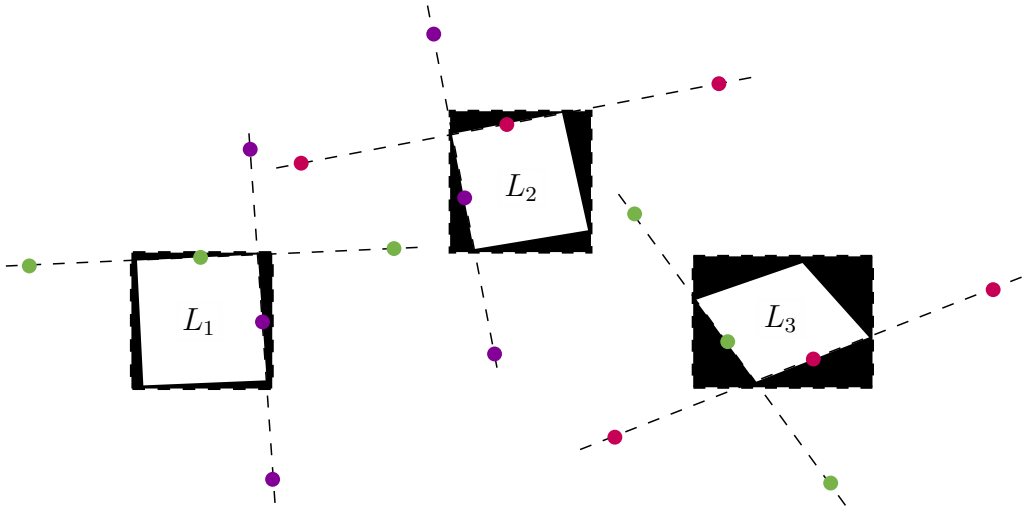
Regarding Fig. 5.2, it can be seen that the hidden segment 1 is not completely invisible since its junction lines to segment 2 and 3 are still observable. Hence, when dealing with squared objects like the box, this can be used to compute the transformation matrix of a hidden segment based on two bordering segments.

However, not every neighbor segment can be used for that purpose, for example the background. Obviously, the background is completely independent from the hidden segment and does not hold any information about its position. The decision, which two segments can be used to obtain a hidden segment's position, can only be made in advance while the segment is still visible. This step will be called *correlation determination*.

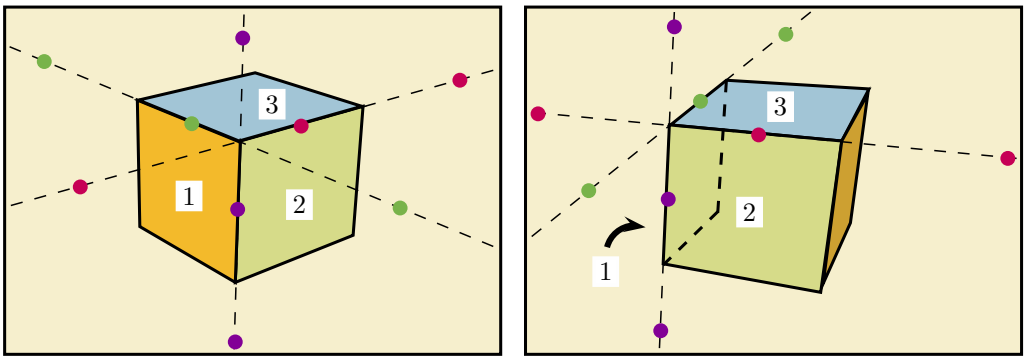
The boundary lines in the segmented image are obtained segment-wise as described in section 4.4. The first task is to find common lines between segment pairs. Furthermore, it has to be checked with each new frame, if these line-matches are still valid. If not, it might have been the case of a "wrong line" in the sense of section 4.8.1, e.g. a line between an object and the background. It is also possible that a former correlation between segments just broke up. However, if a stable common line could be established for two segments, this will be called a *correlation*.

If one segment is correlated to two other segments, its position can be established at least on the condition that the corresponding lines are not identical (lie on top of each other). As described in section 4.3, the computation of the transformation matrix requires at least four points or lines. As a consequence, knowing two common lines is not enough – it is necessary to establish four points on those two lines in a general position which means that there are not three points in one line.

The implementation of the correlation determination is as follows. The layers are extended so that so-called *alignment points* (AP) can be stored by a unique ID. It would also be possible to store the APs within the layer array but it is more efficient to save them separately in an associative array $\mathrm{ID}_i \mapsto \mathbf{z}_i$ where $\mathbf{z}_i$ is the AP's location in the layer's coordinate system. New AP's are added when a common line between two segments was found. In that case, at least two points on that line are selected and stored as new APs in both layers (see Fig. 5.3). Since the line was found in image coordinates, each new point has



(a) Layers with alignment points



(b) (c)

Figure 5.3: The layers are extended to store alignment points (a). Those are mapped to the frame just like the ordinary layers (b). If, in reverse, the positions of the alignment points in the frame are known, they can be used to transform the whole layer. This is done in (c) for invisible segment 1 by using the positions of the alignment points found in layer 2 and 3.

to be transformed to the coordinate system of the corresponding layer using its inversed transformation matrix. Therefore, each of the new points is transformed according to the respective layer but stored under the same ID. That way, it will later be possible to match APs in different layers.

So, while each layer has its own list of APs, two layers can include one or more common APs (with identical ID) in their list. Now, given the positions $\mathbf{z}'_i$ in the frame of a sufficient number of APs, a transformation matrix $\mathbf{M}$ can be computed that transforms the layer so that the APs match the desired positions: $\mathbf{z}'_i = \mathbf{M}\,\mathbf{z}_i$. Consequently, this is an alternative method to map the layer to the frame. Contrary to the transformation update processes in "successive" and "skipping mode" (described in section 4.2 respectively 4.8.2), the computation of the matrix is done directly between layer and frame using the point correspondences $\mathbf{z}_i \leftrightarrow \mathbf{z}'_i$ (see section 4.3 for a description of the estimation of homographies).

Obviously, the AP's target positions $\mathbf{z}'_i$ have to be known. If a layer is invisible and its mapping to the frame shall be computed using its APs, all currently visible layers have to be checked whether they include some common APs recognizable by their matching ID. If this is the case, the positions of the APs in the frame are given by the visible layer's transformation. As an example, let the APs of a hidden layer be $\mathbf{z}_1^{\mathrm{h1}}, ..., \mathbf{z}_4^{\mathrm{h1}}$ where the lower index denotes the AP's ID. Furthermore, let $\mathbf{z}_1^{\mathrm{v2}}, \mathbf{z}_2^{\mathrm{v2}}$ and $\mathbf{z}_3^{\mathrm{v3}}, \mathbf{z}_4^{\mathrm{v3}}$ be the AP's of two visible layers. Note that segment $\mathrm{v2}$ has the APs with ID 1 and 2 in common with $\mathrm{h1}$, while $\mathrm{v3}$ has with it the common APs 3 and 4. $\mathbf{M}_n^{\mathrm{v2}}$ and $\mathbf{M}_n^{\mathrm{v3}}$ are the transformation matrices of the visible layers to the current frame which have been obtained by the standard update process. Then, the transformation matrix $\mathbf{M}_n^{\mathrm{h1}}$ can be computed using the following correspondences:

$$
\begin{aligned}
\mathbf{z}_1^{\mathrm{h1}} &\longleftrightarrow \mathbf{z}_1^{\mathrm{h1}'} = \mathbf{M}_n^{\mathrm{v2}}\mathbf{z}_1^{\mathrm{v2}} \\
\mathbf{z}_2^{\mathrm{h1}} &\longleftrightarrow \mathbf{z}_2^{\mathrm{h1}'} = \mathbf{M}_n^{\mathrm{v2}}\mathbf{z}_2^{\mathrm{v2}} \\
\mathbf{z}_3^{\mathrm{h1}} &\longleftrightarrow \mathbf{z}_3^{\mathrm{h1}'} = \mathbf{M}_n^{\mathrm{v3}}\mathbf{z}_3^{\mathrm{v3}} \\
\mathbf{z}_4^{\mathrm{h1}} &\longleftrightarrow \mathbf{z}_4^{\mathrm{h1}'} = \mathbf{M}_n^{\mathrm{v3}}\mathbf{z}_4^{\mathrm{v3}}
\end{aligned}
$$

This is the minimum case of four correspondences that need to be in a general position. If more AP's are available, they all enter the computation making it more stable. Consequently, when a new common line is established, 20 APs are created.

Besides adding new APs, it is important to check if the already stored APs are still valid. Theoretically, the APs with identical IDs should be mapped to the exact same image point by all layers including it. If not, the correlation is broken. However, since the individual homographies are estimated with an error, a correlation has to be maintained within a range of deviation. Since the expected

variance of each transformation is known to the system (see section 4.10), the APs also have a deviation estimate. The allowed distance between two APs with same ID is then given by the sum of both deviations. If this range is exceeded, the APs are deleted. However, this does not mean that the two layers are not correlated anymore – this is only then the case when *all* common APs have been deleted.

5.2.2 Homography-based Correlation Determination

The most important drawback of the previously described method is that correlated layers have to have junction lines. This restricts its applicability to rectangular segments. In order to cope with arbitrary segment shapes, another method for creating alignment points has to be found.

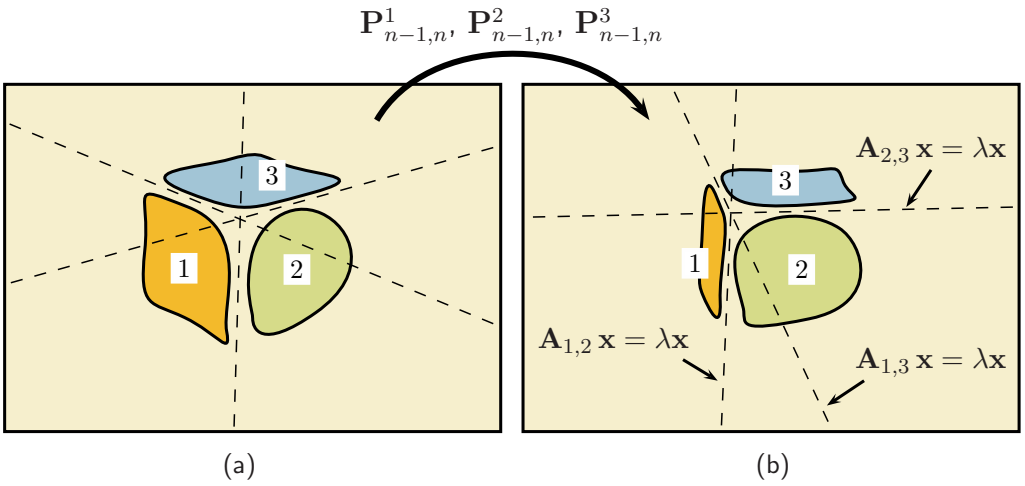


Figure 5.4: If the segments are not rectangular, their junction line is invisible. However, it can be found by computing the points that are equally transformed by the matrices $\mathbf{P}^1_{n-1,n}$ and $\mathbf{P}^2_{n-1,n}$ of the putatively correlated segments 1 and 2 (analog for pairs 1, 3 and 2, 3). It is given by the degenerate eigenspace of the relative homography $\mathbf{A}_{1,2}$ (see text).

The junction lines between two segments used in the previous section have the essential property that all points on it are transformed identically by the transformation matrices of both layers – if this was not the case the junction line would not be constant since the touching border pixels would move apart.

Instead of selecting those visible lines which fulfill this condition, it is also possible to use the condition to determine those lines (see Fig. 5.4). Given the

transformation matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ of two putatively correlated segments, one needs to find points that are transformed equally by both according to

$$\mathbf{P}_1 \mathbf{x} = c\mathbf{P}_2 \mathbf{x} \ , \tag{5.1}$$

where the factor c takes into account that the point vectors in homogeneous coordinates are defined only up to a scale. For this purpose, the *relative homography* between $\mathbf{P}_1$ and $\mathbf{P}_2$ is regarded:

$$\mathbf{A}_{12} = \mathbf{P}_1^{-1} \mathbf{P}_2$$

Transforming a point $\mathbf{x}$ according to $\mathbf{A}_{12}$, hence at first by $\mathbf{P}_2$ and afterwards backwards using the inverse of $\mathbf{P}_1$, one will reach the original point (up to a scale) in case (5.1). This can be written as

$$\mathbf{A}_{12} \mathbf{x} = \lambda \mathbf{x}.$$

Consequently, the *fixed points* of $\mathbf{A}_{12}$ are given by its eigenvectors. As this is a 3×3 matrix and has three eigenvectors, there can be up to three fixed points in case of distinct eigenvalues. However, if two eigenvalues, say λ_2 and λ_3, are identical but have distinct eigenvectors $\mathbf{e}_2$ and $\mathbf{e}_3$, then the whole line containing those vectors is fixed point-wise. All points on this line are given by $\mathbf{x} = \alpha\mathbf{e}_2 + \beta\mathbf{e}_3$ and consequently are eigenvectors of $\mathbf{A}_{12}$:

$$\mathbf{A}_{12} \mathbf{x} = \lambda_2 \alpha \mathbf{e}_2 + \lambda_2 \beta \mathbf{e}_3 = \lambda_2 \mathbf{x}$$

If even the eigenvectors are the same, then the transformation has one less fixed point. As a simple example for the determination of fixed points, let

$$\mathbf{P}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{P}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The eigenvalues of $\mathbf{P}_1^{-1}\mathbf{P}_2$ are $\lambda_1 = \lambda_3 = 1$ and $\lambda_2 = 0.5$ with the eigenvectors $\mathbf{x}_1 = {}^{\mathrm{t}}(1, 0, 0)$, $\mathbf{x}_2 = {}^{\mathrm{t}}(0, 1, 0)$ and $\mathbf{x}_3 = {}^{\mathrm{t}}(0, 0, 1)$. Consequently, all points on the line through $\mathbf{x}_1$ and $\mathbf{x}_3$ are fixed. $\mathbf{x}_1$ is the *point at infinity* on the x-axis since the euclidean coordinates are obtained by dividing by the last vector component and "$1/0 = \infty$". $\mathbf{x}_3$ is the origin. Hence, all points on the x-axis are fixed which is clear when looking at the matrices: $\mathbf{P}_1$ is the identity and $\mathbf{P}_2$ multiplies all y-values by two. The points that are transformed identically by both matrices must have $y = 0$.

A second example is

$$\mathbf{P}_1 = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{P}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The eigenvalues of $\mathbf{P}_1^{-1}\mathbf{P}_2$ are $\lambda_1 = \lambda_2 = \lambda_3 = 1$ with the eigenvectors $\mathbf{x}_1 = \mathbf{x}_3 = {}^{\mathrm{t}}(1,0,0)$ and $\mathbf{x}_2 = {}^{\mathrm{t}}(0,1,0)$. There are only two fixed points and both are at infinity. Consequently, no image points are transformed equally by $\mathbf{P}_1$ and $\mathbf{P}_2$ which is also obvious regarding the matrices: The first moves all points by 10 on the x-axis while the second is the identity.

As a last example, let $\mathbf{P}_1$ and $\mathbf{P}_2$ both be the identity. The eigenvalues of $\mathbf{P}_1^{-1}\mathbf{P}_2$ are $\lambda_1 = \lambda_2 = \lambda_3 = 1$ with $\mathbf{x}_1 = {}^{\mathrm{t}}(1,0,0)$, $\mathbf{x}_2 = {}^{\mathrm{t}}(0,1,0)$ and $\mathbf{x}_3 = {}^{\mathrm{t}}(0.09,0,1)$. These three eigenvalues define three point-wise fixed lines. However, since eigenvectors are orthogonal to each other, any point can be written as $\mathbf{x} = \alpha\mathbf{e}_1 + \beta\mathbf{e}_2 + \gamma\mathbf{e}_3$. It follows

$$\mathbf{A}_{12}\,\mathbf{x} = \lambda_1\alpha\mathbf{e}_1 + \lambda_1\beta\mathbf{e}_2 + \lambda_1\gamma\mathbf{e}_3 = \lambda_1\mathbf{x}$$

which shows that all points in the image are transformed equally.

Note, that the same considerations can be performed considering the matrix ${}^{\mathrm{t}}\mathbf{A}_{12}^{-1}$ which transforms line vectors (see section 4.3.2). By looking at the eigenvalues/eigenvectors of this matrix, fixed lines can be found. However, they are not necessarily fixed point-wise which means that all points on it are mapped to the same line but not to the same points on the line.

With that knowledge, the correlation determination, which only works on the transformation matrices of the layers, is performed as follows. For each frame, for each segment s the transformation $\mathbf{P}_{n-1,n}^{s}$ between the current and the previous frame is obtained. For each pair of segments, the relative homography is computed and checked for degenerate eigenvectors (eigenvectors with identical eigenvalues). Due to the presence of noise, a small deviation of the eigenvalues is allowed. Furthermore, since complex eigenvalues can occur which do not represent an image point, only (close to) real eigenvalues are considered.

Between two degenerate eigenvectors, a line is constructed from which a number of alignment points is selected and saved as described in the previous section. If all three eigenvalues are identical and the eigenvectors are distinct, a number of alignment points over the whole image are created.

From here, the procedure is the same as in the previous section: Already stored APs are checked whether they are still valid, and else deleted. The transformation of a hidden layer is also the same since it does not matter how the APs have been obtained.

5.2.3 Benefits and Drawbacks

With both methods, it is possible to compute the transformation of an invisible layer. By that, a reappeared segment can be recognized if it fits to the layer. This matching is done as described in section 5.1.

The proposed methods enable the framework to track hidden segments based on the transformation of some correlated visible layers. The correlation has to be established before the segment disappears. The second method (section 5.2.1) has the advantage that it does not rely on lines. For that reason, it can cope with non-rectangular segments. Apart from that, it is more elegant to obtain the correlations by taking the transformation matrices into account which capture the movements of the segments quite nicely.

However, the line-based method tends to be more precise since the found lines are exactly the boundaries, given that the errors of the segmentation are low. Obtaining the same lines using the estimated homographies is not that simple. Furthermore, fixed lines are often found even if the segments are not correlated. E.g. if a surface rotates around an axis, the points of the corresponding segment lying on that axis do not move in the image. However, this line may then be recognized as being correlated to the background segment where this line is also constant. More generally spoken, to have no correlation would require the relative homography to have three different eigenvalues or three identical eigenvalues and eigenvectors. In this case, no fixed line exists. However, movements of segments can temporarily be similar. The alignment points that have been created on a wrong line have to be deleted if it turns out that they are not correlated. So, verifying and deleting wrong alignment points is essential for a stable performance. A comparison of results obtained with both methods is given in section 6.3.

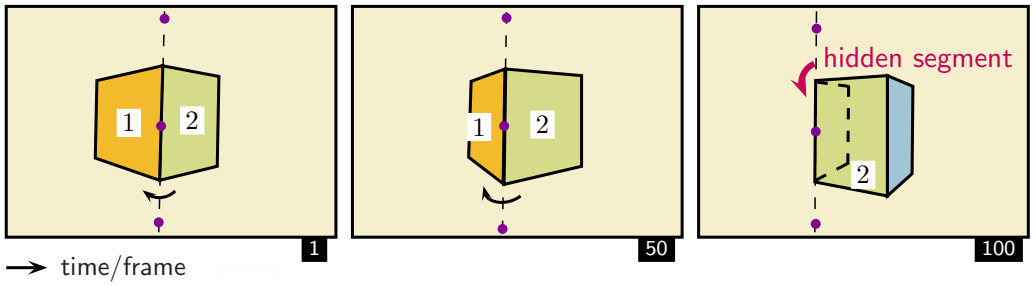


Figure 5.5: The proposed methods for tracking an invisible segment require that at least four alignment points in a general position are available. This is in most cases only possible if two correlated segments are known (and visible). In some situations this is not the case as shown here: Only three points in one line are available.

A more conceptual problem is shown in Fig. 5.5. In order to compute a homography, at least 4 points in a general position are required. Consequently, at least four alignment points are needed that do not lie in one line. This is in most

cases only possible if at least two segments are correlated. An exception is, when two segments have two common lines or when the two segments are correlated over the whole area thus always have the same transformation matrix. (The relative homography has then three identical eigenvalues with distinct eigenvectors). Hence, if not enough alignment points are available, the transformation of the hidden segment cannot be obtained.

In general, it should be possible to derive the hidden segment's position based on only one other segment. Regarding a box like in Fig. 5.5, the shapes of the surfaces on the image plane can be varied by rotating the box. Given the shapes of two neighbored sides, there is no way to rotate the box so that the appearance of one side changes while the other's stays the same. This can only be achieved e.g. by changing camera parameters like "zoom" while moving the box away. By excluding those parameter changes, one might use the results of [47] showing that the relative homographies over an image sequence have certain constraints.

Regarding the introductory example of the shell game, there are cases in which the correlation starts and ends while the corresponding segment is occluded. That kind of cases are also not captured yet (see section 7.2).

Furthermore, chains of correlations are not supported yet but could be added with little effort: Regarding Fig. 5.2, the box may continue the rotation so that segment 2 also vanishes. To obtain the position of hidden segment 1, at first, hidden segment 2 would have to be transformed using its correlation to 3 and 5. Hence, finding the correct order of the computations is the key part of obtaining the positions of all hidden segments. It could be implemented by representing the correlations by a tree.

Chapter 6

Results

The discussed issues of updating/tracking layers, "splitting" and "binding problem" that were discussed in the previous chapters by looking at theoretical examples will now be examined in some complete image sequences. Fig. 6.1 shows an overview.

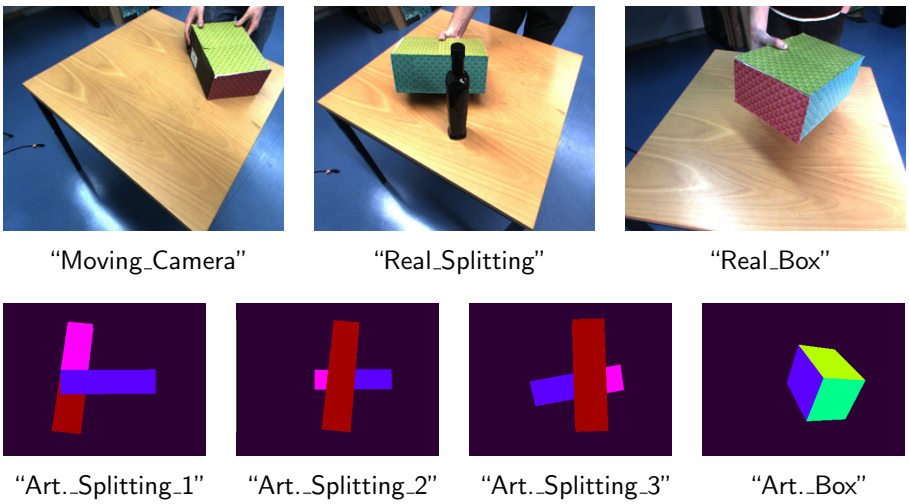


Figure 6.1: Overview of the image sequences used to test the performance of the proposed framework. Real as well as artificial images are used.

The real image sequences were recorded using a 25fps camera in the lab where good lightning circumstances are given. It was taken care of that the velocities are slow enough to gain correct optical flow values. Besides that, only textured objects were used since optical flow algorithms have problems with untextured regions. However, the image segmentation works best with homogeneous areas. For that reason, an artificial texture was created for the box that fulfills both

conditions to a certain degree (the wooden table surface is directly appropriate for that purpose). Note that, in the context of this work, optical flow and image segmentation are "external" technologies.

In order to completely remove any external influence, also some artificial sequences have been created using the open source 3D content creation suite Blender[1]. Creating the scenes using some basic objects, which were assigned different colors, and disabling all mechanisms related to shading allows for obtaining images that can be used as segments so that no external segmentation algorithm needs to be used. Besides that, the speed at each pixel can be stored as a grayscaled image and used in place of optical flow. A detailed description of how Blender has to be set up for this purpose is left out here. It is important to say that the artificial sequences presented in this work can be seen as free of random error. However, some systematic error caused by exporting the data from Blender by saving it as grayscaled images cannot be avoided.

6.1 Keeping Layers Up-To-Date ("Moving_Camera")

The intention of the first presented video is to show how layers are created and updated during the sequence. Fig. 6.2 shows an excerpt of the original images. The scene consists of a table and a box that is held over the table.

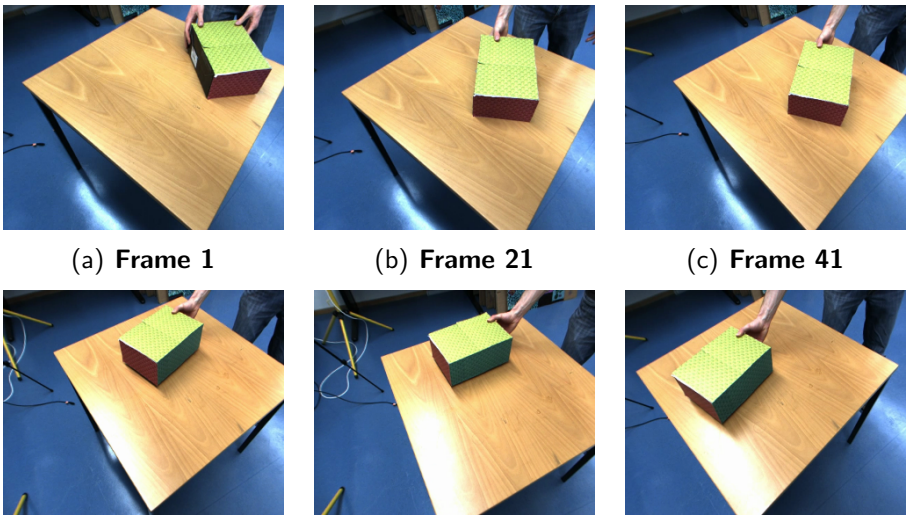
The results are shown in Fig. 6.4. At first, the image segments were obtained by the segmentation algorithm (first column). It can be seen that lots of segments were found that correspond to different objects/parts of objects e.g. the sides of the box or parts of the floor. However, only the table represented by segment number 47 is considered in this example; but note that the described processes are applied to all available segments.

In the first frame, the layer L_{47}, that represents the table, is created. Since there is a foreground object—the box is held over the table—some parts of the surface are occluded. Consequently, the layer only contains the visible parts (first row (frame 1), second column).

As the camera and the foreground object are moving, formerly hidden parts of the table become visible while others become occluded. The framework adds the newly visible parts to the layer as described in section 4.6.1. This completing of the layer over time can be understood by regarding the second column of Fig. 6.4. At frame 84, the layer contains all parts of the table since all of them had been visible for some time.

Note that the layer does not fit directly to the later frames since the camera position changed over time. However, the transformation of the table's appearance in the frame is tracked by the framework so that the layer can always be

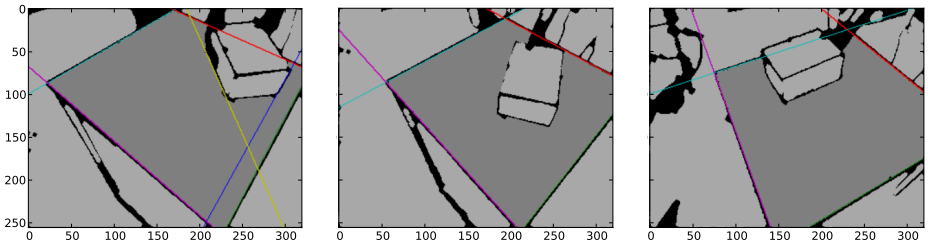
[1] available at http://www.blender.org/



(a) **Frame 1** (b) **Frame 21** (c) **Frame 41**

(d) **Frame 61** (e) **Frame 84** (f) (Frame 104)

Figure 6.2: Original images of sequence "Moving_Camera" (6 of 104 frames are shown). Parts of the table are occluded by the box in the foreground but become visible while camera and box are moving.



(a) **Frame 1** (b) **Frame 41** (c) **Frame 84**

Figure 6.3: Lines of segment 47 (table) in "Moving_Camera" (small selection of frames). The lines (blue and yellow) in frame 1 that border on the box are recognized as "wrong" and ignored in the later frames (see section 4.8.1 for details).

mapped to the current frame. Since the four boundary lines (obtained as described in section 4.4) of the table are visible over the whole sequence (Fig. 6.3), the framework can maintain the "skipping mode" (see section 4.8.2) and yield a good tracking of the table. The mapping of the layer to each frame is shown in the third column of Fig. 6.4. It obviously contains the same data as the layer but matches the perspective of the particular frame.

By mapping the layer to the frame, the framework has a memory of known but currently invisible parts of the table. Note that the table is never completely visible in one frame, there are always parts occluded by the box. However, the layer mapped to the frame gives information about the whole table based on what has been visible before.

6.2 Re-merging Split Segments

The "splitting problem" has been considered in section 5.1. In the current work, the performance of the framework on three artificial sequences and one real image sequence are now presented.

6.2.1 "Artificial_Splitting_1"

The artificial sequence "Artificial_Splitting_1" shows segment 3 moving over 2 so that the latter splits up into 2 and 4. The goal is to recognize that 4 is just a split part of 2 and, accordingly, label it back.

The sequence and the results are shown in Fig. 6.5. In the first frame, layers are created for the two rectangular segments and the background. In the next frames, the segments are tracked so that the layers can be mapped to the frames. Since those mappings are erroneous, the covariances of the estimated transformations are computed as described in section 4.10. Fig. 6.6 shows the expected deviations for some frames. For the frames 1-11, all boundary lines of segment 2 are visible so that the framework can compute its transformation in "skipping mode". Hence, the expected deviation of the transformed pixels of the layer keeps constant around ± 1.

However, the splitting of segment 2 in frame 12 ends the "skipping mode" since only three lines remain. Consequently, from this frame on, the transformation is computed by also taking optical flow into account. The area of low deviation reduces to the still visible parts of the segment (see Fig. 6.6, frame 12). Thus, when mapping the whole layer 2 to the frame, the hidden part's pixels are transformed with a relatively high expected deviation.

The reason for mapping the layer to the frame is to obtain the region where the split part (in this case given by segment 4) can be found. As described in

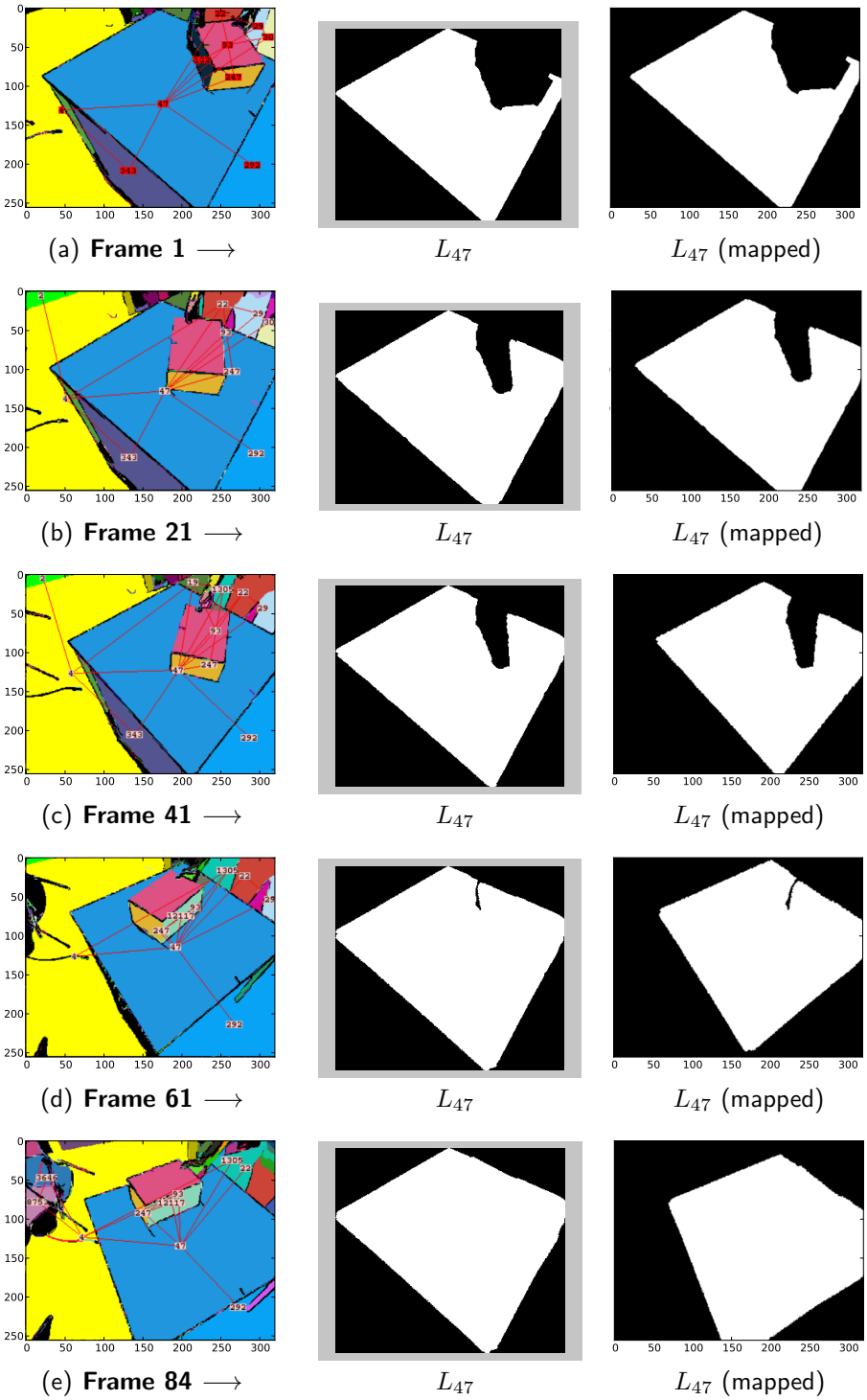


Figure 6.4: Results of sequence "Moving_Camera". The framework completes the layer of the table segment 47 when formerly occluded parts become visible.

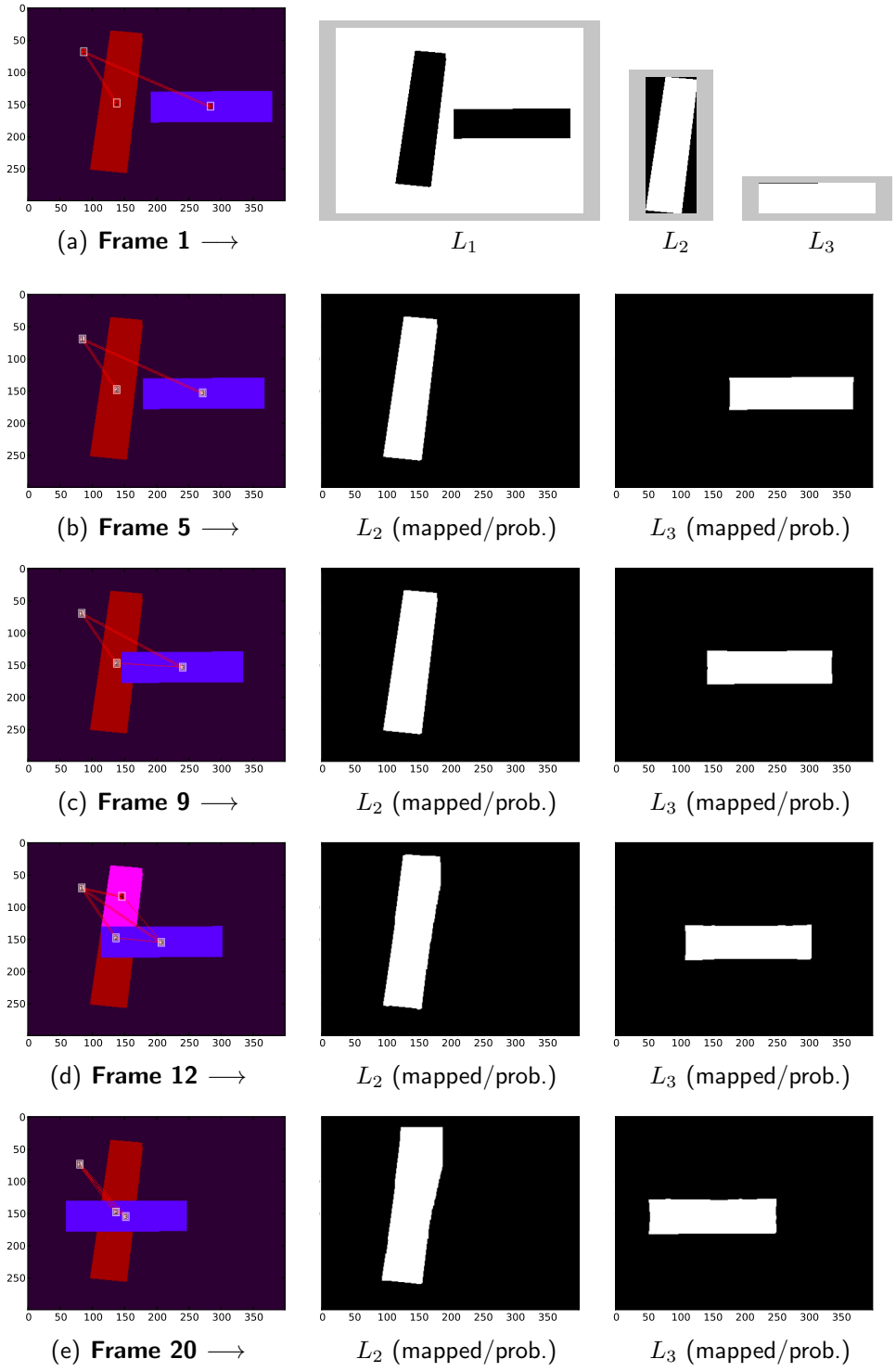
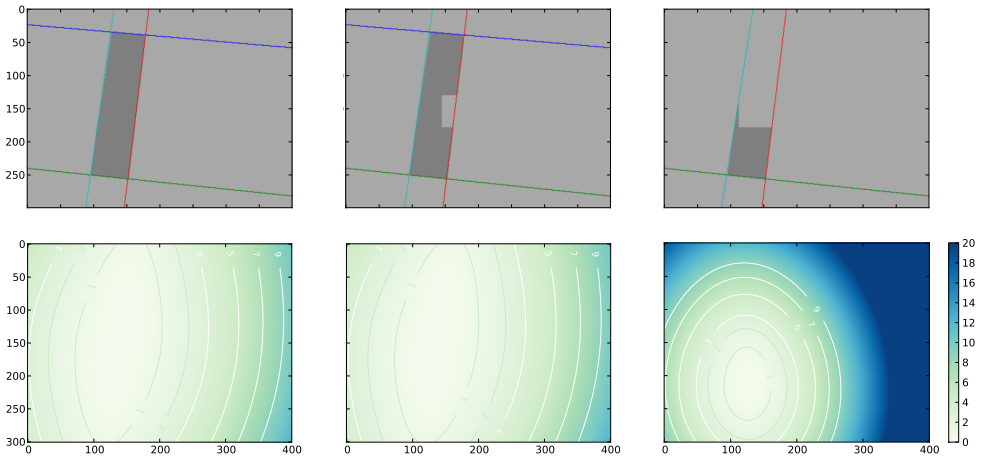


Figure 6.5: Results of sequence "Artificial_Splitting_1". Segment 2 splits up into 2 and 4 in frame 12, the framework labels it back later when the splitting is recognized.



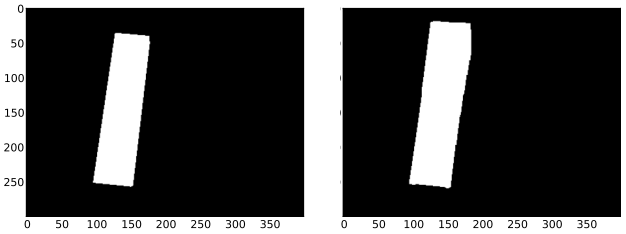
(a) **Frame 5** (b) **Frame 9** (c) **Frame 12**

Figure 6.6: Development of the expected deviations $\sqrt{\sigma_x^2 + \sigma_y^2}$ of segment 2 (second row) in "Artificial_Splitting_1". As long as its boundary lines are completely visible, the expected deviation is low (ca. ± 1). When the splitting occures in frame 12, the deviation in the hidden region increases as the framework switchs back to "successive mode" since only three lines are available (see first row).

section 5.1, the deviation estimate is used to increase this region to make sure that the split segment matches. Fig. 6.7 shows, for frame 12, a comparison between the normal mapping of the layer and the mapping that takes the deviation into account. It is easy to see that the latter enlarges the layer in the hidden parts.

From Fig. 6.5 can be seen that segment 4 is correctly labeled back to 2 for the rest of the sequence. This relabeling takes places in frame 13, one frame after the new segment was found in frame 12. As described in section 5.1, the relabeling is done after checking whether the movements of 2 and 4 match. For that reason, at least one further frame after the actual splitting has to be awaited until a confident decision can be made by the framework. Note that 4 could also be wrongly matched with the background 1 as its layer also covers the position of 4. However, since 1 does not move contrary to 4, they are not mismatched by the framework.

Last thing to mention is that the depth relations of the segments are established as "3 above 1" (in frame 3) and "3 above 2" (in frame 8). This is done by analyzing the segment's movements as described in section 4.7. However, this information is not needed here since no "wrong lines" (see section 4.8.1) are present for segment 2 or 3. (For the background segment 1, there are wrong lines but a presentation of results for segment 1 is left out here since nothing special



(a) L_2 (mapped) (b) L_2 (mapped/prob.)

Figure 6.7: Mapping of L_2 to **Frame 12**. (a) shows the normal mapping. In (b) the expected deviation is taken into account – the area is expanded by 2σ as described in section 5.1.

could be seen.)

6.2.2 "Artificial_Splitting_2"

The sequence "Artificial_Splitting_2" is similar to the previous sequence. The difference is that segment 3 moves behind 2 and is split up into 3 and 4 when it comes out at the other side of 2. The results are shown in Fig. 6.8. The framework correctly labels 4 back to 3. The depth relations of the segments are established to "3 above 1" (in frame 3), "2 above 3" (in frame 5), and "2 above 1" (in frame 5) where the latter is obtained by logic (see section 4.7.3).

6.2.3 "Artificial_Splitting_3"

The sequence "Artificial_Splitting_3" involves a rotation of the split segment but is otherwise similar to the previous sequence. The results are given in Fig. 6.9.

Fig. 6.10 shows the lines found for segment 3. The line at the border to segment 2 in frame 10 is removed since 2 is in the foreground: In frame 6, where 2 and 3 start to touch, the bordering line is removed according to section 4.8.1, "Touching of Segments". In the following frames, their depth relation is recognized as "2 above 3" (in frame 8) so that this line is always ignored (see 4.8.1, "Occlusion of Segments"). The other depth relations are established as "3 above 1" (in frame 3) and "2 above 1" (in frame 8 by logic).

6.2.4 "Real_Splitting"

After the previous artificial examples, the framework's resolving the "splitting problem" is now shown on a real image sequence. A selection of the original

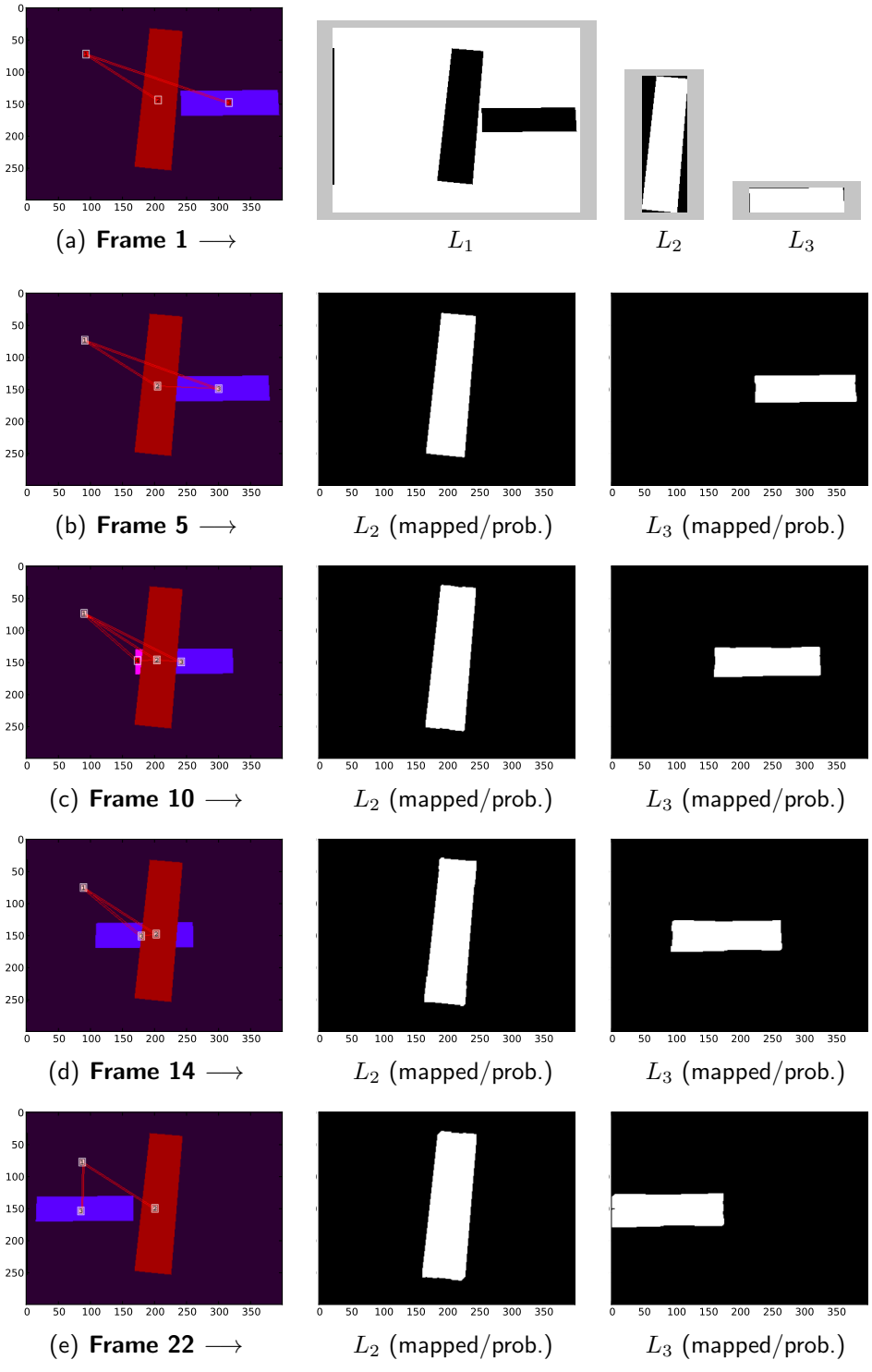


Figure 6.8: Results of sequence "Artificial_Splitting_2". Segment 3 splits up into 3 and 4 in frame 10 when is comes out on the other side of 2. The framework labels it back later when the splitting is recognized.

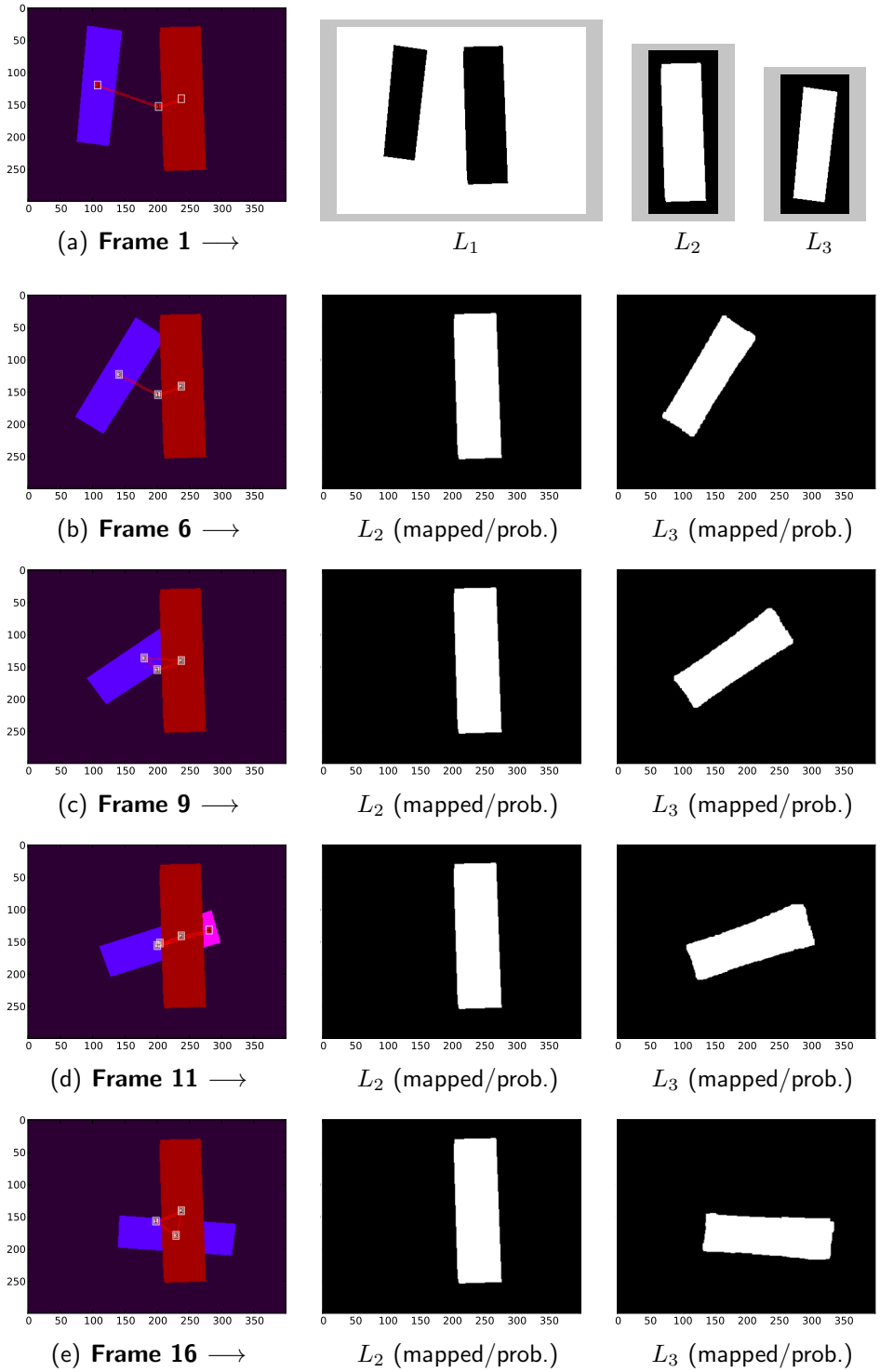
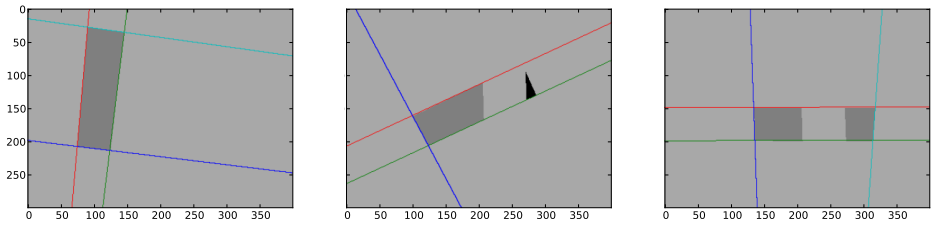


Figure 6.9: Results of sequence "Artificial_Splitting_3". Segment 3 splits up into 3 and 4 in frame 11 when is comes out on the other side of 2. The framework labels it back later when the splitting is recognized.



(a) **Frame 1** (b) **Frame 10** (c) **Frame 14**

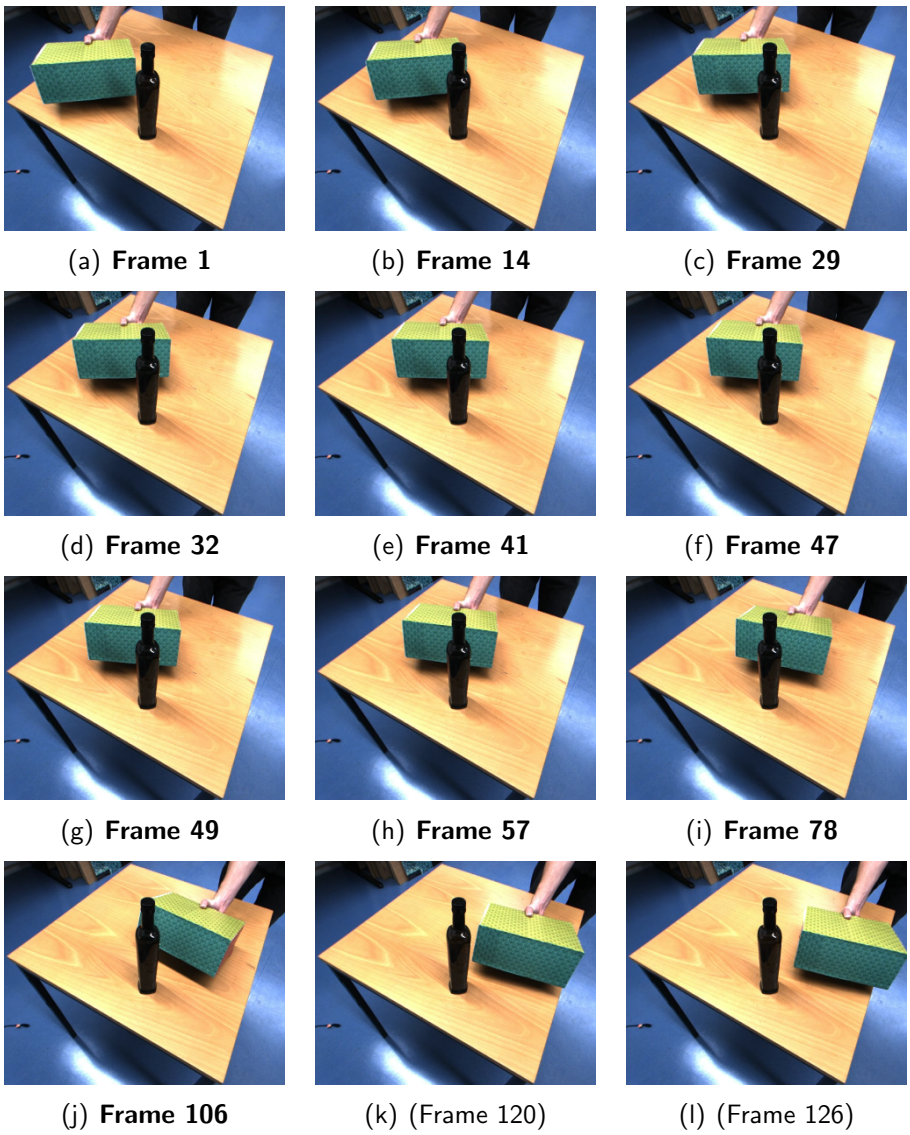
Figure 6.10: Lines of L_3 in "Artificial_Splitting_3". The line bordering on the foreground segment 2 in frame 10 is removed, as well as in all other frames where 2 and 3 have the *Touching* relation.

images is shown in Fig. 6.11. The scene contains a box that is moved behind a bottle. Regarding the segmentation in Fig. 6.13 it can be seen that the segments 91 and 167 belonging to the box (front and top side) are split by the bottle (107): the new segments 308 and 484 appear in frame 32 respectively 47. Furthermore, a part of the table (segment 36) is, from frame 29 on, represented by a second segment 560, caused by the arm. The arm itself consists of many too small segments which are ignored/removed by the framework. Since small segments are often caused by noise and their homography estimation is poor in most cases, the framework requires a segment to have 300 pixels as a minimum size.

As described, in this single sequence the "splitting problem" occures three times and the framework is able to resolve all of them. For an easier understanding, Fig. 6.12 shows the important frames scaled-up. It can be seen that that segment 308 is labeled back to 167 in frame 41, 560 back to 36 in frame 49 and 484 to 92 in frame 57.

Fig. 6.13 shows the layers of the sequence and their mapping to the frames. For segment 167 the boundary lines are shown in the fifth column. It can be seen that the right line which borders on the bottle (107) is present only in the first, but not in the later frames, because the framework recognizes it as a "wrong line". Since both segments are already touching in the first frame, the line's incorrectness is recognized by RANSAC (section 4.8.1, "RANSAC") and not by the method "Touching of Segments" (also in section 4.8.1).

The depth relations of the main acting segments are recognized as "92 above 36", "167 above 36", "107 above 92" and "107 above 167", all in frame 9. By logic, "107 above 36" is derived. Furthermore, "92 above 2", "167 above 2" and "18 above 23" is obtained, where 2 and 23 are parts of the floor and 18 is a part of the person holding the box. Since the movements in this sequence are very slow (compared to those in the artificial ones), the depth relations are checked



(a) **Frame 1** (b) **Frame 14** (c) **Frame 29**

(d) **Frame 32** (e) **Frame 41** (f) **Frame 47**

(g) **Frame 49** (h) **Frame 57** (i) **Frame 78**

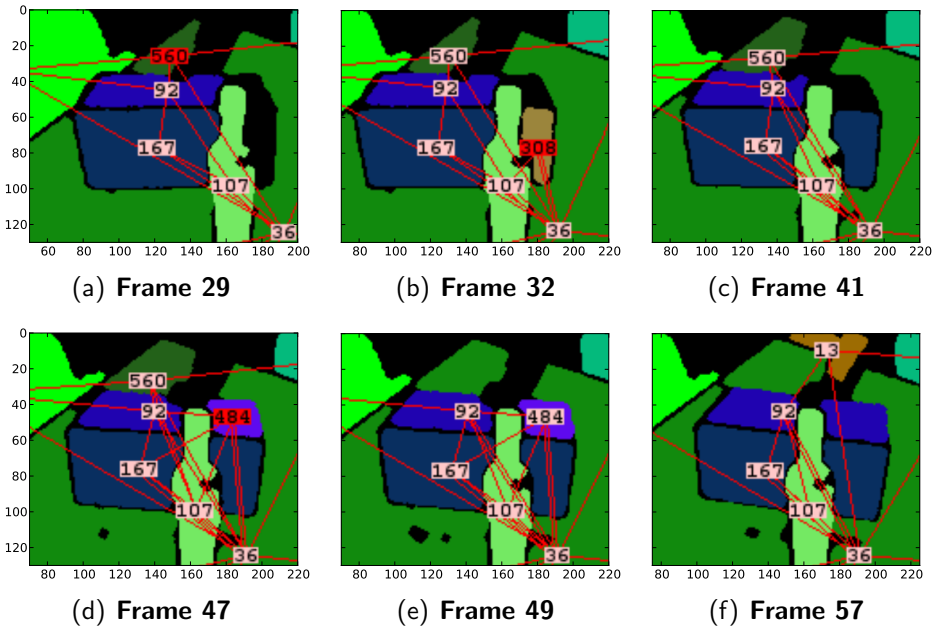
(j) **Frame 106** (k) (Frame 120) (l) (Frame 126)

Figure 6.11: Original images of sequence "Real_Splitting" (12 of 126 frames are shown). The box is moved behind the bottle.

only every 8-th frame. That is why the relations are recognized in frame 9.

The last column in Fig. 6.13 shows the development of the covariances of layer 167. As it moves behind the bottle the region with low deviation shrinks, and becomes larger from frame 46 on, when its part on the right side is re-merged.

The expected covariance of 92 also increases when it moves behind 107. For that reason, its mapping (second column in Fig. 6.13) grows a lot on the right



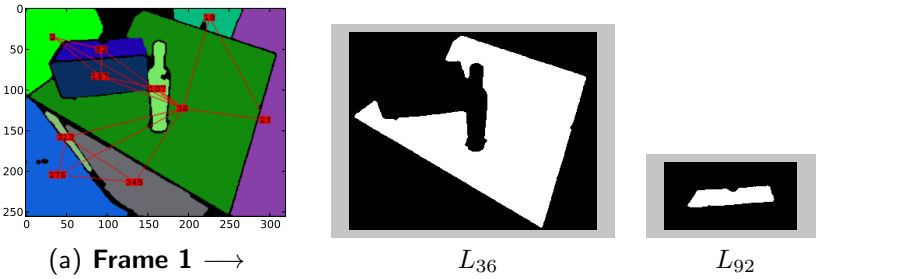
(a) **Frame 29** (b) **Frame 32** (c) **Frame 41**

(d) **Frame 47** (e) **Frame 49** (f) **Frame 57**

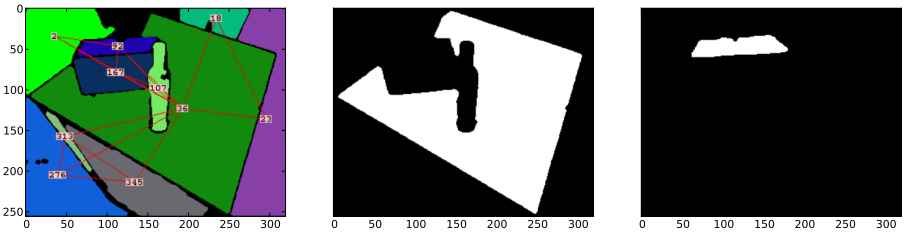
Figure 6.12: Segments of sequence "Real Splitting" scaled up. One part of the table, given by segment 36, is represented by the extra segment 560. The framework relabels it in frame 49. The same happens to 308 in frame 41 and 484 in 57.

side. Since 92 is a relatively small segment, the deformation is large. It can be said that the size of segment 92 defines the minimum size for the framework to perform well.

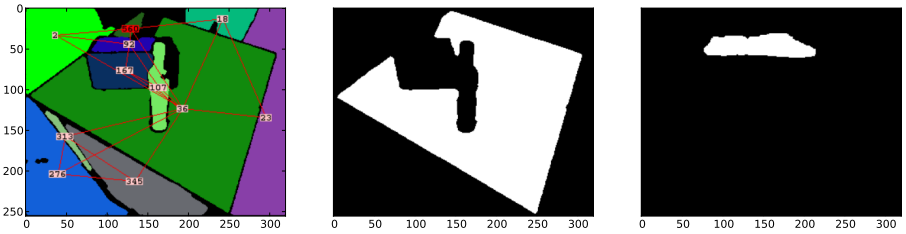
In frame 57, a transformation error occurs because the optical flow data seems not to match well enough with the line data. In such a case, a new layer is created and the covariance (last column) is also reset. Although errors like that are obviously not desired, they are only critical when they occur together with a "splitting problem" which eventually cannot be resolved in that case. When a layer is recreated, the data of the last frame is copied assuming that the movement to the current frame is little. By that, loss of data is minimized since hidden parts of the layer are preserved. However, such a problem did not occur in the tested sequences.



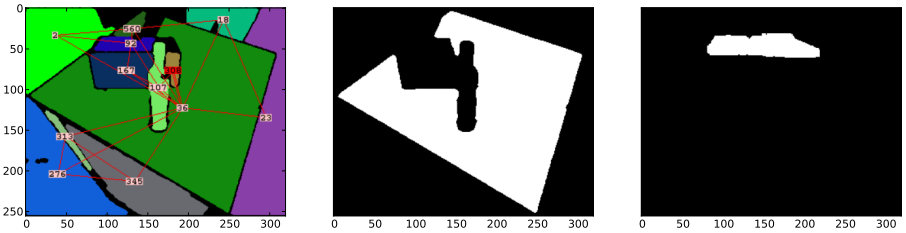
(a) **Frame 1** $\longrightarrow$ L_{36} L_{92}



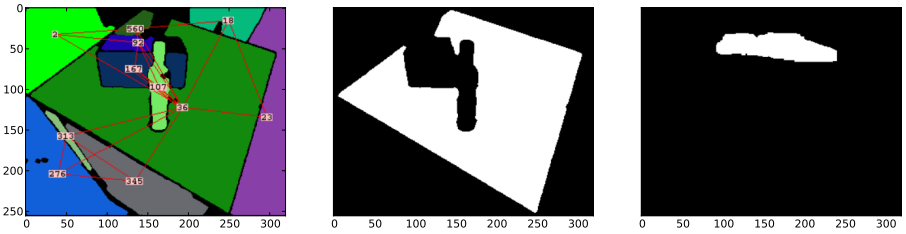
(b) **Frame 14** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(c) **Frame 29** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(d) **Frame 32** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(e) **Frame 41** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)

Figure 6.13: Results of sequence "Real_Splitting". The segments 36, 92 and 167 are split into two segments due to occlusion. The framework recognizes this and labels them back.

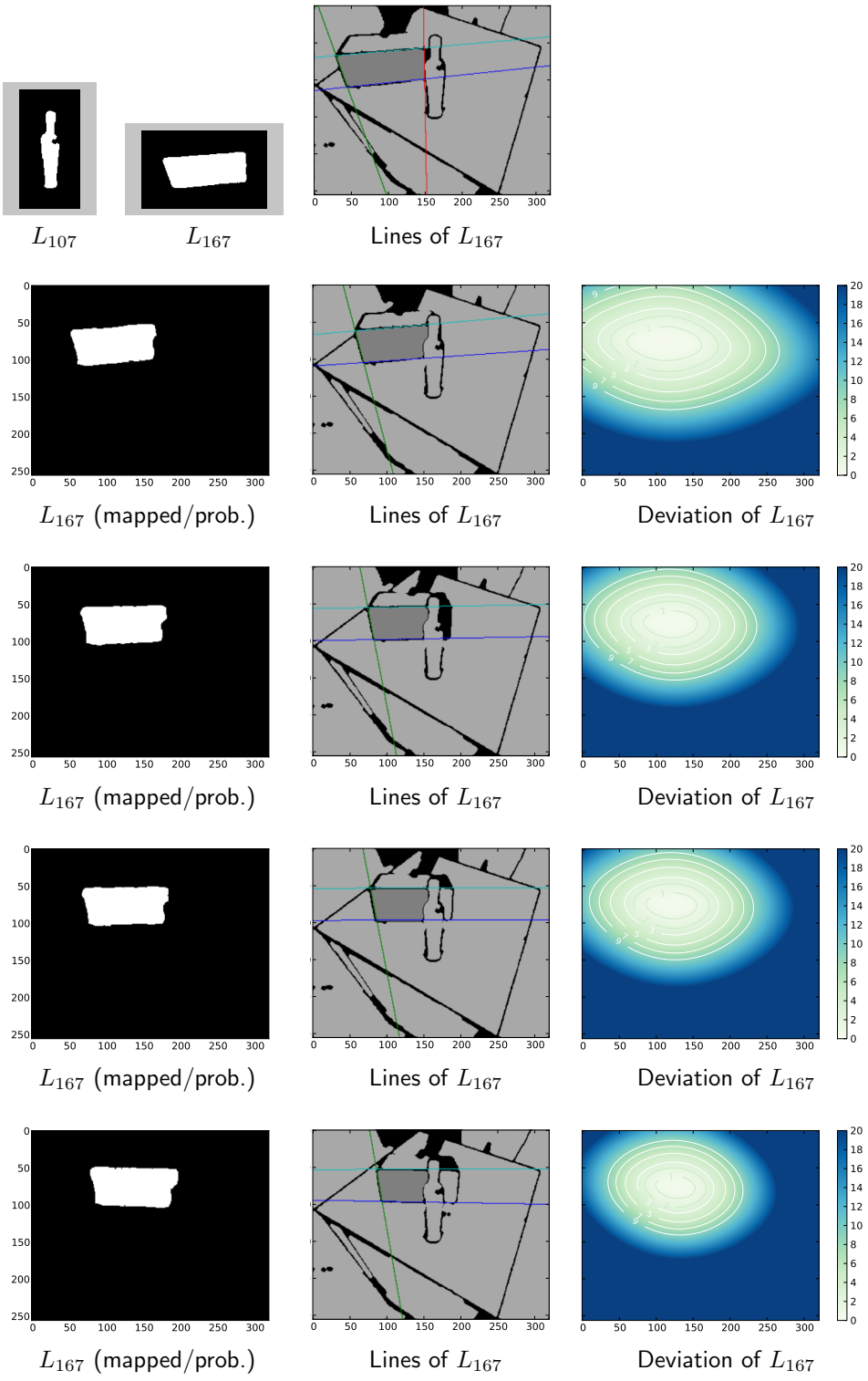
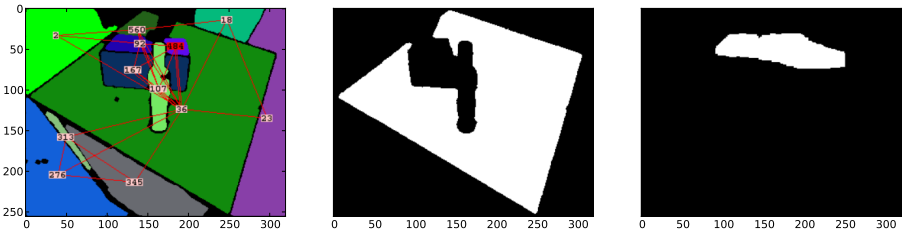
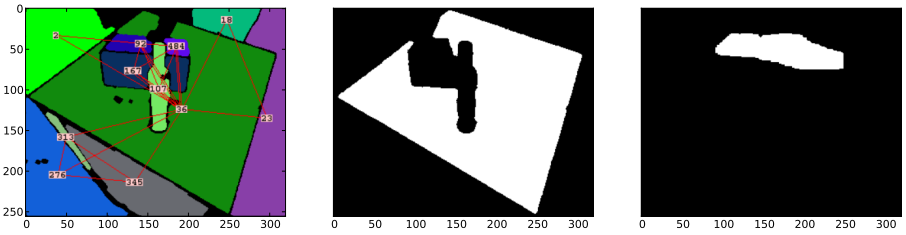


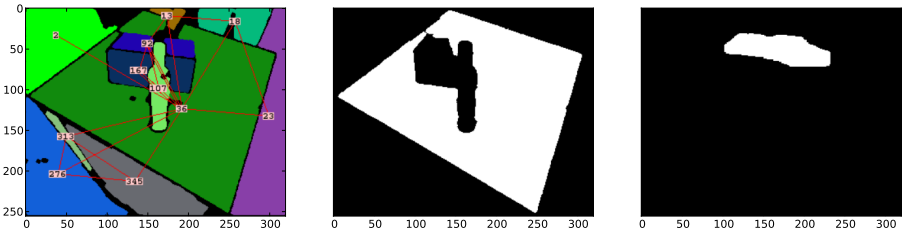
Figure 6.13 (continuation from left page)



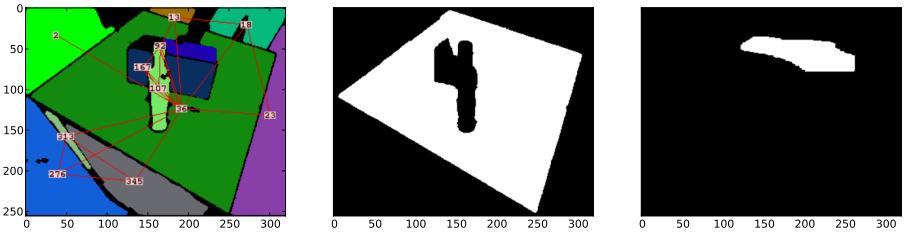
(f) **Frame 47** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



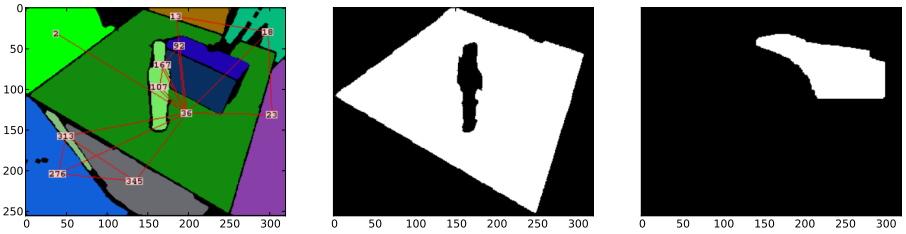
(g) **Frame 49** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(h) **Frame 57** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(i) **Frame 78** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)



(j) **Frame 106** $\longrightarrow$ L_{36} (mapped/prob.) L_{92} (mapped/prob.)

Figure 6.13 (continuation): Results of sequence "Real_Splitting". The segments 36, 92 and 167 are split into two segments due to occlusion. The framework recognizes this and labels them back.

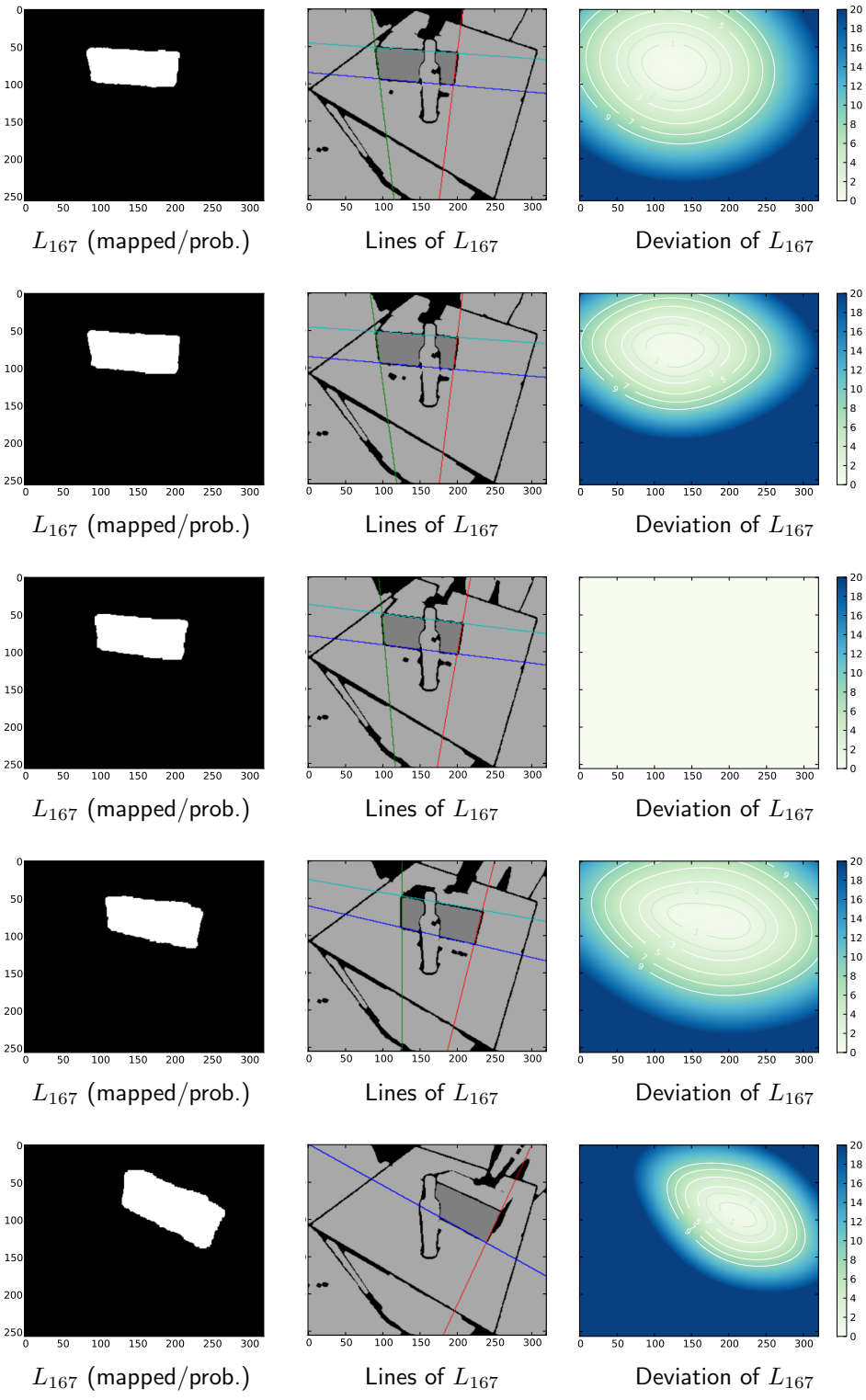


Figure 6.13 (continuation from left page)

6.3 Tracking Hidden Segments

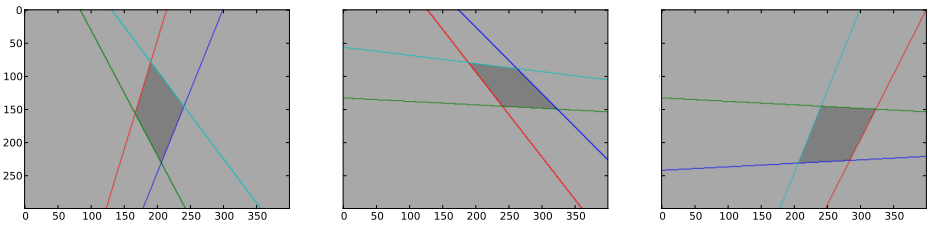
In section 5.2 it was described how the framework finds correlations of segments in order to track hidden segments. This was called the "binding problem". In this section, the performance of the framework on an artificial sequence and a real image sequence are presented.

6.3.1 "Artificial_Box"

The sequence "Artificial_Box" shows a box. Its initially visible surfaces have the segment numbers 3, 5 and 6 as can be seen in Fig. 3.15 (continuation). In frame 6, segment 3 becomes occluded as the box rotates. Furthermore, in a later frame, a new segment with number 2 appears. However, when the box rotates back the surface that was formerly represented by segment 3 reappears as segment 4. The goal is to relabel 4 back to 3 which is achieved in frame 20 with both methods proposed in sections 5.2.1 and 5.2.2.

Line-based Method

As described in section 5.2.1, the first method for determining the correlations of segment pairs is based on boundary lines which are shown in Fig. 6.14. First,



(a) Segment 3 (b) Segment 5 (c) Segment 6

Figure 6.14: Lines of segments 3, 5 and 6 in **Frame 4** that are used by the line-based method to create Alignment Points for segment pairs.

the common line of each segment pair has to be established, then the Alignment Points (AP) are created on that line. Fig. 6.16 shows those points for frames 4 and 19. Common APs have the same color. The error bars indicate their expected deviation during the transformation. Since each layer has its own transformation (and consequently its own error estimate), common APs have different error estimates. In frame 19 where layer 3 is still invisible (it was not relabeled yet), the APs are used to compute its transformation.

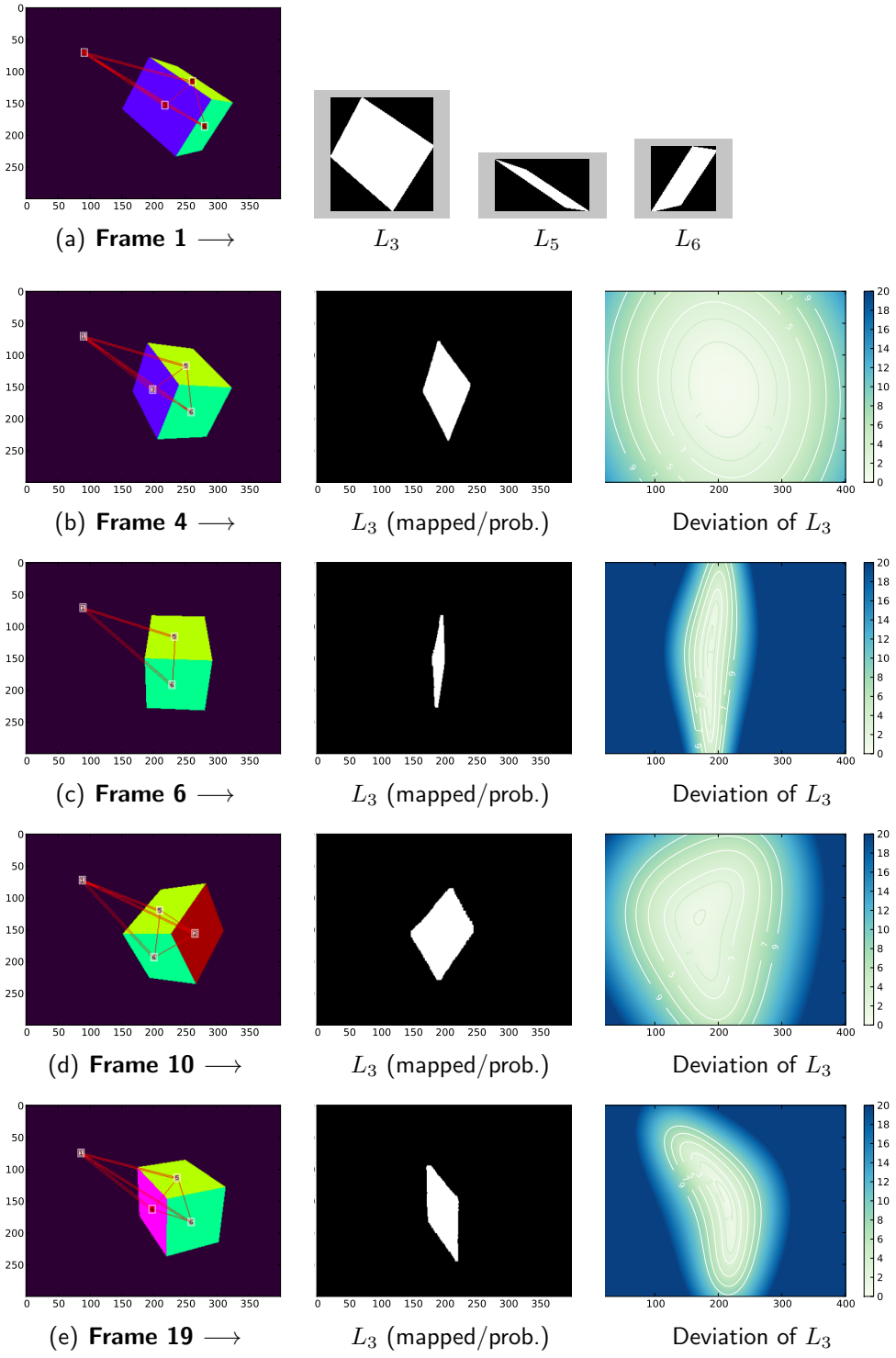
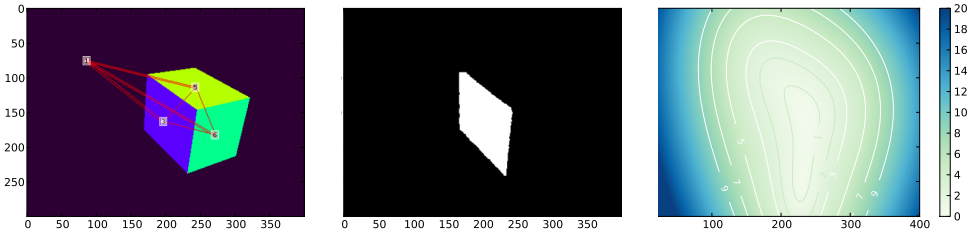


Figure 6.15: Results for sequence "Artificial_Box". The side of the box represented by segment 3 vanishes and reappears as 4. The framework recognizes this and labels it back to 3 in frame 20.



(f) **Frame 20** $\longrightarrow$ L_3 (mapped/prob.) Deviation of L_3

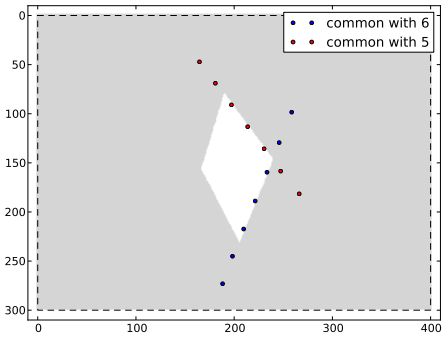
Figure 6.15 (continuation): Results for sequence "Artificial_Box". The side of the box represented by segment 3 vanishes and reappears as 4. The framework recognizes this and labels it back to 3 in frame 20.

Regarding the expected deviation of the pixels (last column in Fig. 3.15 (continuation), frame 19), it can be seen that the area of low deviation ranges around the borders to segment 5 and 6, exactly where the alignment points are located.

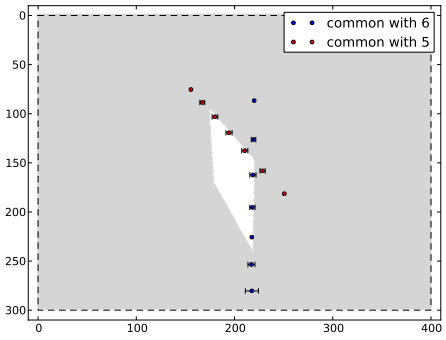
Homography-based Method

The second method completely relies on the homography transformations as described in section 5.2.2. The APs obtained by that method are shown in Fig. 6.17 for frame 4 and 19. It can be seen, that some APs are being deleted in the shown frames since they were recognized as wrong. Furthermore, at frame 4, no APs between 5 and 6 have been found yet since their common boundary have moved relatively few.

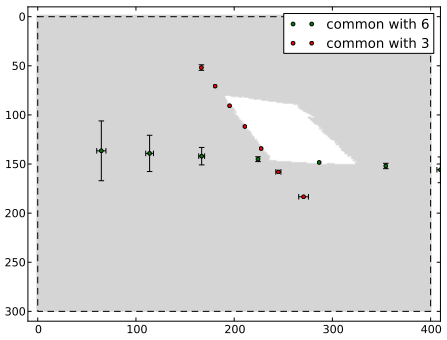
These observations show that the second method is able to establish good alignment points without using lines. The found points in Fig. 6.17 lie on the segment's borders which can be understood as ground truth in this case. However, as the method takes the transformation, hence the movements of the segments into account, the APs cannot be established until a large movement is observed. Comparing Fig. 6.16(b,c) and Fig. 6.17(b,c), it can be seen that the homography-based method, contrary to the line-based method, did not recognize the correlation between the segments 5 and 6 until frame 4. However, as the box moves further, the correlation is recognized as can be seen in Fig. 6.17(e,f) where corresponding APs are present. Note, that this is just a further comparison between both methods, but keep in mind that a correlation between those segments is not needed to solve the binding problem involved with segment 3 which occurs in this sequence.



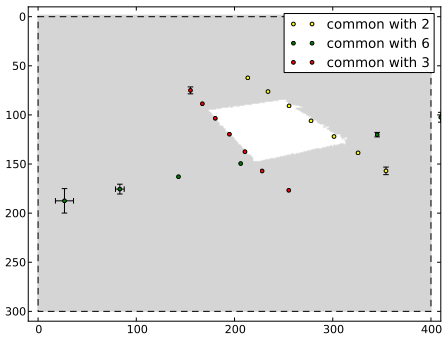
(a) Alignment Points of L_3



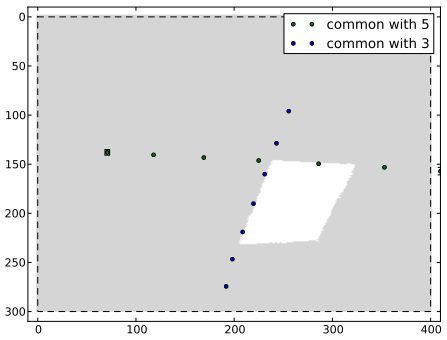
(d) Alignment Points of L_3



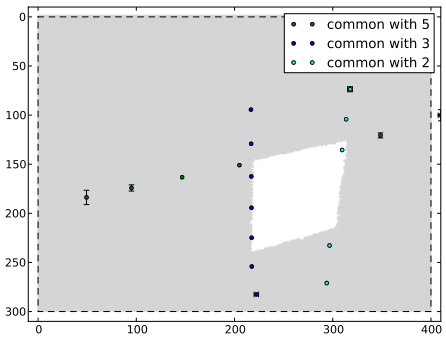
(b) Alignment Points of L_5



(e) Alignment Points of L_5

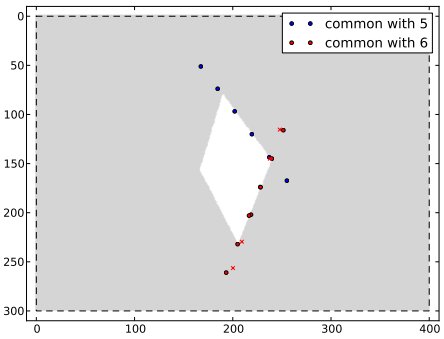


(c) Alignment Points of L_6

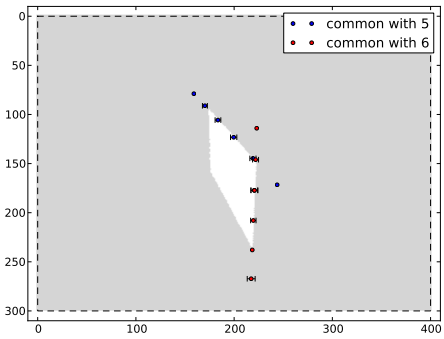


(f) Alignment Points of L_6

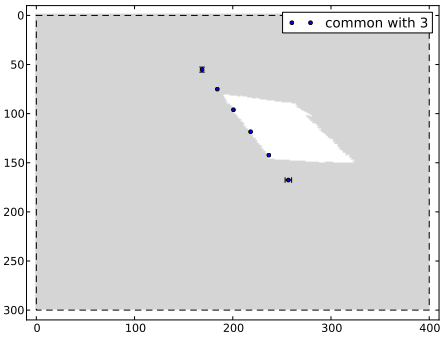
Figure 6.16: Result of the line-based method. Alignment Points of L_3, L_5 and L_6 in **Frame 4** (a-c) and **Frame 19** (d-f) of sequence "Artificial_Box".



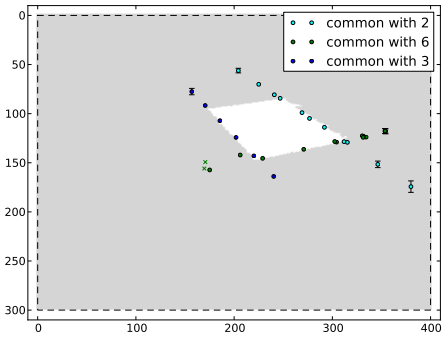
(a) Alignment Points of L_3



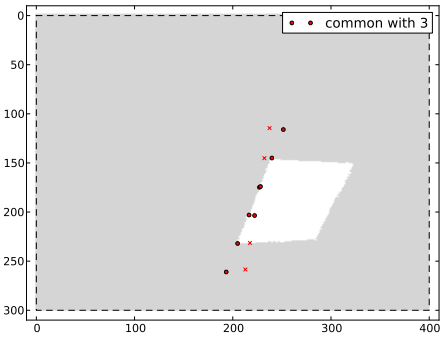
(d) Alignment Points of L_3



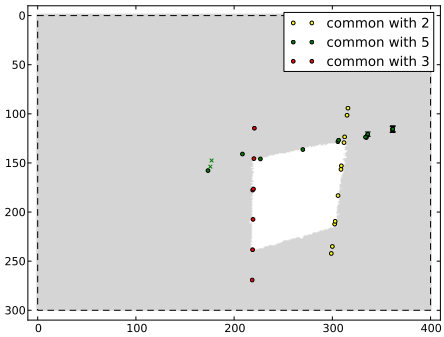
(b) Alignment Points of L_5



(e) Alignment Points of L_5



(c) Alignment Points of L_6

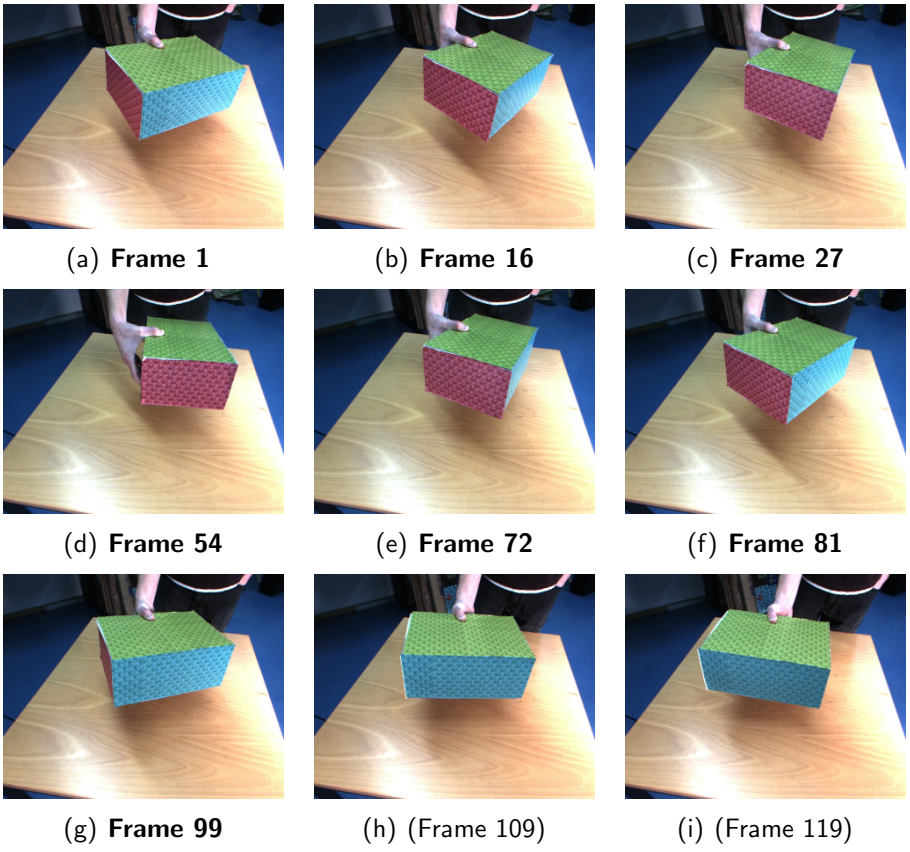


(f) Alignment Points of L_6

Figure 6.17: Result of the homography-based method. Alignment Points of L_3, L_5 and L_6 in **Frame 4** (a-c) and **Frame 19** (d-f) of sequence "Artificial_Box". Points displayed as "$\times$" have just been recognized as wrong and deleted.

6.3.2 "Real_Box"

The same box which was used to create the sequence "Real_Splitting" is now rotated just like the artificial box in the previous section. A selection of the frames can be seen in Fig. 6.18.



(a) **Frame 1** (b) **Frame 16** (c) **Frame 27**

(d) **Frame 54** (e) **Frame 72** (f) **Frame 81**

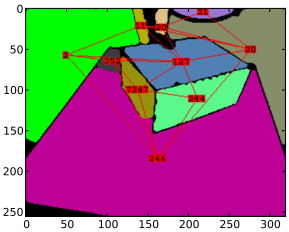
(g) **Frame 99** (h) (Frame 109) (i) (Frame 119)

Figure 6.18: Original images of Sequence "Real_Box" (9 of 119 frames are shown). The box is rotated so that the front side vanishes (in frame 27) and reappears (in frame 72).

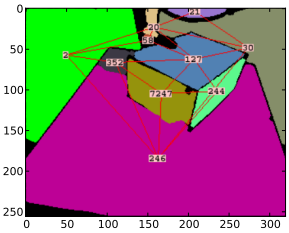
The segmentation and the results are shown in Fig. 3.19 (continuation). The front side of the box is represented by segment 244. In frame 27, is becomes occluded as the box rotates. In frame 72, it reappears as segment 9342.

Line-based Method

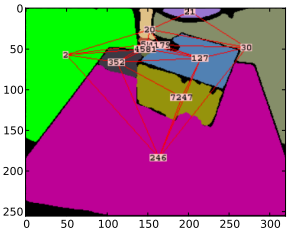
Using the line-based method, the segment is correctly labeled back to 244 in frame 81. Fig. 6.20 shows the lines found in frame 16. The APs of the box's



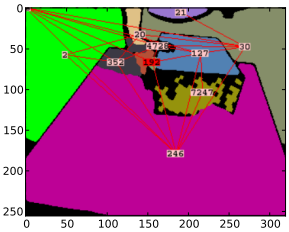
(a) **Frame 1** $\longrightarrow$ L_{7247} L_{127}



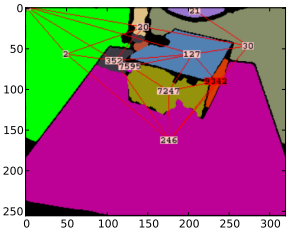
(b) **Frame 16** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)



(c) **Frame 27** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)



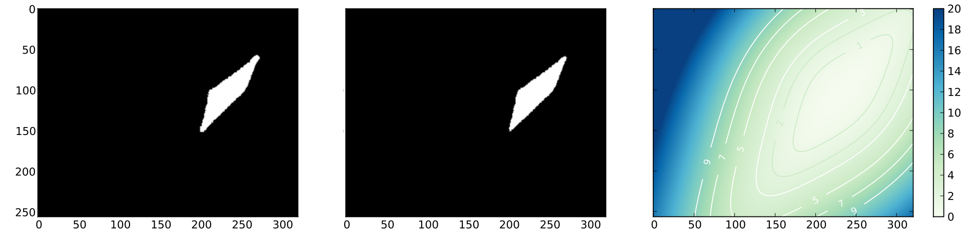
(d) **Frame 54** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)



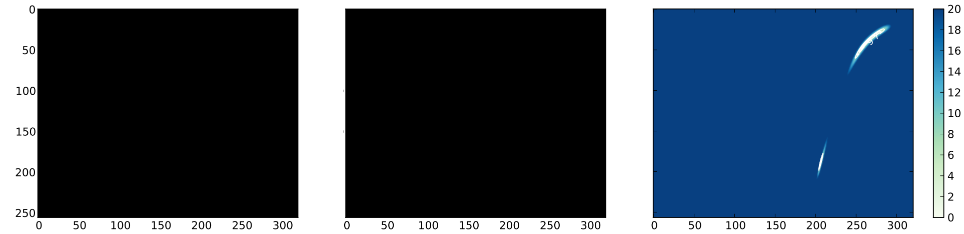
(e) **Frame 72** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)

Figure 6.19: Results of sequence "Real_Box". Segment 244 becomes occluded and reappears as 7247 during the roation of the box. The framework relabels it back correctly.

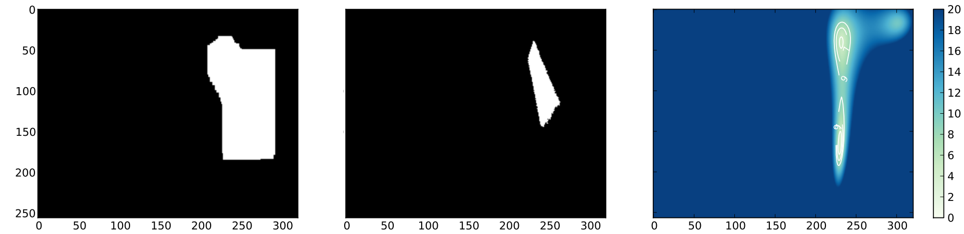
L_{244}



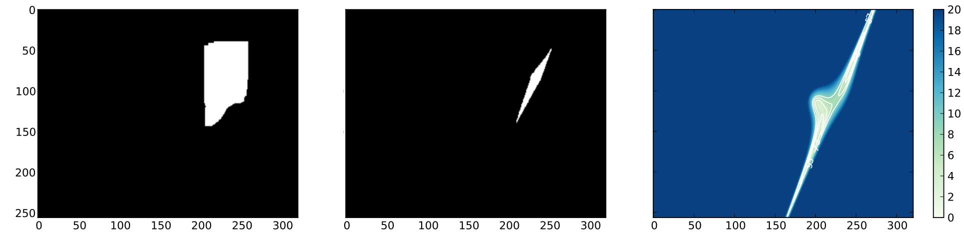
L_{244} (mapped/prob.) L_{244} (mapped) Covariance of L_{244}



L_{244} (mapped/prob.) L_{244} (mapped) Covariance of L_{244}

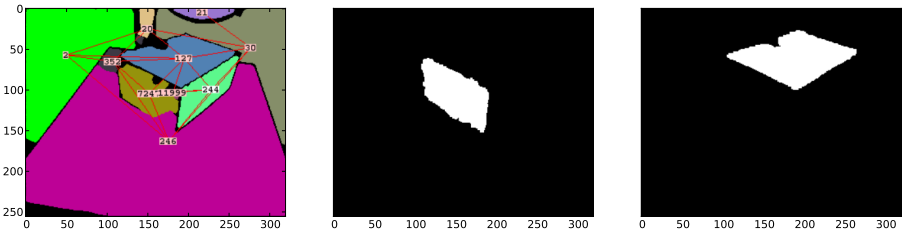


L_{244} (mapped/prob.) L_{244} (mapped) Covariance of L_{244}

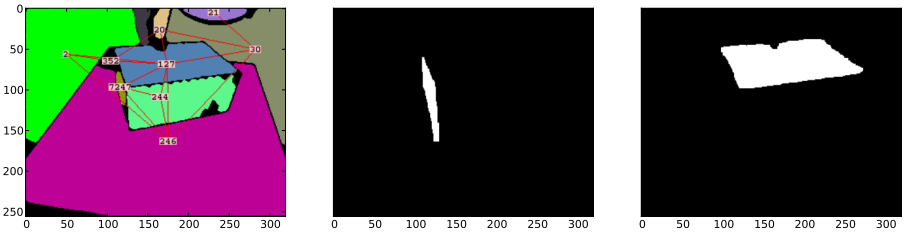


L_{244} (mapped/prob.) L_{244} (mapped) Covariance of L_{244}

Figure 6.19 (continuation from left page)



(f) **Frame 81** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)



(g) **Frame 99** $\longrightarrow$ L_{7247} (mapped/prob.) L_{127} (mapped/prob.)

Figure 6.19 (continuation): Results of sequence "Real_Box". Segment 244 becomes occluded and reappears as 7247 during the roation of the box. The framework relabels it back correctly.

segments are shown in Fig. 6.21. The expected deviations of the layer pixels of layer 244 are shown in the last column of Fig. 3.19 (continuation). In the frames where the segment is invisible, the covariance is very high (frames 27-72). The reason for that is that the alignment points are almost in one line in this position of the box. However, when the box rotates back, the accuracy of the transformation improves (frame 81). The fourth column shows the layer mapped to the frames by taking the deviation into account, while the fifth row shows the ordinary transformation (analog to Fig. 6.7). It can be seen by comparing it to the segment (first row) that the latter is fine for all frames. However, the former has a strong distortion which shows that the expected deviation is estimated very pessimistically. This issue could be resolved by adding more APs since the number of points used to compute a transformation affects the precision. However, it is better to over-estimate the errors and obtain a larger region where the occluded segment can be found, than under-estimating the deviations. Problems might arise when several hidden segments have to be matched and some of them fall into the same range. On the other hand, since the segment's movements are taken into account when matching a hidden segment (see section 5.2.2), mismatches caused by this should not occur.

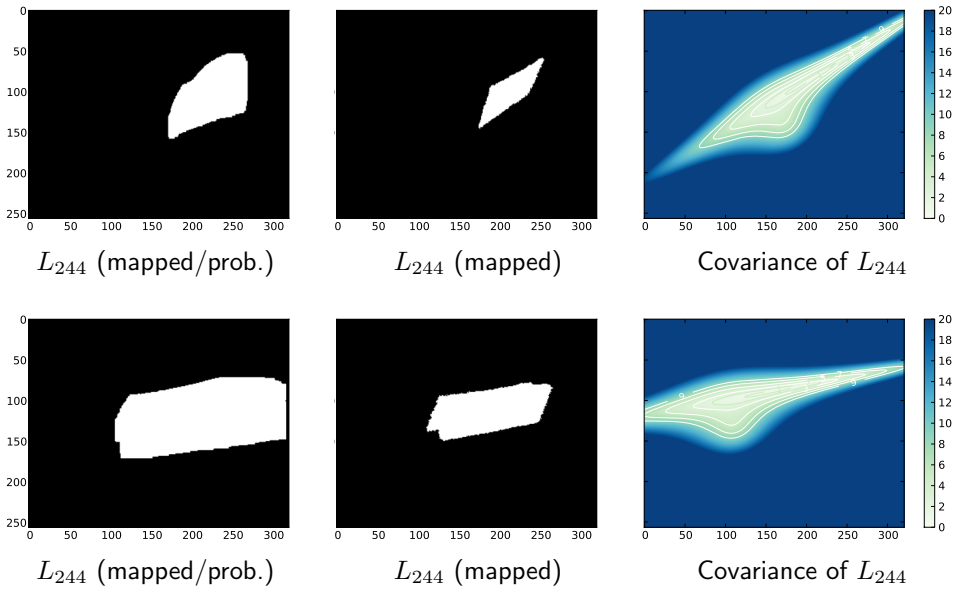
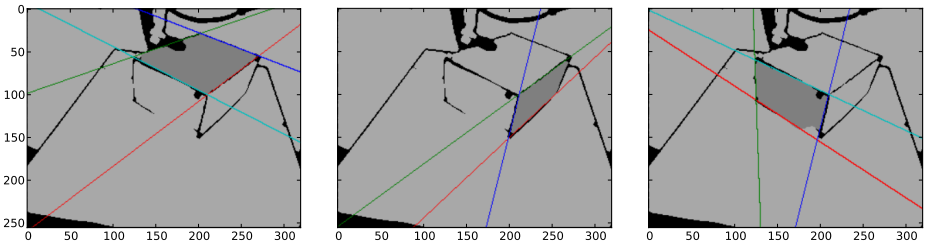


Figure 6.19 (continuation from left page)

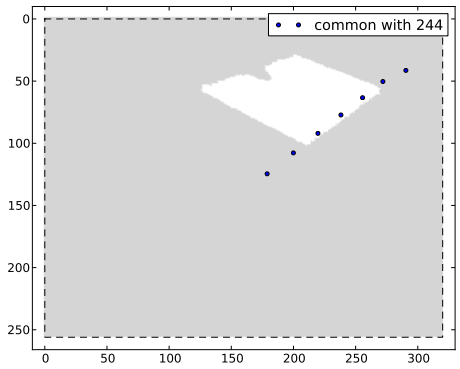
Homography-based Method

The APs created by the homography-based method are shown in Fig. 6.22 for frame 16. It can be seen that points have been created for the segment pair $(127, 244)$, but not for $(244, 7247)$. Hence, no correlation between 244 and 7247 could be established before 244 disappears. As a consequence, it cannot be tracked while it is hidden. The reason why the algorithm fails in this sequence is that segment 7247 seems to be quite noisy. Also note, that, due to the unstable boundary, the line-based method did not recognize the common line between

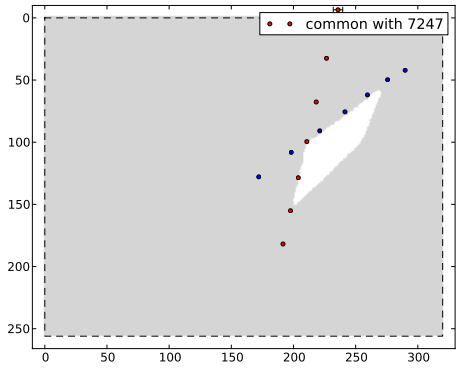


(a) Segment 127 (b) Segment 244 (c) Segment 7247

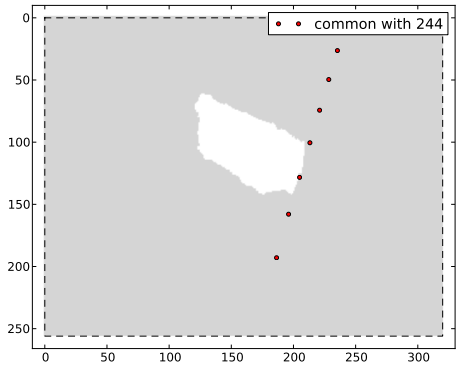
Figure 6.20: Lines of segments 127, 244 and 7247 in **Frame 16** that are used by the line-based method to create Alignment Points for segment pairs.



(a) Alignment Points of L_{127}

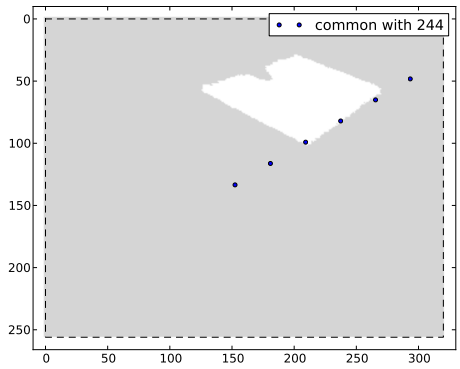


(b) Alignment Points of L_{244}

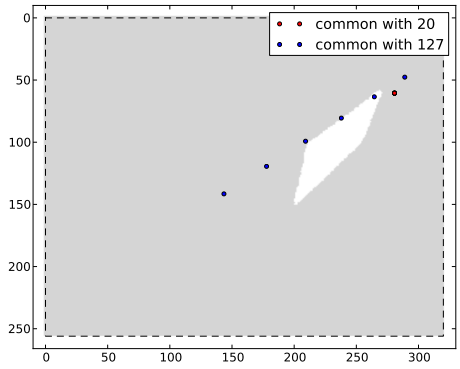


(c) Alignment Points of L_{7247}

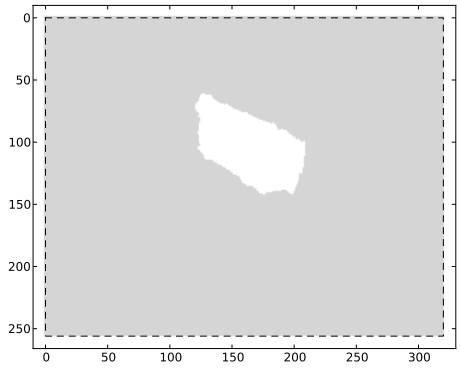
Figure 6.21: Alignment Points in **Frame 16** obtained with the line-based method.



(a) Alignment Points of L_{127}



(b) Alignment Points of L_{244}



(c) Alignment Points of L_{7247}

Figure 6.22: Alignment Points in **Frame 16** obtained with the homography-based method.

127 and 7247 so that no common APs could be created neither in Fig. 6.21. (This, however, was not a problem in the previous section since they are not needed as both segments do not vanish during the sequence.) As a consequence, the transformation is estimated with errors and the correlation is not found. This might be resolved by increasing the maximal distance of eigenvalues (see section 5.2.2), but this might lead to wrong correlations with other segments. To summarize, the homography-based method is still experimental and needs improvement which would have exceeded this work. However, since the APs between 127 and 244 have been recognized correctly in this real image sequence and since the method succeeded in the artificial sequence, it can be said that it goes into the right direction.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

In this thesis, a framework has been proposed that enriches raw image segmentation with object permanence. The problem of segment splits is safely resolved and the labels of formerly occluded segments are restored if a correlation with other segments had been established before the occlusion occurred. Additionally, the assignment of a set of segments to one object is ensured. On top of that, real object boundaries are extracted by explicitly handling occlusions of segments and deriving their depth relations.

One of the greatest achievements is that the whole system is model-free. Although small segments, which are most probably caused by noise, are removed, there is no conceptual limitation for segments being handled by the system. This makes it applicable for tracking tasks in which a prior selection of objects of interest is not possible, e.g. because (automatically) establishing these objects is the actual task or at least one part of it. As an example, *action recognition* can be mentioned which aims at handling actions with arbitrary objects (see section 2.1). For this particular high level task, the proposed framework also creates an abstract scene description for each frame given by the spatial relations of the segments which are then used to build a graph. Although that kind of description is very simple, quite promising results have been achieved as shown in Aksoy et al. [3].

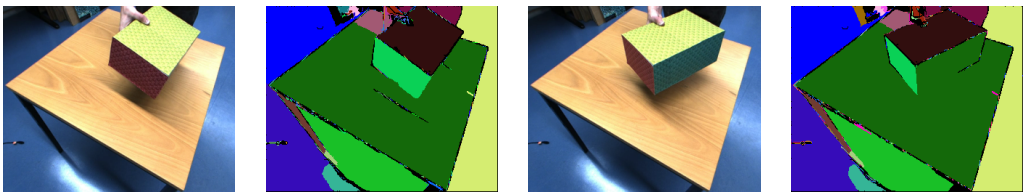
Concerning occlusions, the proposed framework shows two following key differences in comparison to other tracking methods (e.g. [20, 34]): (a) The framework provides a position of each hidden segment during the occlusion instead of "only" identifying it when it reappears. (b) The framework not only tracks a position in terms of x- and y-coordinates of each object but also considers its perspective appearances on the image plane, however, without the use of complex

3D reconstruction. For that purpose, this work has taken good care of obtaining precise homography estimations from point and line correspondences. Consequently, the segments' shapes and their temporal transformations were used as the only measurement for segment similarity which was used to match new segments with hidden tracked segments. Nonetheless, adding other measurements such as color should enhance the performance even more (see section 7.2). Besides these distinctions in the performance, there is, to the author's knowledge, no similar work which uses a layer-based approach together with homography estimation to provide object permanence in image segments.

The proposed framework not only provides a method for tracking and an abstract scene description but also a memory which is essential for object permanence. Besides updating the memory when new data is available, the system likewise removes data that turns out to be wrong. Although this might sound trivial, teaching robots how to forget is also an important issue.

The performance of the framework has been shown in three image sequences that have been recorded with a camera, and four artificial sequences, which, all together, cover the typical constellations related to splitting and binding problems. In order to present statistics on the accuracy and reliability of the system, more videos would have had to be analyzed. Even though the framework synthesizes abstract scene descriptions and is able to ensure object permanence, this work is only concerned with permanence issues in videos which do not include actions that could be analyzed by the action recognition algorithm. In other words, the permanence problem has been separated from the scene description task. (On the contrary, in Aksoy et al. [3] only sequences without permanence issues but with actions are shown.)

This is due to the fact that the used image segmentation algorithm in its current version has problems with occlusions and the appearance of new objects. These problems arise from its highly parallel implementation which is optimized for real-time. Note that the proposed framework requires the image segmentation to provide unique segment labels. If this is not the case as in Fig. 7.1, it is not able to distinguish them. Since the image segmentation is an external module, this is not a shortcoming of the proposed framework and therefore could either be solved by improving the algorithm in this specific area or by bringing in another algorithm. In conclusion, it can be said that since usable segmentation results where difficult to create, the movies shown in this thesis are rather simple and focus on the issue of permanence. In section 7.2, an idea is given how the proposed framework could help to solve the segmentation problems mentioned above.



(a) Frame n... (b) ...and segments, (c) frame $n+m$... (d) ...and segments.

Figure 7.1: Example where the used segmentation algorithm fails: The reappearing front side of the box and the table's surface have the same label, indicated by their identical colors in (d).

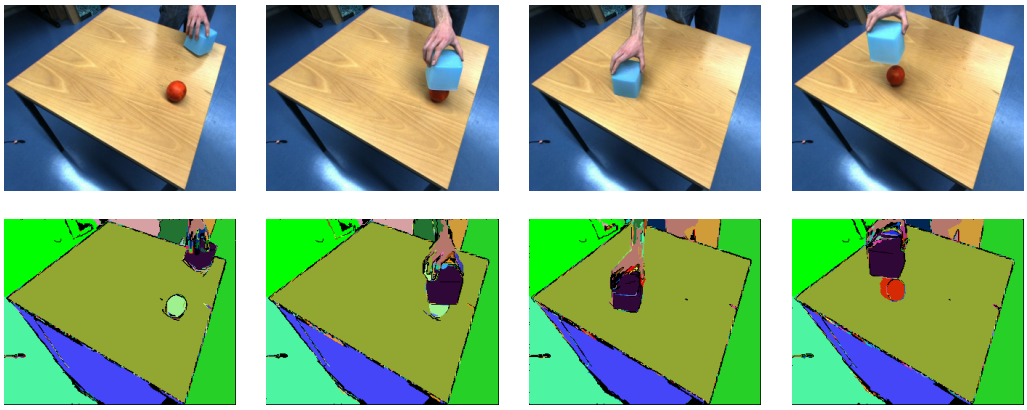
7.2 Outlook

There are already various ideas to improve the framework's performance on the one hand and for extending it by additional capabilities on the other hand.

An essential part of this work is the estimation of homography matrices. At first, only point correspondences between subsequent frames (in form of optical flow) have been used since they are also needed by the segmentation algorithm. Hence, the flow data can be used without additional computational cost. In order to increase the accuracy, line correspondences have been added later. Not only the fact that lines are in general more accurate than single points, but also the possibility of computing the transformation between arbitrary (and not only subsequent) frames is a big advantage. The latter proceeding had been called "skipping mode" in this thesis. However, the system only benefits of that mechanism when treating angular segments while arbitrarily shaped segments can still only be tracked from frame to frame using optical flow ("successive mode"). To overcome this limitation, the framework would have to be extended by using *feature points*, e.g. *SIFT features* [28]. The proposed framework can easily be modified to use that kind of point correspondences in "skipping mode" and would then be able to track arbitrarily shaped segments with a similar precision like angular segments.

The obtained transformation matrices could also be used to enrich the abstract scene description. Currently, the graphs only contain the spatial relations of segments and the description of the image sequence arises from their development over time. However, some actions cannot be covered by that, e.g. turning or shaking an object, since its movement is the essential hint and not its relation to another object. The estimated homographies could be used to capture such movements. A similar approach was shown by Shen et al. [39] where homographies are used to recognize human motion patterns.

While the homography estimation offers many opportunities, it is also the most noise-prone part of the proposed framework. For that reason, the estimation of expected deviation has been included. However, it only covers uncertainties originating from the pixel configuration with respect to the hidden parts of the layers. It does not cover errors that are caused by the fact that the observations in the form of the image segments are erroneous. Methods like *Kalman filtering* could be used to overcome this problem, similar to the work of Klappstein et al. [25]. Currently, errors are only treated by the use of RANSAC to recognize outliers, and thresholds. However, e.g. the establishing of alignment points for finding segment correlations has serious problems with imprecise homography estimations as had been shown in the results. Further work is definitely needed here for the system to become more robust.



(a) Frame 1 (b) Frame 30 (c) Frame 120 (d) Frame 150

Figure 7.2: Original images and segments of sequence "Shell_Game". The small box is put over the ball which becomes invisible by that. After moving the box and lifting it up again, the ball reappears but with a new segment number.

Another open issue is the recognition of correlations of already hidden objects. Currently, alignment points are created for visible segment pairs which seem to belong to the same object. Fig. 7.2 shows an example where the correlation of two segments cannot be observed before the occlusion, since the occlusion causes the correlation. Hence, the creation of alignment points has to be extended by a "guessing" method which finds an explanation for the fact that the ball is still invisible when the box moves to the left. Based on the experience that hidden objects keep in the region of their occluder (proposed by Huang et al. [20]), the proposed framework can be extended to achieve this goal.

The use of alignment points allows for tracking hidden segments during the occlusion. The framework's matching of these segments with newly appeared

segments requires them to be at the exact same position (with respect to the estimated deviation). The previously mentioned "guessing" of alignment points has to take into account that the hidden segment might not move in a perfectly correlated way: Regarding Fig. 7.2 it is easy to see that the ball has some freedom within the box. Hence, it might not be at the exact predicted position when the box moves up.

This fact becomes even more critical when more than one ball is caught by the box. While they are hidden, it is not possible to tell in advance in which configuration they will be when they become visible again. For that purpose, additional similarity measurements for segments might be helpful, e.g. color. This can be done either by storing a color distribution for each segment or by saving colored pixels in the layers.

The tracking of segments during the occlusion has obvious advantages for tasks which require their more or less precise positions all the time. However, besides that, the segmentation of the image sequence itself could be improved by implementing a feedback mechanism. Segmentation problems as shown in Fig. 7.1 only occur during the sequence but not in the initial frame. This is due to its optimization for real-time: The segmentation of the first frame takes relatively long while all subsequent frames are processed very quickly. Preventing errors like those shown could be done by performing a re-segmentation every few frames but this would increase the processing time. Instead of that, using the proposed layer framework, the regions in which a re-segmentation might be needed could be predicted: they correspond to the hidden segments (like the front side of the box). By that, only small parts of the image would have to be re-segmented.

For an increase of performance, this obviously requires that the proposed framework works in real-time which is not the case at the moment. It is programmed in Python and needs some seconds for each frame. However, the main computational effort is needed for the transformation of the layers, which can be done very quickly by the use of GPUs. The author is very optimistic that the proposed algorithms can be used to build a real-time system that, together with the used real-time segmentation algorithm, provides object permanence in image segments.

Bibliography

[1] Hough transform. http://en.wikipedia.org/wiki/Houghtransform.

[2] A. Abramov, E. Aksoy, J. Dörr, F. Wörgötter, K. Pauwels, and B. Dellen. 3d semantic representation of actions from efficient stereo-image-sequence segmentation on gpus. In 5th International Symposium 3D Data Processing, Visualization and Transmission, 2010.

[3] E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Unsupervised learning of object-action relations from semantic scene graphs. Int. J. Robotics Res. (IJRR)., 2011.

[4] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In ICRA, pages 398–405, 2010.

[5] R. Baillargeon, E. S. Spelke, and S. Wasserman. Object permanence in five-month-old infants. Cognition, 20(3):191–208, 1985.

[6] M. Blatt, S. Wiseman, and E. Domany. Superparamagnetic clustering of data. Physical Review Letters, 76(18):3251–3254, 1996.

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23:2001, 2001.

[8] A. Bugeau and P. Pérezz. Track and cut: simultaneous tracking and segmentation of multiple objects with graph cuts. J. Image Video Process., 2008:3:1–3:14, January 2008.

[9] O. Chum, J. Matas, and Š. Obdržálek. Enhancing RANSAC by generalized model optimization. In Proc. of the Asian Conference on Computer Vision (ACCV), pages 812–817, Seoul, Korea South, January 2004. Asian Federation of Computer Vision Societies.

[10] E. Dubrofsky and R. J. Woodham. Combining line and point correspondences for homography estimation. In Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II, ISVC '08, pages 202–213, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] C. Eckes and J. C. Vorbrüggen. Combining data-driven and model-based cues for segmentation of video sequences. World Congress on Neural Networks, San Diego, 1996.

[12] A. Elgammal and L. S. Davis. Probabilistic framework for segmenting people under occlusion. In In Proc. of IEEE 8th International Conference on Computer Vision, pages 145–152, 2001.

[13] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM, 24(6):381–395, June 1981.

[14] D. Geman and S. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell., 31(6):721–741, 1984.

[15] I. Haritaoglu, D. Harwood, and L. S. Davis. Who? when? where? what? In in 2.5 D In Eurepean Conference on Computer Vision, pages 877–892, 1998.

[16] I. Haritaoglu, D. Harwood, and L. S. Davis. Hydra: multiple people detection and tracking using silhouettes. In Image Analysis and Processing, 1999. Proceedings. International Conference on, pages 280–285, 1999.

[17] R. I. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[18] H. B. Helbig, J. Steinwender, M. Graf, and Kiefer. Action observation can prime visual object recognition. experimental brain research. 200(3-4):251–258, 2010.

[19] P. Hough. Method and means for recognizing complex patterns. U.S. Patent 3069654, 1962.

[20] Y. Huang and I. Essa. Tracking multiple objects through occlusions. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02, CVPR '05, pages 1051–1058, Washington, DC, USA, 2005. IEEE Computer Society.

[21] E. Ising. Beitrag zur Theorie des Ferromagnetismus. Z. Phys., 31:253–258, 1925.

[22] N. Jojic and B. J. Frey. Learning flexible sprites in video layers. In In CVPR, pages 199–206, 2001.

[23] S. Khan and M. Shah. Tracking people in presence of occlusion. In In Asian Conference on Computer Vision, pages 1132–1137, 2000.

[24] H. Kjellstrom, J. Romero, and D. Kragic. Simultaneous visual recognition of manipulation actions and manipulated objects. European Conference on Computer Vision, 2008.

[25] J. Klappstein, F. Stein, and U. Franke. Applying kalman filtering to road homography estimation. Workshop on Planning Perception and Navigation for Intelligent Vehicles, 2007.

[26] D. Koller, J. Weber, and J. Malik. Robust multiple car tracking with occlusion reasoning. Technical Report UCB/CSD-93-780, EECS Department, University of California, Berkeley, Nov 1993.

[27] P. König and N. Krüger. Perspectives: Symbols as self-emergent entities in an optimization process of feature extraction and predictions. Biological Cybernetics, 94(4):325–334, 2006.

[28] D. G. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image. U.S. Patent 6,711,293 B1, 2004.

[29] J. Malcolm, Y. Rathi, and A. Tannenbaum. Multi-Object Tracking Through Clutter Using Graph Cuts. Computer Vision, IEEE International Conference on, 0:1–5, 2007.

[30] J. S. Marques, P. M. Jorge, A. J. Abrantes, and J. M. Lemos. Tracking groups of pedestrians in video sequences. In In IEEE Workshop on Multi-Object Tracking, page 101, 2003.

[31] S. J. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler. Tracking groups of people. Computer Vision and Image Understanding, 80(1):42–56, 2000.

[32] N. Papadakis and A. Bugeau. Tracking with occlusions via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell., 33:144–157, 2011.

[33] D. Papadias, Y. Theodoridis, T. Sellis, and M. J. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with r-trees. pages 92–103, 1995.

[34] V. Papadourakis and A. Argyros. Multiple objects tracking in the presence of long-term occlusions. Comput. Vis. Image Underst., 114:835–846, July 2010.

[35] K. Pauwels and M. Van Hulle. Optic flow from unstable sequences through local velocity constancy maximization. Image and Vision Computing, 2008, in press.

[36] R. B. Potts. Some generalized order-disorder transformations. Proc. Cambridge Philos. Soc., 48:106–109, 1952.

[37] G. Rizzolatti and L. Craighero. The mirror-neuron system. Annual Review of Neuroscience, 27:169–192, 2004.

[38] S. Sabatini, G. Gastaldi, F. Solari, K. Pauwels, M. Van Hulle, J. Diaz, E. Ros, N. Pugeault, and N. Krueger. Compact (and accurate) early vision processing in the harmonic space. In VISAPP, pages 213–220, 2007.

[39] Y. Shen, N. Ashraf, and H. Foroosh. Action recognition based on homography constraints. In ICPR, pages 1–4, 2008.

[40] M. Sridhar, G. A. Cohn, and D. Hogg. Learning functional object-categories from a relational spatio-temporal representation. Proc. 18th European Conference on Artificial Intelligence, pages 606–610, 2008.

[41] R. Szeliski. Computer Vision: Algorithms and Applications. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[42] H. Tao, H. S. Sawhney, and R. Kumar. Object tracking with bayesian estimation of dynamic layer representations. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 24(1):75–89, 2002.

[43] P. H. S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. Computer Vision and Image Understanding, 78:138–156, 2000.

[44] I. Vicente, V. Kyrki, , and D. Kragic. Action recognition and understanding through motor primitives. 21(15):1687–1707, 2007.

[45] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. IEEE Transactions on Image Processing, 1994.

[46] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: real-time tracking of the human body. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):780–785, July 1997.

[47] L. Zelnik-manor and M. Irani. Multiview constraints on homographies. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24:214–223, 2002.

[48] H. Zeng, X. Deng, and Z. Hu. A new normalized method on line-based homography estimation. Pattern Recogn. Lett., 29:1236–1244, 2008.

[49] Z. Zhang. Estimating projective transformation matrix (collineation, homography). Technical report, Microsoft Research, 2010.

[50] Y. Zhou and H. Tao. A background layer model for object tracking through occlusion. In Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2, ICCV '03, pages 1079–, Washington, DC, USA, 2003. IEEE Computer Society.

List of Figures