Real-Time Segmentation of Stereo Videos on a Portable System With a Mobile GPU

Alexey Abramov, Karl Pauwels, Jeremie Papon, Florentin Wörgötter, and Babette Dellen

***Abstract*—In mobile robotic applications, visual information needs to be processed fast despite resource limitations of the mobile system. Here, a novel real-time framework for model-free spatiotemporal segmentation of stereo videos is presented. It combines real-time optical flow and stereo with image segmentation and runs on a portable system with an integrated mobile graphics processing unit. The system performs online, automatic, and dense segmentation of stereo videos and serves as a visual front end for preprocessing in mobile robots, providing a condensed representation of the scene that can potentially be utilized in various applications, e.g., object manipulation, manipulation recognition, visual servoing. The method was tested on real-world sequences with arbitrary motions, including videos acquired with a moving camera.**

***Index Terms*—Mobile systems, stereo segmentation, visual front end.**

I. Introduction

REAL-TIME VISUAL information is becoming more and more important in robotic applications for two main reasons. First, the research done during the past few decades in computer vision and image processing allows transforming visual information into more descriptive but nevertheless quite precise representations of the visual scene for use in a wide range of robotic applications, e.g., robot movement, object grasping, and object manipulation [23], [30]. Second, new hardware architectures and programming models for multicore computing have been proposed in the last ten years, through which many algorithms could be upgraded to real-time processing [1]. Currently, different hardware platforms are used as accelerators for complex computations in the domain of visual processing, such as multicore processors, digital signal processors, field programmable gate arrays, and graphics processing units (GPUs).

In the area of visual processing, the evolution of GPUs during the last four years has been of particular importance. GPUs are specialized microprocessors that have been initially invented for image processing and acceleration of 2-D and 3-D graphics rendering. GPUs are used in workstations, personal computers, mobile phones, and embedded systems. Presently, GPUs are a part of every computer and can be used immediately without any additional hardware upgrades. During the last four years, GPUs have evolved into highly parallel, multithreaded, multicore processors with tremendous computational power and very high memory bandwidth. For algorithms of high complexity, their parallel architecture makes them more efficient than general-purpose CPUs in many cases. Therefore, GPUs can be used not only for graphics processing but also for general-purpose parallel computing. Moreover, the graphics capabilities of GPUs make the visual output of the processed data directly from the microprocessor much simpler compared to other parallel platforms. The parallel programming model of compute-unified device architecture (CUDA) proposed by NVIDIA in 2007 makes parallelization of software applications on GPUs quite transparent [46].

However, processing power, memory bandwidth, and number of cores are not the only important parameters in robotic systems. Since robots are dynamic, movable, and very often wireless systems, huge processing platforms with high power consumption (mostly for cooling) are not practicable despite their high processing efficiency. Because of this, mobile parallel systems running on portable devices are of growing interest for computer-controlled robots. Today, mobile GPUs from the NVIDIA G8X series are supported by CUDA and can be used very easily for general-purpose parallel computing. In Fig. 1, the dynamics of development for desktop and mobile GPUs from the NVIDIA G8X series until today are shown, demonstrating that desktop GPUs are three times more powerful and have three times faster memory bandwidths than mobile ones. However, powerful desktop GPUs consume so much power that it is almost impossible to use them in small computer-controlled robots, while even the most powerful mobile GPUs integrated into mobile PCs do not need an extra power supply. Taking this fact into account, we consider in this paper a mobile PC with an integrated mobile GPU from

Manuscript received August 25, 2011; revised November 21, 2011; accepted January 12, 2012. Date of publication May 30, 2012; date of current version August 30, 2012. This work was supported by the European Community's Seventh Framework Program FP7/2007-2013 (Specific Program Cooperation, Theme 3, Information and Communication Technologies), under Grant 270273 by Xperience and under Grant Agreement 269959 by IntellAct. The work of K. Pauwels was supported by the Spanish Ministry for Science and Innovation through a Juan de la Cierva Fellowship JCI-2011-11019. The work of B. Dellen was supported by the Spanish Ministry for Science and Innovation through a Ramon y Cajal Fellowship. This paper was recommended by Associate Editor G. Lafruit.

A. Abramov, J. Papon, and F. Wörgötter are with the Department for Computational Neuroscience, Bernstein Center for Computational Neuroscience, Institute of Physics III, Georg-August University, Göttingen 37077, Germany (e-mail: abramov@physik3.gwdg.de; jpapon@physik3.gwdg.de; worgott@physik3.gwdg.de).

K. Pauwels is with the Department of Computer Architecture and Technology, University of Granada, Granada 18071, Spain (e-mail: kpauwels@atc.ugr.es).

B. Dellen is with the Institut de Robòtica i Informàtica Industrial, Barcelona 08028, Spain (e-mail: bdellen@iri.upc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2199389

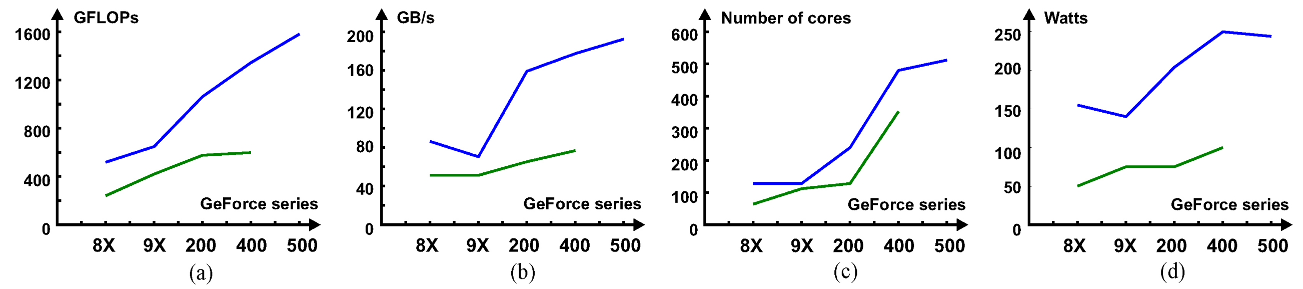


Fig. 1. Comparison of desktop (blue) and mobile (green) graphics cards for NVIDIA GeForce 8X, 9X, 100, 200, 400, 500-series GPUs with a minimum of 256 MB of local graphics memory. The following parameters are compared. (a) Processing power in floating point operations per second. (b) Maximum theoretical memory bandwidth. (c) Number of CUDA cores. (d) Graphics card power.

NVIDIA supported by CUDA as a portable system. Such a system can run for up to three hours in autonomous mode being supplied by the laptop battery.

Mobile robots have to process and structure abundant dynamic visual information in real time in order to interact with their environment in a meaningful way. For example, the understanding of the visual scene in terms of object and object-action relations [30] requires objects to be detected, segmented, tracked [41], and important descriptors, e.g., shape information, to be extracted [26]. This process corresponds to a dramatic compression of the initial visual data into symbol-like descriptors, upon which abstract logic or learning schemes can be applied. This occurs, for example, if a robot performs object manipulations or needs to come closer to an object to execute a grasping action. Finding this reduced symbol-like representation without prior knowledge on the data (model free) thus represents a major challenge in cognitive-vision applications—this problem is also known as the signal-symbol gap [40].

The video segmentation problem is generally formulated as the grouping of pixels into *spatiotemporal* volumes where each found object is uniquely identified and satisfies *temporal coherency*, i.e., carries the same label along the whole video stream [11], [12]. Several approaches for the video segmentation problem have been proposed over the last two decades. They can be summarized as follows.

Online and offline methods: Online video segmentation techniques use only preceding information and do not need future data. Such methods can segment video sequences of arbitrarily length in a continuous, sequential manner [6], [10], [12]–[14], [16], [18]. *Offline* methods, on the contrary, require the entire video sequence as input [5], [7], [11], [15]. Offline techniques are more robust in terms of temporal coherence, but they cannot be involved in perception-action loops since future perception is unknown.

Dense and sparse techniques: A video segmentation method is *dense* if it treats all objects visible in the scene trying to assign each pixel a proper spatiotemporal volume [5], [11]–[16], [18]. Techniques that perform segmentation of only preselected objects are *sparse* [6], [7], [10]. If not all objects are selected, the consequent employment of segments, given by sparse techniques, is very constrained and excludes an estimation of object positions relative to the environment.

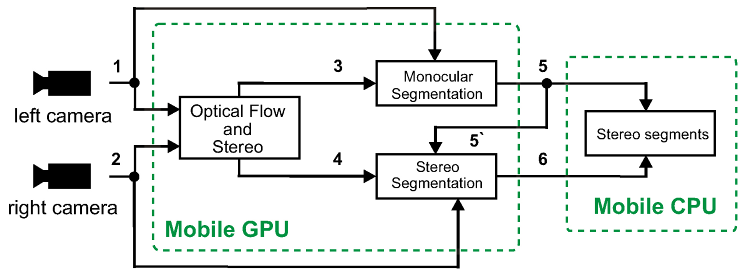
Automatic and nonautomatic approaches: The method is *automatic* or *unsupervised* if it runs without interaction with a user and does not need any prior knowledge about objects [12]–[15], [18]. *Nonautomatic* or *supervised* techniques are very often driven by user input, use some prior knowledge about the visual scene, and make assumptions about the number of objects present [5]–[7], [10], [16]. The hierarchical graph-based video segmentation, proposed by Grundmann *et al.* [11], can run in both automatic and nonautomatic modes.

Since mobile robots are usually autonomous systems that interact with their environment, only online automatic video segmentation techniques can be employed in the perception-action loop. Moreover, a complete information about the visual scene can be derived only by the use of dense methods. The following methods are the most famous and up-to-date online dense automatic video segmentation techniques.

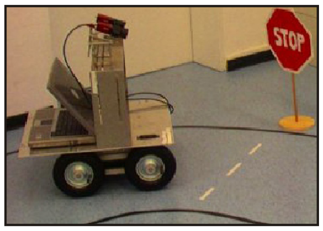
The mean-shift video segmentation, proposed by Paris [13], is based on the popular image segmentation technique by Comaniciu and Meer [28]. The temporal coherence is achieved by estimating the density of feature points, associated with all pixels, with a Gaussian kernel using data from all preceding frames. The method has a real-time performance on gray-level videos of size 640×360 pixels.

Multiple hypothesis video segmentation (MHVS) from superpixel flows [12] generates multiple presegmentations per frame considering only a few preceding frames. For each presegmentation it finds sequences of time consistent superpixels, called superpixel flows or hypotheses. Each hypothesis is considered as a potential solution and a hypothesis leading to the best spatiotemporal coherence. In this approach, the segmentation decision is postponed until evidence has been collected across several frames. Despite quite accurate segmentation results the MHVS needs seconds to process one frame, which makes it impossible to use it in real-time robotic applications.

Video segmentation based on propagation, validation, and aggregation of a preceding graph [14] exploits inter-frame correlations to propagate reliable groupings from the previous frame to the current. A preceding graph is built and labeled for the previous frame and temporally propagated to the current frame using a global motion estimation, followed by validation based on similarity measures. Pixels remained unlabeled after the propagation is grouped into subgraphs by a simple color clustering. Although the method gives results of a very high quality, it runs at frame rates inapplicable to real-time utilization.



(a)



(b)

Fig. 2. (a) Architecture of the framework for segmentation of stereo videos on the portable system with a mobile GPU. (b) Prototype of the movable robot steered by a mobile system with stereo cameras and a laptop with an integrated mobile GPU.

Matching images under unstable segmentations [18] is based on the fact that object regions obtained by existing segmentation methods do not always produce perceptually meaningful regions. In this approach, the current frame is segmented independently of preceding frames and the temporal coherence is achieved by region matching between the current and previous frames using the partial match cost, which allows fragments belonging to the same region to have low match cost with the original region. However, the method cannot run in real time due to a slow region matching procedure.

The last three approaches provide very accurate spatiotemporal volumes and can segment arbitrary long video sequences, but these methods do not run in real time and as a consequence cannot be employed in the perception-action loop. The mean-shift video segmentation approach, on the contrary, runs in real time, but works only on gray-scale videos and needs all past data to achieve satisfactory temporal coherence.

Although stereo data have recently been employed for segmentation [8], [9], there is no specialized, online, dense, and automatic method that performs spatiotemporal segmentation of stereo videos with establishing correspondences between left and right segments. Segmented stereo videos provide additional information about the scene and allow us to derive 3-D relations between objects [30]. Moreover, segment correspondences can be used for depth computation [43], which is of high importance for object manipulation tasks.

In this paper, we present a novel visual front end for real-time spatiotemporal segmentation of stereo videos on a mobile PC with an integrated mobile GPU. The proposed visual front end is online, automatic, dense, and solves the following problems.

1) Stereo images are segmented in a consistent model-free way.
2) The temporal coherence in a stereo video stream is achieved using a label-transfer strategy based on estimated motion and disparity data, resulting in a consistent partitioning of neighboring frames together with a consistent labeling. Only the results obtained on the very last left and right frames are employed at one time in order to guarantee spatiotemporal coherence for the current left and right frames, respectively.
3) All computations run in real time or close to real time, which allows the framework to be used in the perception-action loop.

Parts of this paper were previously published at a conference [26]. In this paper, some significant improvements have been achieved as compared to the conference version.

1) The used segmentation kernel has been optimized in order to achieve real-time performance.
2) The perceptual color space CIE ($L^*a^*b^*$) is used instead of the input RGB space, which leads to the formation of more accurate spatiotemporal volumes.
3) The method does not rely on motion estimated for the left video stream only. Motion estimation for the right stream is included, which makes right segments more stable and decreases processing time for segmentation of stereo videos.
4) The algorithm can now run on mobile GPUs.
5) The method was tested on more complex scenes including sequences with moving cameras, and the performance of the method has been quantified in much more detail.

This paper is organized as follows. In Section II, we describe the architecture of the framework together with the proposed algorithms. In Section III, we present the results of an extensive experimental evaluation, and finally, in Section IV, we conclude our work.

II. Segmentation of Stereo Videos

A. Overview

The architecture of the framework for segmentation of stereo videos is shown in Fig. 2(a). It consists of a stereo camera, a mobile computer with an integrated mobile GPU, and various processing components that are connected by channels in the framework. Each component in the framework can access the output data of all other components in the framework. The processing flow is as follows. Stereo images (synchronized left and right frames) are captured by a stereo camera. The acquired images are undistorted and rectified (in real-time with a fixed stereo geometry [1]) before they enter the framework. Optical flow is computed for the current left and right frames together with the disparity map on a GPU using real-time algorithms, and the results are accessible from channels 3 and 4, respectively (see Section II-C).

For videos from the left video stream, segment labels from the previous segmentation are warped to the current frame using the optical flow vector field (channel 3). This new label configuration is used as an initialization for the fast segmentation algorithm, which also runs on a GPU (see Section II-D). The adjustment of initial labels to the current frame will be referred to as *relaxation process*. This way the required time for segmentation of sequential frames can be reduced, and, even more importantly, a temporally coherent labeling of the frames can be achieved, i.e., segments describing the same object part are likely to carry the same label. The segmentation results of the left frame (monocular segmentation) can be accessed from channel 5.

A label initialization of the current right frame is created by warping of both the current left (channel 5′) and previous right segments using the disparity information and optical flow vector field (channel 4), respectively (see Section II-E). Similar to the segmentation of the left stream, the initial labels are adjusted to the current right frame by the relaxation process. The segmentation results of the right frame, which is now consistently labeled with respect to its corresponding left frame, are stored in channel 6.

Once segmentation for both left and right frames is achieved, meaningful stereo segments can be extracted. Segments smaller than a predefined threshold are removed. After all these processing steps each object is represented by uniquely identified left and right segments. This information can be exploited directly by a mobile robot. A prototype of the movable robot steered by a mobile system including stereo cameras and a laptop with an integrated mobile GPU is shown in Fig. 2(b).

B. Image Segmentation Kernel

Many different approaches for image segmentation have been proposed during the past three decades. Today, the most famous and efficient techniques are normalized cuts [29], graph-based [27], mean shift segmentation [28], graph cuts, and energy-based methods [21]. All these methods operate on single images, and cannot be applied directly to the video segmentation problem due to the temporal incoherence between adjacent frames, i.e., when segments of the same object keep different labels. As a consequence, some additional techniques are needed in order to link corresponding segments.

In the proposed framework, the real-time image segmentation algorithm based on the method of superparamagnetic clustering of data is used as an image segmentation kernel [3]. The method of superparamagnetic clustering represents an input image being segmented by a Potts model [44] of spins and solves the segmentation problem by finding the equilibrium states of the energy function of a ferromagnetic Potts model in the superparamagnetic phase [32], [33].

The Potts model 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. Three phases, depending on the system temperature, i.e., disorder introduced to the system, are observed: the paramagnetic, the superparamagnetic, and 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.* 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 [31]. Finding the image partition corresponds to the computation of the equilibrium states of the Potts model.

The equilibrium states of the Potts model have been approximated in the past using the Metropolis–Hastings algorithm with annealing [42] and methods based on cluster updating, which are known to accelerate the equilibration of the system by shortening the correlation times between distant spins, such as Swendsen–Wang [38], Wolff [39], and energy-based cluster updating [32], [33]. All of these methods obey detailed balance, ensuring convergence of the system to the equilibrium state. In this paper, we achieve efficient performance using the Metropolis algorithm with annealing [42], which can be easily parallelized and implemented on a GPU architecture.

The method of superparamagnetic clustering of data was chosen as an image segmentation kernel for the video segmentation problem due to the following advantages. Since the segmentation problem is solved by finding equilibrium states of the Potts model using an annealing procedure, there are no particular requirements to the initial states of spins and they can take on any one of the q available states. The closer the initial states are to the equilibrium, the less time the Metropolis algorithm needs to converge. This property allows us to achieve temporal coherency in the segmentation of monocular and stereo video streams just by using the previous segmentation results for the spin initialization of the current frame, while taking shifts between frames into account. A final segmentation result is obtained within a small number of Metropolis updates only, drastically reducing computation time. However, any other segmentation technique can be used for segmentation of the very first frame and the obtained segments can be considered as spin variables in the Potts model.

The real-time image segmentation kernel proceeds as follows. Using the Potts model an input image is represented in a form of color vectors $\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g_N}$ arranged on the $N = L_x L_y$ sites of a 2-D lattice. In the Potts model, a spin variable σ_k, which can take on q discrete values ($q > 2$) $w_1, w_2, \ldots, w_q$, called spin states, is assigned to each pixel of the image. We define a spin state configuration by $S = \{\sigma_1, \sigma_2, \ldots, \sigma_N\} \in \Omega$, where Ω is the space of all spin configurations. For video segmentation, the parameter q should be chosen as large as possible since the spin states need to serve also as segment labels. In our experiments, we used $q = 256$. It is important to note that this choice of q has no influence on the performance and computation time of the Metropolis algorithm itself. A global energy function or a cost function of this particular q-state Potts configuration $S \in \Omega$ is the Hamiltonian

$$H[S] = -\sum_{<i,j>} J_{ij} \delta_{\sigma_i \sigma_j} \tag{1}$$

which represents the system energy where $< i, j >$ denotes the closest neighborhood of spin i with $||i, j|| \leqslant \ell$, where ℓ is a constant that needs to be set. 2-D bonds (i, j) between two pixels with coordinates (x_i, y_i) and (x_j, y_j) are created only if $|(x_i - x_j)| \leqslant \ell$ and $|(y_i - y_j)| \leqslant \ell$. In this paper, we use

$\ell = 1$. J_{ij} is an interaction strength or coupling constant and the Kronecker δ_{ij} function is defined as $\delta_{ij} = 1$ if $\sigma_i = \sigma_j$ and zero otherwise, where σ_i and σ_j are the respective spin variables of two neighboring pixels i and j (see Fig. 4). A coupling constant, determining the interaction strength between two spins i and j, is given by

$$J_{ij} = 1 - \Delta_{ij}/\overline{\Delta} \tag{2}$$

where $\Delta_{ij} = ||\mathbf{g_i} - \mathbf{g_j}||$ is the color difference between the respective color vectors $\mathbf{g_i}$ and $\mathbf{g_j}$ of the input image. $\overline{\Delta}$ is the mean distance averaged over all bonds in the image. The interaction strength is defined in such a way that regions with similar color values will get positive weights with a maximum value of one for equal colors, whereas dissimilar regions get negative weights [47]. The mean distance $\overline{\Delta}$ represents the intrinsic (short-range) similarity within the whole input image[1]

$$\overline{\Delta} = \alpha \cdot \left(\frac{1}{N} \frac{1}{(2\ell+1)^2 - 1} \sum_{i=1}^{N} \sum_{<i,j>} ||\mathbf{g_i} - \mathbf{g_j}|| \right) \tag{3}$$

where $(2\ell+1)^2 - 1$ is the number of neighbors of a spin. The factor $\alpha \in (0, 10]$ is a system parameter used to increase or decrease the coupling constants.

Coupling constants are computed in the CIE $(L^*a^*b^*)$ color space [19] instead of the input RGB format. Although RGB is a widely used color space, it is not suitable for color segmentation and analysis because of the high correlation among all three components. The high correlation means that changes in intensity lead to changes in values of all three color components. The CIE $(L^*a^*b^*)$ color space, which is obtained by applying a nonlinear transformation to the RGB, is a perceptual color space that gives a better hint of how different two colors are for a human observer [22]. In the CIE $(L^*a^*b^*)$ space a pixel is represented by three values L^*, a^*, and b^* where L^* denotes lightness, while a^* and b^* denote color information. The color difference between two color vectors $\mathbf{g_i} = (L_i^*, a_i^*, b_i^*)^T$ and $\mathbf{g_j} = (L_j^*, a_j^*, b_j^*)^T$ is determined by [20]

$$||\mathbf{g_i} - \mathbf{g_j}|| = \sqrt{\Psi_L^2 + \Psi_C^2 + \Psi_H^2} \tag{4}$$

$$\Psi_L = \frac{\Delta L^*}{K_L} \quad \Psi_C = \frac{\Delta C_{ab}^*}{1 + K_1 C_i^*} \quad \Psi_H = \frac{\Delta H_{ab}^*}{1 + K_2 C_j^*} \tag{5}$$

$$\Delta L^* = L_i^* - L_j^* \quad C_i^* = \sqrt{(a_i^{*2} + b_i^{*2})} \tag{6}$$

$$C_j^* = \sqrt{(a_j^{*2} + b_j^{*2})} \quad \Delta C_{ab}^* = C_i^* - C_j^* \tag{7}$$

$$\Delta H_{ab}^* = \sqrt{\Delta a^{*2} + \Delta b^{*2} - \Delta C_{ab}^{*2}} \tag{8}$$

$$\Delta a^* = a_i^* - a_j^* \qquad \Delta b^* = b_i^* - b_j^* \tag{9}$$

where K_L, K_1, and K_2 are the weighting factors. Since the current method uses eight-connectivity of pixels, interaction strengths for each pixel of the image need to be computed in four different directions: horizontal, left diagonal, vertical, right diagonal [see Fig. 3(b)]. Matrices containing coupling constants that affect the formation of segments are shown for one image in Fig. 3(c)–(f).

[1]Note that (2) is ill-defined in the case of $\overline{\Delta} = 0$. But, in this case, only a single uniform surface exists and segmentation is not necessary.

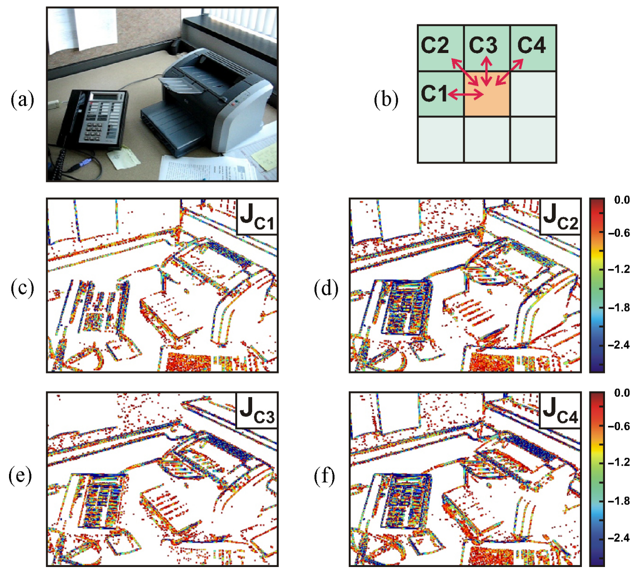


Fig. 3. Coupling constants for the eight-connectivity case in the CIE $(L^*a^*b^*)$ color space. (a) Original frame. (b) Mask for eight-connected connectivity. (c)–(f) Matrices with coupling constants computed for horizontal, left diagonal, vertical, and right diagonal directions. Note that only coupling constants leading to the formation of segments are shown here ($J < 0$).

The segmentation problem is solved by finding regions or clusters of correlated spins in the low-temperature equilibrium states of the Hamiltonian $H[S]$. Simulated annealing works by simulating a random walk on the set of spin states Ω looking for low-energy states. According to the Metropolis algorithm, one update iteration consists of the following steps.

1) The system energy $H[S_{\text{cur}}]$ of the current spin configuration S_{cur} is computed according to (1).
2) For each pixel i, a set of n (number of neighbors) new possible spin configurations $\hbar = S_1', S_2', \cdots, S_n'$ is created by changing the spin state of pixel i to the spin states of the neighbors. The number of new possible spin configurations n does not depend on q.
3) Every spin configuration $S_i' \in \hbar$ is considered as a potential new configuration of the system. Therefore, energy values of all configurations from the set $\hbar$ need to be computed according to (1).
4) Among all new possible configurations from the set $\hbar$ a spin configuration with the minimum energy value is selected according to

$$H[S_{\text{new}}] = \min(H[S_1'], H[S_2'], \cdots, H[S_n']). \tag{10}$$

The respective change in energy between the current configuration S_{cur} and a configuration $S_{\text{new}} \in \hbar$ having the energy value $H[S_{\text{new}}]$ is defined as $\Delta H \equiv H[S_{\text{new}}] - H[S_{\text{cur}}]$. According to $\Delta H > 0$ or $\Delta H \leqslant 0$, moves can be classified as uphill and downhill, respectively.
5) To effect a bias in favor of moves that decrease the energy, downhill moves are always accepted, whereas uphill moves are accepted only sometimes in order to

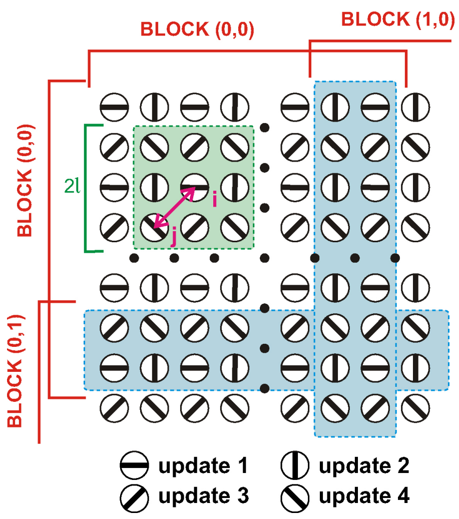


Fig. 4. Update of a spin state configuration of the input image on a GPU. Pixels depicted by the same pattern are updated simultaneously. Blue regions show overlaps between neighboring thread blocks. The green region includes $(2\ell + 1)^2 - 1$ pixels from the closest neighborhood of the pixel i. The arrow shows an interaction between pixels i and j.

avoid getting trapped in local minima. Therefore, the probability that the proposed move leading to increase in energy will be accepted is given by [17]

$$P(S_{\text{cur}} \to S_{\text{new}}) = \exp\left(-\frac{|\Delta H|}{T_n}\right). \tag{11}$$

A number ξ is drawn randomly from a uniform distribution in the range of [0, 1]. If $\xi < P(S_{\text{cur}} \to S_{\text{new}})$, the move is accepted.

6) The temperature is gradually reduced after every iteration according to the predefined annealing schedule $T_{n+1} = \gamma \cdot T_n$, where γ is the annealing coefficient ($\gamma < 1$).

The update process (1)–(5) runs until convergence, i.e., when no more spin flips toward a lower energy state are being observed. The equilibrium state of the system, achieved after several Metropolis iterations, corresponds to the image partition or segmentation. Then, the final segments larger than a predefined threshold are extracted.

Each spin update in the Metropolis algorithm involves only the nearest neighbors of the considered pixel. Hence, the spin variables of pixels that are not neighbors of each other can be updated simultaneously [45]. Therefore, the Metropolis algorithm fits very well to the GPU architecture and all spin variables can be updated in four iterations as shown in Fig. 4. On a GPU an image is divided into some processing blocks (taking overlaps between them into account), which are distributed between multiple multiprocessors on the card. In the current implementation, a thread block of size 16 × 16 is used and each thread loads and updates four pixels. Such a configuration makes it possible to avoid idle threads (only loading data from overlaps without performing any spin update) and to use the resources of the GPU in a very efficient way.

Four coupling constants (a byte each) and a current spin value (one byte) are loaded from the global memory to the shared memory for each pixel resulting in five matrices of size 32 × 32 within every thread block. In total, 5 kB of the shared memory is occupied by one block, which makes it possible to run four blocks on each multiprocessor of the NVIDIA GeForce GT 240M at the same time. Because of the high intensity of the Metropolis procedure with numerous accesses to the same data within one iteration, the shared memory is used for data access due to its low latency instead of the global memory. Since threads from diverse thread blocks cannot cooperate with each other, overlaps between blocks need to be synchronized via the global memory, once all spin variables of the current spin configuration are updated. Global memory throughput is maximized through the coalesced memory accesses achieved by rearrangement of input data in the global memory and usage of data types fulfilling the size and alignment requirements [2]. Due to the fact that the coupling constants can be computed simultaneously for all pixels in the image, this procedure is performed on the GPU as well.

In this paper, only the first frame from the left video stream is segmented completely from scratch (all spin variables are initialized randomly). Subsequent left frames and their corresponding right frames are first initialized by a spin state configuration taken from previous frames considering movements and stereo displacements between them (see Section II-A). Therefore, the relaxation process is applied to preinitialized images to adjust initial spins to the current frame (see Sections II-D, II-E).

Since every found segment is carrying a spin variable that is unique within the whole image, the terms *spin* and *label* are equivalent in this paper.

C. Phase-Based Optical Flow and Disparity

Since fast processing is a very important issue in this paper, the real-time phase-based optical flow and stereo algorithms, proposed by Pauwels *et al.* [34], are used to find pixel correspondences between adjacent frames in a monocular video stream and left and right frames in a stereo video stream. Both algorithms run on a GPU and belong to the class of phase-based techniques, which are highly robust to changes in contrast, orientation, and speed. The optical flow 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. Although any other optical flow estimation technique can be used here [36], we decided on the mentioned phase-based approach since it combines high accuracy with computational efficiency. Furthermore, due to the shared image representation based on the responses of a Gabor filterbank, stereo correspondences, used to find corresponding stereo segments (see Section II-E), can be obtained with very little overhead. A comparable qualitative evaluation of the methods including test sequences from the Middlebury benchmark can be found in [35] and [37]. Implementation details and performance analyses of the phase-based optical flow and stereo algorithms are given in [35].

D. Monocular Segmentation Using Optical-Flow-Based Label Warping

In the current framework optical flow is computed for both the left and right video streams. The algorithm provides a

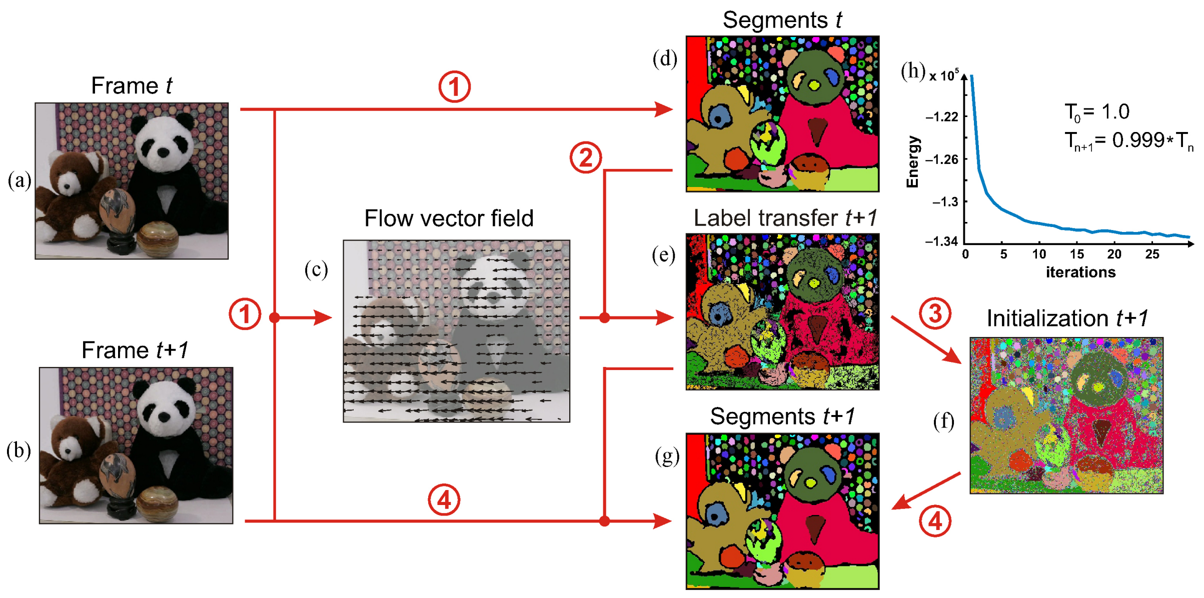


Fig. 5. Segmentation of two adjacent frames in a sequence using 30 iterations and $\alpha = 2.5$. Numbers at arrows show the sequence of computations. (a) Original frame t. (b) Original frame $t+1$. (c) Estimated optical flow vector field from the phase-based method (subsampled 13 times and scaled six times) (step 1). (d) Extracted segments S_t for frame t (step 1). (e) Label transfer from frame t to frame $t+1$ (step 2). (f) Initialization of frame $t+1$ for the image segmentation kernel (step 3). (g) Extracted segments S_{t+1} for frame $t+1$ (step 4). (h) Convergence of the Metropolis algorithm for frame $t+1$.

vector field that indicates the motion of pixels in textured regions

$$\mathbf{u}(x, y) = (u_x(x, y), u_y(x, y)). \tag{12}$$

An estimated optical flow vector field for two adjacent frames t and $t+1$ out of the *Toy* sequence with a moving camera from the motion annotation benchmark[2] is shown in Fig. 5(a)–(c). Since we are using a local algorithm, optical flow cannot be estimated everywhere (for example not in the very weakly textured black regions of the panda toy). For pixels in these regions, vertical and horizontal flows, i.e., u_y and u_x, do not exist. Suppose frame t is segmented and S_t is its final label configuration [see Fig. 5(d)]. An initial label configuration for frame $t+1$ is found by warping all labels from frame t taking estimations from the optical flow vector field into account [see Fig. 5(e)]

$$S_{t+1}(x_{t+1}, y_{t+1}) = S_t(x_t, y_t) \tag{13}$$

$$x_t = x_{t+1} - u_x(x_{t+1}, y_{t+1}) \tag{14}$$

$$y_t = y_{t+1} - u_y(x_{t+1}, y_{t+1}) \tag{15}$$

where (u_x, u_y) is the flow at time $t+1$. Since there is only one flow vector per pixel, there will only be one label transferred per pixel. Note that it is not the case if the flow at time t is used for linking, since there can be multiple flow vectors pointing to the same pixel in frame $t+1$.

Pixels that did not obtain an initialization via (13) are then given a randomly chosen label between 1 and q, which is not occupied by any of the found segments [see Fig. 5(f)]. Once frame $t+1$ is initialized, a relaxation process (see Section II-B) is needed in order to fix erroneous bonds that can take place during the transfer of spin states. Flow interpolations for weakly textured regions are not considered in this paper because: 1) the image segmentation kernel inherently incorporates the data during spin relaxation; and 2) an interpolation based on a camera motion estimation is only useful in static scenes (with moving cameras), but cannot help when dealing with moving objects.

The relaxation process runs until convergence and only after that are the final segments extracted [see Fig. 5(g) where the corresponding segments between frames t and $t+1$ are labeled with identical colors]. Convergence of the relaxation process against a number of iterations is shown in Fig. 5(h). For the relaxation process we use a schedule with the starting temperature $T_0 = 1.0$ and annealing coefficient $\gamma = 0.999$. As we can see the annealing process with this schedule converges after 25–30 iterations, making it possible to segment stereo video streams with a frame size of 320×256 in real time. Longer annealing schedules can lead to better segmentation results, but at the cost of processing time.

E. Stereo Segmentation Using Disparity-Based Label Warping

The segmentation of every right frame is performed in a similar way. Here, an initial label configuration for the right frame at time t is obtained by warping the labels from both the corresponding left frame and the previous right frame. Labels from the left frame are transferred using the disparity map D [see Fig. 6(a)–(c)] and labels from the previous right frame are transferred using the optical flow vector field [see Fig. 6(e)]. Since the stereo algorithm relies on phase (and not magnitude), it can find correct matches even in weakly textured regions. Also, ambiguous matches are avoided by the use of a coarse-to-fine control mechanism. However, it cannot find reliable information under drastically changing light conditions (see the reflection shift over the table). We suppose that the left

[2]Available at http://people.csail.mit.edu/celiu/motionAnnotation.

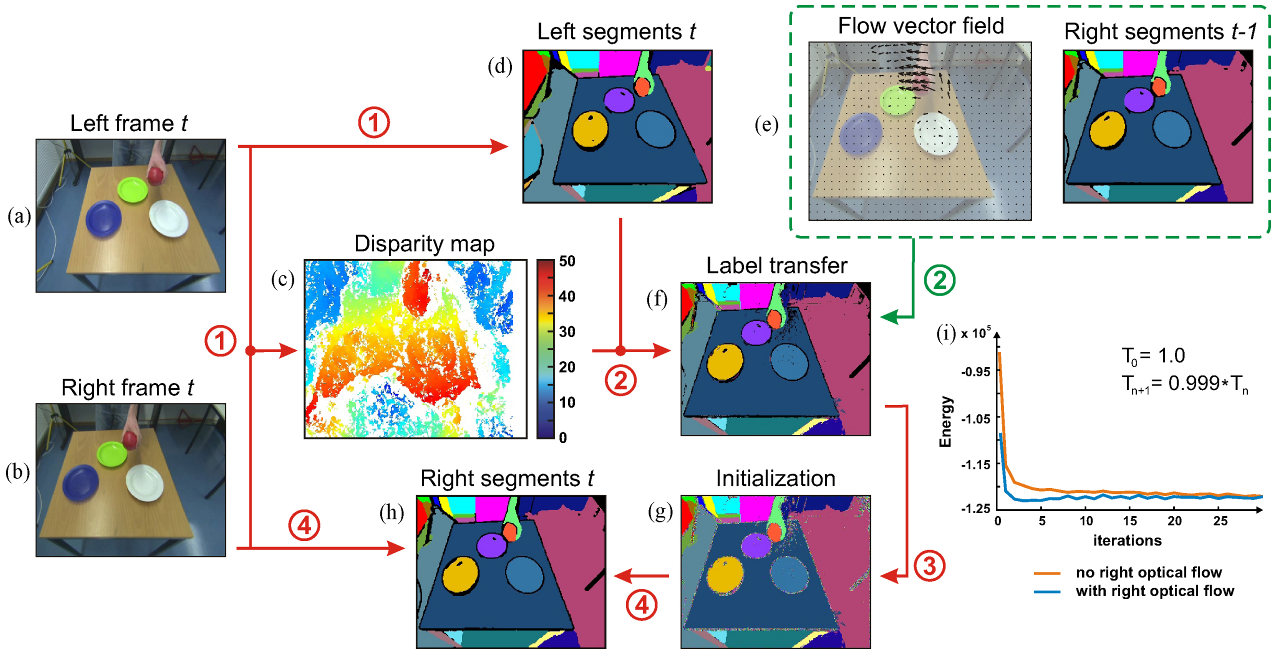


Fig. 6. Segmentation of a stereo pair for the time moment t. Numbers at arrows show the sequence of computations. (a) Original left frame L_t. (b) Original right frame R_t. (c) Disparity map estimated by the phase-based method (step 1). (d) Extracted segments S_L for frame L_t after 30 iterations with $\alpha = 2.5$ (step 1). (e) Segments and estimated optical flow vector field for right frame $t-1$ (subsampled 13 times and scaled six times). (f) Label transfer from frame L_t to frame R_t (step 2). (g) Initialization of frame R_t for the image segmentation kernel (step 3). (h) Extracted segments S_R for frame R_t after ten iterations with $\alpha = 2.5$ (step 4). (i) Convergence of the Metropolis algorithm for frame R_t.

frame L_t is segmented and S_L is its final label configuration [see Fig. 6(d)]. Labels from the previous right frame R_{t-1} are warped according to the procedure described in Section II-D, whereas the labels from the current left frame L_t are warped as follows:

$$S_R(x_R, y_R) = S_L(x_L, y_L) \tag{16}$$

$$x_L = x_R + D(x_R, y_R), \quad y_L = y_R. \tag{17}$$

The disparity map D is computed relative to the right frame, which guarantees that there will only be one label transferred per pixel from the left frame. Both warpings are performed at the same time [see Fig. 6(f)]. In the case of multiple correspondences, i.e., if a pixel in frame R_t has label candidates in frames L_t and R_{t-1}, there are no preferences and we select randomly either the flow or the stereo. In this way, they can both contribute without bias and the segmentation kernel can make the final decision. Pixels that did not obtain a label initialization via (16) are given a randomly chosen label between 1 and q, which is not occupied by any of the found segments [see Fig. 6(g)]. Once frame R_t is initialized, a relaxation process (see Section II-B) is needed in order to fix erroneous bonds that can take place during the transfer of spins. The relaxation process runs again until it converges and only after that are the final right segments S_R at time t extracted [see Fig. 6(h), where the corresponding segments between frames L_t and R_t are labeled with identical colors].

Convergence of the relaxation process against a number of iterations is shown in Fig. 6(i) for the proposed label transfer and for the label transfer based only on disparity shifts. Here, we use the same annealing schedule as for the segmentation of the left video stream. We can see that the use of the previous right labels drastically reduces a number of iterations needed for convergence and already after five to ten iterations the final right segments can be extracted. Using only stereo information, about 25–30 iterations are needed in order to reach the equilibrium state. This is because the occlusions in the stereo images are significantly larger than the occlusions between adjacent frames taken from one video stream.

III. Experimental Results

A. *Quantitative Evaluation*

To measure the quality of video segmentations we use the *segmentation covering* metric introduced by Arbeláez *et al.* [24]. The idea of the metric is to evaluate the covering of a machine segmentation S by a human segmentation S'. A human segmentation, called also *ground-truth segmentation*, is a manual annotation of a video sequence showing how humans perceive the scene, whereas a machine segmentation is an output result of the considered video segmentation algorithm. In this paper, the human-assisted motion annotation tool proposed by Liu *et al.* [16] is used, which allows a user to annotate video sequences very efficiently. For one frame, the segmentation covering metric is defined as

$$C(S' \to S) = \frac{1}{N} \sum_{R \in S} |R| \cdot \max_{R' \in S'} O(R, R') \tag{18}$$

where N is the total number of pixels in the image, $|R|$ the number of pixels in region R, and $O(R, R')$ is the overlap

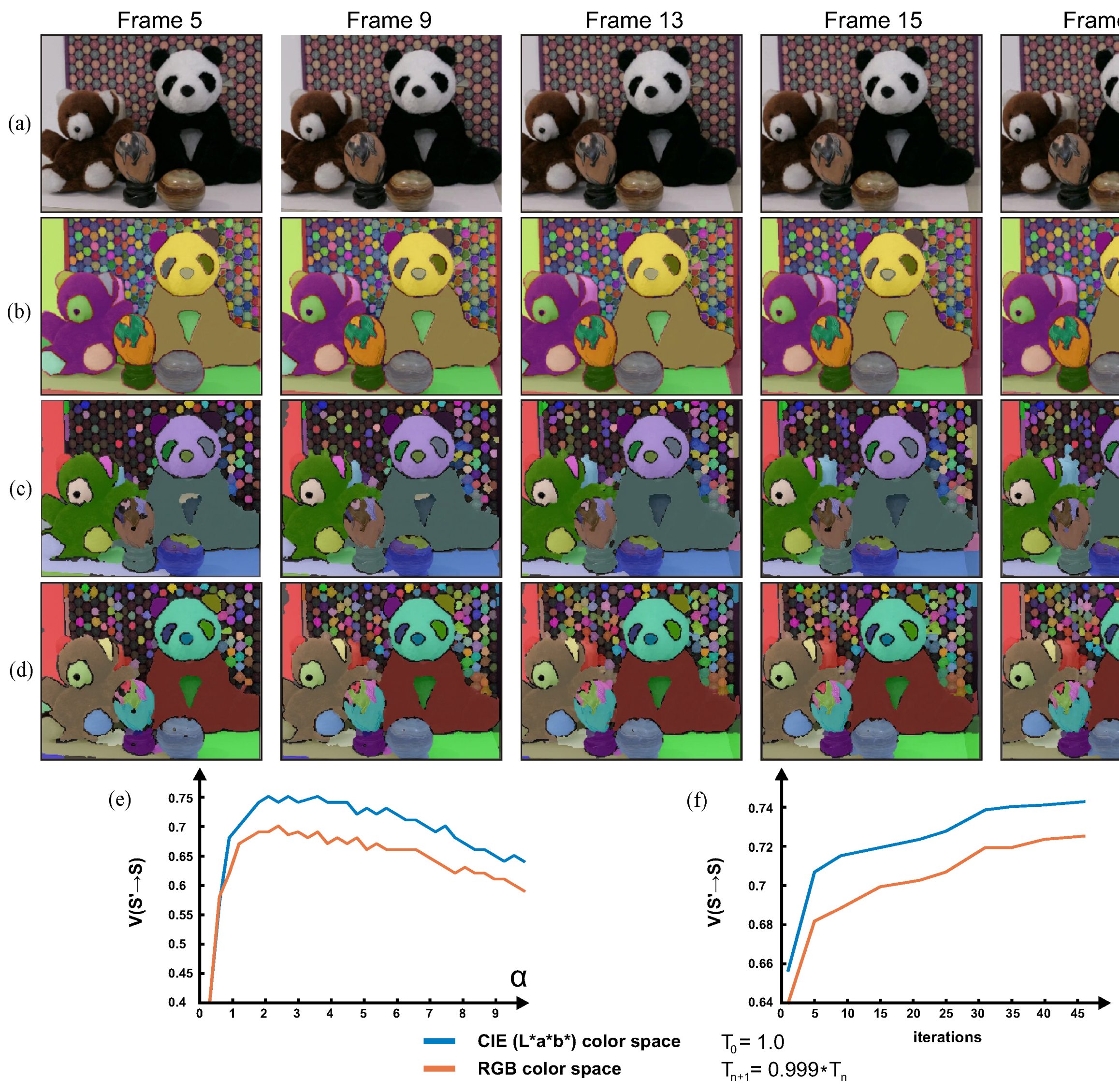


Fig. 7. Segmentation results for the *Toy* monocular video sequence with a moving camera. (a) Original frames. (b) Ground-truth segmentation created by the human-assisted annotation. (c) Machine segmentation performed in the input RGB color space (30 iterations, $\alpha = 2.5$), $V(S' \to S) = 0.69$. (d) Machine segmentation performed in the perceptual color space CIE ($L^*a^*b^*$) (30 iterations, $\alpha = 2.5$), $V(S' \to S) = 0.75$. (e), (f) Segmentation covering shown for both color spaces against the system parameter α and the number of iterations.

between regions R and R' defined as

$$O(R, R') = \frac{|R \cap R'|}{|R \cup R'|}. \tag{19}$$

To find the most similar machine spatiotemporal volume for each volume from the ground-truth video segmentation, we compute the segmentation covering for all ground-truth regions in all frames applying the following constraints: 1) all segments within one volume are temporally coherent, i.e., carry the same label in all frames; and 2) if the ground-truth volume extends over the more frames than the selected machine volume, then we repeat the selection of the best overlap over the remaining time intervals [15]. The segmentation covering for the video sequence is computed by averaging of the segmentation coverings over all frames M in the sequence

$$V(S' \to S) = \frac{1}{M} \sum_{i=1}^{M} C_i(S' \to S). \tag{20}$$

For the segmentation evaluation of stereo videos temporal coherence in both the left and right streams needs to be taken into account.

B. Monocular Segmentation Results

In Fig. 7, video segmentation results for the *Toy* video sequence with a moving camera are presented. The ground-truth segmentation created with the human-assisted motion annotation tool is shown in Fig. 7(b). Note that the ground-truth segmentations provided on the webpage of the motion annotation benchmark cannot be used for the comparison in this paper, since they show layer segmentation based on motion only without considering color differences. The video segmentation results for both the RGB and CIE color spaces are shown in Fig. 7(c) and (d), respectively. In both cases, the same segmentation parameters and the same annealing schedule were used. As we can see, the results obtained in

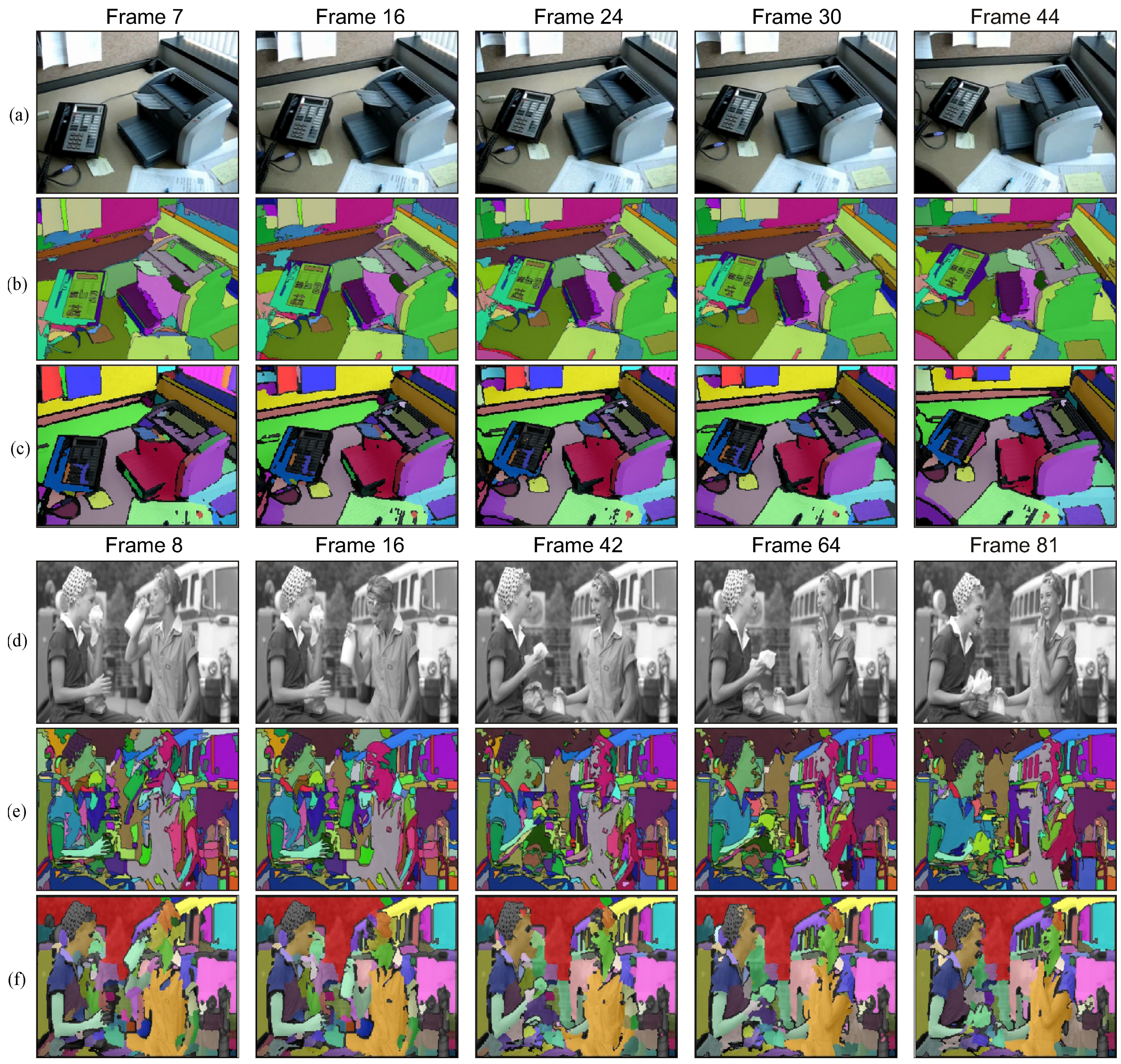


Fig. 8. Segmentation results for monocular video sequences *Phone* (a) with a moving camera and *Women* (d) with moving objects. (b) Graph-based video segmentation results obtained at 70% of highest hierarchy level, $V(S' \rightarrow S) = 0.55$. (c) Segmentation results from the proposed method derived after 30 iterations with $\alpha = 1.5$, $V(S' \rightarrow S) = 0.63$. (e) Graph-based video segmentation results obtained at 50% of highest hierarchy level, $V(S' \rightarrow S) = 0.35$. (f) Segmentation results from the proposed method derived after 30 iterations with $\alpha = 2.0$, $V(S' \rightarrow S) = 0.46$.

the CIE color space are more accurate, which is confirmed by the comparison of segmentation covering values computed for both color spaces and shown against the system parameter α in Fig. 7(e). Moreover, the image segmentation kernel in the CIE space needs less time to converge. Fig. 7(f) shows how segmentation covering values are changing for both color spaces depending on the number of iterations in the relaxation process.

More segmentation results in the CIE color space are shown in Fig. 8. Here, one more sequence (*Phone*) from the same benchmark is used together with a well-known sequence, *Women*, containing moving objects [see Fig. 8(a) and (d)]. Segmentation results for both sequences obtained by the proposed method are shown in Fig. 8(c) and (f). Although all types of sequences can be successfully segmented using the same set of parameters (30 iterations for the relaxation, $\alpha = 2.5$ and $T_{n+1} = 0.999 \cdot T_n$ starting with $T_0 = 1.0$), here α was slightly tuned for each sequence to get the best possible segmentation results. The proposed method is compared here to the hierarchical graph-based video segmentation [11], which is known as one of the most efficient spatiotemporal segmentation techniques. Since the graph-based approach uses future data for segmentation and not ours, both methods cannot be compared entirely and here we only show that our approach gives output comparable to the results of the conventional video segmentation methods.

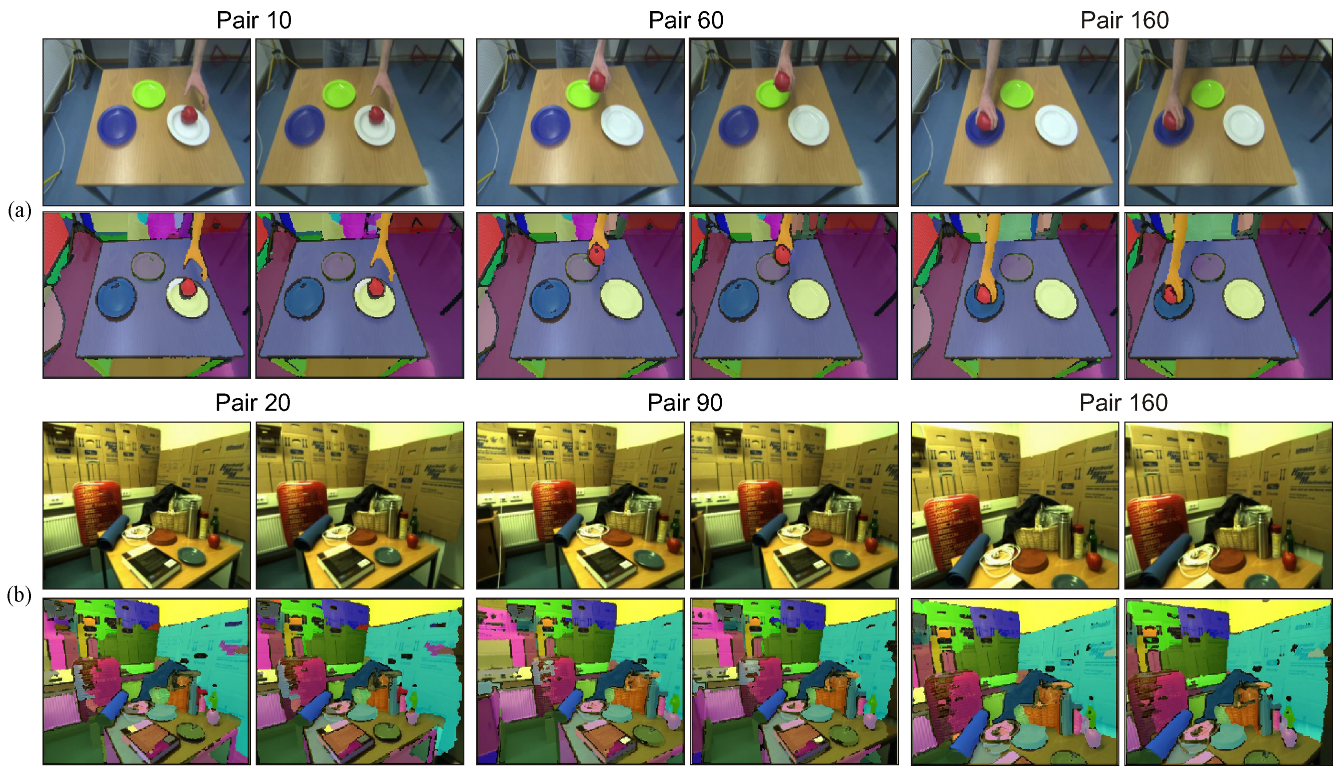


Fig. 9. Segmentation results for stereo frame sequences of the sample actions "moving an apple over plates" with (a) moving objects and "cluttered scene" with a (b) moving stereo camera. Results are obtained using the following parameters: 30 and 15 iterations are applied for the relaxation of left and right frames, respectively, $\alpha = 2.5$ for both the left and right streams, the annealing schedule is $T_{n+1} = 0.999 \cdot T_n$ starting with $T_0 = 1.0$.

From three hierarchy levels available on the webpage[3] for segmentation, the best segmentation result for each sequence was chosen [see Fig. 8(b), (e)]. We can see that the graph-based method leads sometimes to dramatic merges of segments or oversegmentations, which is not the case in the proposed approach (for example the proposed method outperforms the graph-based method at the boundary of the fax machine). However, the graph-based technique deals in some situations better with very textured objects (like the phone in the *Phone* sequence). Also note that the gray-scale *Women* sequence is an extremely difficult case for color-based segmentation techniques due to the lack of color information. The proposed approach has higher segmentation covering values compared to the graph-based technique for the examples shown, but one has to be aware that ground-truth segmentation obtained for these complex scenarios must be considered biased at least to some degree. For the *Phone* and *Women* sequences, the first and the last frames were used to compute the segmentation covering values.

C. Stereo Segmentation Results

Segmentation results for two stereo videos are shown in Fig. 9. Since the sequences are quite long, only some stereo pairs at key points of actions are shown. In the first sequence, called "moving an apple over plates," a hand moves an apple around the table and places it on a plate [see Fig. 9(a)]. In the second scenario, "cluttered scene," the scene is static, but the stereo camera moves [see Fig. 9(b)].

[3]Available at http://neumann.cc.gt.atl.ga.us/segmentation.

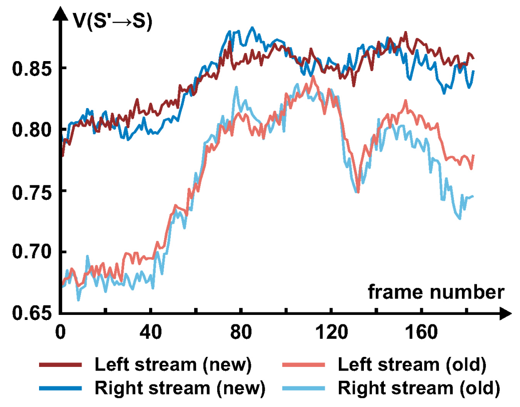


Fig. 10. Segmentation covering for the stereo sequence "moving an apple over plates" shown for the previous and current framework versions. The average values are 0.77 (left stream) and 0.76 (right stream) for the previous version and 0.84 (left and right streams) for the current version, respectively.

As we can see the temporal coherence along with consistent labeling is achieved in the segmentation of both stereo sequences and the determined stereo segments correspond to the natural partitioning of the original stereo pairs. Too small segments are completely removed from the final label configuration. The performance comparison of the proposed framework with its previous version [26] (using input RGB color space and optical flow for the left stream only) is shown in Fig. 10 as segmentation covering against the current frame number. As we can see, in the proposed framework the left and right sequences are segmented with higher accuracy (the average segmentation covering value is 0.84 for both

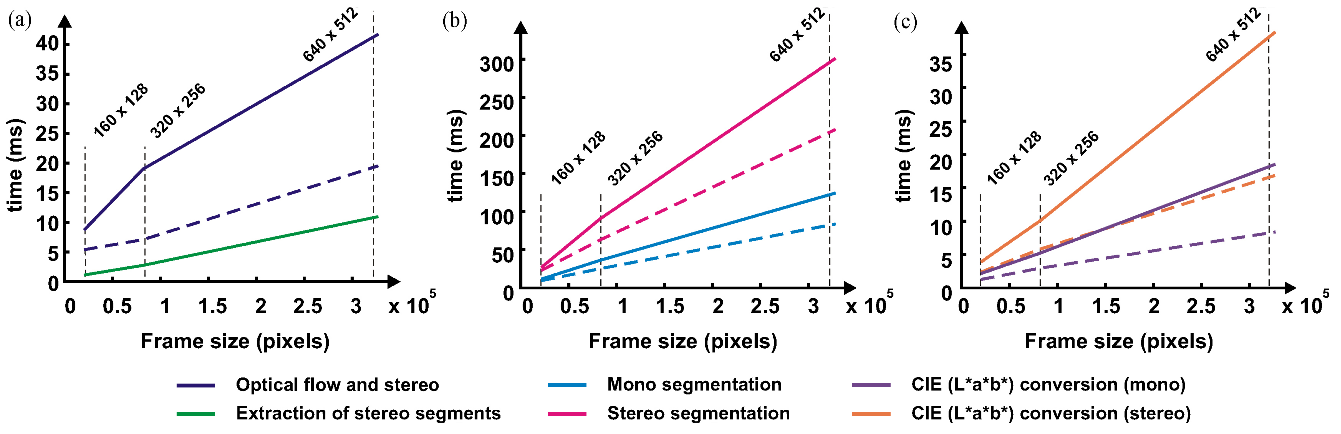


Fig. 11. Processing times of all stages of the framework for segmentation of stereo videos on the mobile system with an integrated mobile GPU. For computations running on the mobile GPU, processing times derived on the desktop GPU are shown for comparison (dashed lines). (a) Runtime for optical flow with stereo and extraction of stereo segments. (b) Processing time of monocular segmentation (30 iterations) and stereo segmentation (15 additional iterations). (c) Runtime for conversion from the input RGB color space to the CIE ($L^*a^*b^*$) for both monocular and stereo sequences.

streams). Moreover, the current version is more robust, having significantly smaller deviations of the segmentation covering values along the whole sequence.

D. Experimental Environment

The proposed framework runs on a laptop with mobile Intel Core 2 Duo CPU with 2.2 GHz and 4 GB RAM. The mobile GPU used in the laptop is NVIDIA GeForce GT 240M (with 1 GB device memory). This card has six multiprocessors and 48 processor cores in total and belongs to the 200-series of mobile NVIDIA GPUs. The card is shared by all the framework components running on the GPU. As a desktop GPU (used for the comparison of processing times) we use NVIDIA GeForce GTX 295 (with 896 MB device memory) consisting of two GPUs, each of which has 30 multiprocessors and 240 processor cores in total. In this paper, we use only one GPU of this card.

E. Processing Time

The processing times for all components of the framework are shown as a function of frame size in Fig. 11. Image resolutions 160 × 128, 320 × 256, and 640 × 512 are marked by black dashed lines. The processing times of components running on the mobile GPU are compared to the respective runtime on the desktop GPU [Fig. 11(a)–(c)]. For segmentation of monocular video streams 30 Metropolis iterations are used, whereas for stereo video streams 45 iterations are needed in total [see Fig. 11(b)]. Note that the relaxation process takes about 60% of the whole runtime.

Although all computations on the mobile card are significantly slower (the speedup factors derived on the desktop card in relation to the mobile one for optical flow or stereo and segmentation kernel are 2.1 and 2.4, respectively), it is still possible to process several frames per second for all considered resolutions as shown in Table I.

TABLE I
OBTAINED PROCESSING TIMES PER FRAME AND FRAME RATES

	CPU	GTX 295	GT 240M
Resolution (px)	s (Hz)	ms (Hz)	ms (Hz)
160 × 128	0.8 (1.2)	40.0 (25.0)	47.4 (21.1)
320 × 256	3.4 (0.3)	75.0 (13.3)	117.0 (8.5)
620 × 512	13.9 (0.1)	230.0 (4.3)	376.0 (2.7)

IV. Discussion and Conclusion

We presented a novel framework for real-time spatiotemporal segmentation of stereo video streams on a portable system with an integrated mobile GPU. The proposed visual front end is online, automatic, and dense. The performance of the framework has been demonstrated on real-world sequences acquired with moving cameras and containing arbitrary moving objects. A tradeoff between processing time and hardware configuration exists. Since robotic systems are usually dynamic, movable and very often wireless autonomous systems, huge computers with high power consumption were not even considered in this paper as a proper hardware architecture. As the most suitable platform for this task, we chose a mobile PC with an integrated mobile GPU. Being supplied by the laptop battery such a system can run in autonomous mode up to three hours. A GPU is used as an accelerator for highly parallel computations of the system such as optical flow, stereo, and image segmentation kernel. For the frame resolutions of 160 × 128 and 320 × 256 we achieved a processing time that is sufficient for many real-time robotic applications. The system manages to process bigger frames as well, but not in real time.

The following problems are solved by the visual front end: stereo images from stereo videos are segmented in a consistent model-free way (without prior knowledge of data), the temporal coherence in a stereo video stream is achieved resulting in a consistent labeling of the original frames. However, consistent labeling for a long video sequence can be obtained by the proposed framework only for quite simple scenarios. The scenarios are given as follows.

1) Objects should not get entirely occluded along the action, since the current method can deal only with partial occlusions. If an object is occluded by any other

object, it will not be recognized when it reappears. In order to properly track occluded objects, additional mechanisms are needed that perform high-level analysis of objects [4], [25]. It is not possible to resolve such kinds of problems on the pixel domain.

2) Objects should not move too fast. Phase-based optical flow and stereo used in the current system have a speed limit of 2 pixels per scale, so using four scales, the limit is $2^4 = 16$ pixels [34]. In the case of a very fast movement, more than 50% of the label transfers can be erroneous. This leads to a completely erroneous initialization of the current frame, which cannot be resolved by the relaxation process. The segmentation covering value for such a segment will be dramatically low, which signals inaccurate video segmentation. For the tracking of fast moving objects large displacement optical flow is needed [48].
3) No disjointed parts of physically the same object should be joined during the action. If two large parts of the same object represented by different segments are merged, we face the domain fragmentation problem when large uniform areas are being split into subsegments despite high attractive forces within them [47]. In the current system, the domain fragmentation problem can be resolved only by a very long annealing schedule (see Section II-B), which cannot be achieved in real time.

An important goal of this paper was the improvement of the computational speed of the system, since a low latency in the perception-action loop is a crucial requirement of systems where a visual front end is needed. Consequently, since the proposed framework is running in real time or close to real-time mode, it can be used in a wide range of robotic applications, such as object manipulation, visual servoing, and robot navigation. All these applications require object detection and tracking along with extraction of meaningful object descriptors as a preprocessing step.

In the future, we aim to overcome the mentioned problems 1)–3) and apply the developed framework to more complex actions in the context of cognitive robotics tasks. For very complex scenarios where objects are getting occluded all the time, some high-level knowledge about objects needs to be accumulated during that part of the sequence where objects are present and visible.

Acknowledgment

The authors would like to thank T. Kulvicius for his valuable discussions.

References

[1] G. Bradski, "The OpenCV library," *Doctor Dobb's J. Softw. Tools*, vol. 25, no. 11, pp. 122–126, Nov. 2000.

[2] *Nvidia CUDA C Programming Guide*, NVIDIA Corporation, Santa Clara, CA, 2011.

[3] A. Abramov, T. Kulvicius, F. Wörgötter, and B. Dellen, "Real-time image segmentation on a GPU," in *Proc. Facing Multicore-Challenge*, vol. 6310. 2010, pp. 131–142.

[4] K. Nummiaro, E. Koller-Meier, and L. J. Van Gool, "Object tracking with an adaptive color-based particle filter," in *Proc. DAGM Symp.*, 2002, pp. 353–360.

[5] Y. Huang, Q. Liu, and D. N. Metaxas, "Video object segmentation by hypergraph cut," in *Proc. CVPR*, 2009, pp. 1738–1745.

[6] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. J. Van Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Proc. ICCV*, 2009, pp. 1515–1522.

[7] M. Unger, T. Mauthner, T. Pock, and H. Bischof, "Tracking as segmentation of spatial-temporal volumes by anisotropic weighted TV," in *Proc. EMMCVPR*, 2009, pp. 193–206.

[8] Ľ. Ladický, P. Sturgess, C. Russell, S. Sengupta, Y. Bastanlar, W. Clocksin, and P. Torr, "Joint optimization for object class segmentation and dense stereo reconstruction," in *Proc. BMVC*, 2010, pp. 1–11.

[9] C. D. Mutto, P. Zanuttigh, G. M. Cortelazzo, and S. Mattoccia, "Scene segmentation assisted by stereo vision," in *Proc. 3DIMPVT*, 2011, pp. 57–64.

[10] C. Wang, M. de la Gorce, and N. Paragios, "Segmentation, ordering and multi-object tracking using graphical models," in *Proc. ICCV*, 2009, pp. 747–754.

[11] M. Grundmann, V. Kwatra, M. Han, and I. A. Essa, "Efficient hierarchical graph-based video segmentation," in *Proc. CVPR*, 2010, pp. 2141–2148.

[12] A. V. Reina, S. Avidan, H. Pfister, and E. L. Miller, "Multiple hypothesis video segmentation from superpixel flows," in *Proc. ECCV*, 2010, pp. 268–281.

[13] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *Proc. ECCV*, 2008, pp. 460–473.

[14] S. Liu, G. Dong, C. H. Yan, and S. H. Ong, "Video segmentation: Propagation, validation and aggregation of a preceding graph," in *Proc. CVPR*, 2008, pp. 1–7.

[15] W. Brendel and S. Todorovic, "Video object segmentation by tracking regions," in *Proc. ICCV*, 2009, pp. 833–840.

[16] C. Liu, W. T. Freeman, E. H. Adelson, and Y. Weiss, "Human-assisted motion annotation," in *Proc. CVPR*, 2008, pp. 1–8.

[17] P. Salamon, P. Sibani, and R. Frost, *Facts, Conjectures, and Improvements for Simulated Annealing*. Philadelphia, PA: Soc. Indust. Appl. Math., 2002.

[18] V. Hedau, H. Arora, and N. Ahuja, "Matching images under unstable segmentations," in *Proc. CVPR*, 2008, pp. 1–8.

[19] D. C. Tseng and C. H. Chang, "Color segmentation using perceptual attributes," in *Proc. Int. Conf. Patt. Recog.*, 1992, pp. 228–231.

[20] *Industrial Colour-Difference Evaluation*, CIE Pub. 116-1995, CIE Central Bureau, Vienna, Austria, 1995.

[21] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient N-D image segmentation," *Int. J. Comput. Vis.*, vol. 70, no. 2, pp. 109–131, 2006.

[22] H. D. Cheng, X. H. Jiang, Y. Sun, and J. L. Wang, "Color image segmentation: Advances and prospects," *Pattern Recognit.*, vol. 34, no. 12, pp. 2259–2281, 2001.

[23] H. Kjellström, J. Romero, and D. Kragic, "Visual object-action recognition: Inferring object affordances from human demonstration," *Comput. Vision Image Understand.*, vol. 115, no. 1, pp. 81–90, 2011.

[24] P. Arbelaez, M. Maire, C. C. Fowlkes, and J. Malik, "From contours to regions: An empirical evaluation," in *Proc. CVPR*, 2009, pp. 2294–2301.

[25] J. Y. A. Wang and E. H. Adelson, "Representing moving images with layers," *IEEE Trans. Image Process.*, vol. 3, no. 5, pp. 625–638, May 1994.

[26] A. Abramov, E. E. Aksoy, J. Dörr, K. Pauwels, F. Wörgötter, and B. Dellen, "3D semantic representation of actions from efficient stereo-image-sequence segmentation on GPUs," in *Proc. 5th Int. Symp. 3DPVT*, 2010, pp. 1–8.

[27] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[28] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[29] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[30] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of object-action relations by observation," *Int. J. Robot. Res.*, vol. 30, no. 10, pp. 1229–1249, 2011.

[31] M. Blatt, S. Wiseman, and E. Domany, "Superparamagnetic clustering of data," *Phys. Rev. Lett.*, vol. 76, no. 18, pp. 3251–3254, 1996.

[32] C. von Ferber and F. Wörgötter, "Cluster update algorithm and recognition," *Phys. Rev. E*, vol. 62, no. 2, pp. 1461–1464, 2000.

[33] R. Opara and F. Wörgötter, "A fast and robust cluster update algorithm for image segmentation in split-lattice models without annealing: Visual latencies revisited," *Neural Computat.*, vol. 10, no. 6, pp. 1547–1566, 1998.

[34] K. Pauwels, N. Krüger, M. Lappe, F. Wörgötter, and M. M. Van Hulle, "A cortical architecture on parallel hardware for motion processing in real time," *J. Vision*, vol. 10, no. 10, pp. 1–18, 2010.

[35] K. Pauwels, M. Tomasi, J. Diaz, E. Ros, and M. M. Van Hulle, "A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 999–1012, Jul. 2012.

[36] A. Wedel, T. Pock, C. Zach, D. Cremers, and H. Bischof, "An improved algorithm for TV-L1 optical flow," in *Proc. Dagstuhl Motion Workshop*, 2008.

[37] T. Gautama and M. M. Van Hulle, "A phase-based approach to the estimation of the optical flow field using spatial filtering," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1127–1136, Sep. 2002.

[38] R. Swendsen and S. Wang, "Nonuniversal critical dynamics in Monte Carlo simulations," *Phys. Rev. Lett.*, vol. 76, no. 18, pp. 86–88, 1987.

[39] U. Wolff, "Collective Monte Carlo updating for spin systems," *Phys. Rev. Lett.*, vol. 62, no. 4, pp. 361–364, 1989.

[40] P. König and N. Krüger, "Perspectives: Symbols as self-emergent entities in an optimization process of feature extraction and predictions," *Biol. Cybern.*, vol. 94, no. 4, pp. 325–334, 2006.

[41] P. Salembier and F. Marqués, "Region-based representations of image and video: Segmentation tools for multimedia services," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 8, pp. 1147–1169, Dec. 1999.

[42] D. Geman and S. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 6, pp. 721–741, Nov. 1984.

[43] B. Dellen and F. Wörgötter, "Disparity from stereo-segment silhouettes of weakly-textured images," in *Proc. BMVC*, 2009, pp. 1–11.

[44] R. B. Potts, "Some generalized order-disorder transformations," *Proc. Cambridge Philos. Soc.*, vol. 48, no. 1, pp. 106–109, 1952.

[45] G. T. Barkema and T. MacFarland, "Parallel simulation of the Ising model," *Phys. Rev. E*, vol. 50, no. 2, pp. 1623–1628, 1994.

[46] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar.–Apr. 2008.

[47] C. Eckes and J. C. Vorbrüggen, "Combining data-driven and model-based cues for segmentation of video sequences," in *Proc. World Congr. Neural Netw.*, 1996, pp. 868–875.

[48] T. Brox and J. Malik, "Large displacement optical flow: Descriptor matching in variational motion estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 500–513, Mar. 2011.

Alexey Abramov received the M.Sc. degree in computer science from the Moscow Engineering and Physics Institute, State University, Moscow, Russia. He is currently pursuing the Ph.D. degree with the Department for Computational Neuroscience, Bernstein Center for Computational Neuroscience, Institute of Physics III, Georg-August University, Göttingen, Germany.

His current research interests include image processing, image segmentation and object tracking, stereo image processing, and real-time computer vision with high-performance computing on parallel hardware.

Karl Pauwels received the M.Sc. degree in commercial engineering, the M.Sc. degree in artificial intelligence, and the Ph.D. degree in medical sciences from Katholieke Universiteit Leuven, Leuven, Belgium.

He is currently a Post-Doctoral Researcher with the Department of Computer Architecture and Technology, University of Granada, Granada, Spain. His current research interests include optical flow, stereo, and camera motion estimation in the context of real-time computer vision.

Jeremie Papon received the M.Sc. degree in electrical engineering from Stanford University, Palo Alto, CA. He is currently pursuing the Ph.D. degree with the Department for Computational Neuroscience, Bernstein Center for Computational Neuroscience, Institute of Physics III, Georg-August University, Göttingen, Germany.

His current research interests include Bayesian predictive filtering, image processing using massively parallel hardware, and real-time vision system architectures.

Florentin Wörgötter received the Dipl.Biol. degree in biology and mathematics from the University of Düsseldorf, Düsseldorf, Germany, and the Ph.D. degree from the University of Essen, Essen, Germany, in 1988. His Ph.D. research was focused on the visual cortex.

He was with the California Institute of Technology, Pasadena, from 1988 to 1990, engaged in computational issues. He became a Researcher with the University of Bochum, Bochum, Germany, in 1990, where he worked on experimental and computational neuroscience of the visual system. He was a Professor of computational neuroscience with the Department of Psychology, University of Stirling, Stirling, Scotland, from 2000 to 2005, where his interests included learning in neurons. He has been with the Department for Computational Neuroscience, Bernstein Center for Computational Neuroscience, Institute of Physics III, Georg-August University, Göttingen, Germany, since July 2005. His group has developed the RunBot, a fast and adaptive biped walking robot. His current research interests include information processing in closed-loop perception-action systems, which includes aspects of sensory processing, motor control, and learning or plasticity. These approaches are tested in walking, as well as in driving robotic implementations.

Babette Dellen received the Diplom degree in physics from the University of Cologne, Cologne, Germany, and the Ph.D. degree in physics from Washington University, St. Louis, in 2006, where she worked on computational models of the visual system of birds and mammals.

She was a Post-Doctoral Researcher with the Department of Computational Neuroscience, Bernstein Center for Computational Neuroscience, University of Göttingen, Göttingen, Germany, and with the Max-Planck-Institute for Dynamics and Self-Organization, Göttingen, from 2006 to 2010, where she worked mainly on computer vision. She is currently with the Institut de Robòtica i Informàtica Industrial, Barcelona, Spain, a joint research center of the Technical University of Catalonia, Catalonia, Spain, and the Spanish Council for Scientific Research. Her current research interests include computer vision methods for generating suitable visual representations for robotic applications.

Dr. Dellen received the Ramon y Cajal Fellowship from the Spanish Ministry for Science and Innovation in 2009.