1

Computer Vision in Vehicles

Reinhard Klette
School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand

This chapter is a brief introduction to academic aspects of computer vision in vehicles. It briefly summarizes basic notation and definitions used in computer vision. The chapter discusses a few visual tasks as of relevance for vehicle control and environment understanding.

1.1 Adaptive Computer Vision for Vehicles

Computer vision designs solutions for understanding the real world by using cameras. See Rosenfeld (1969), Horn (1986), Hartley and Zisserman (2003), or Klette (2014) for examples of monographs or textbooks on computer vision.

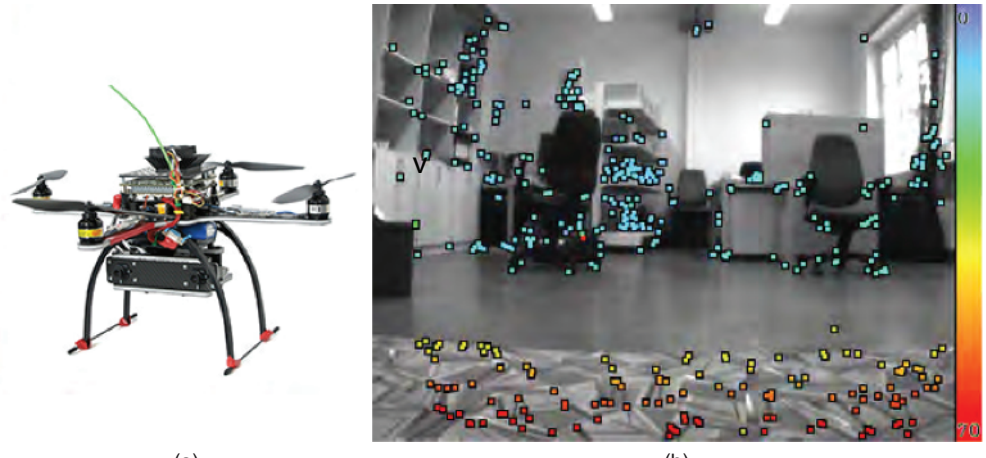
Computer vision operates today in *vehicles* including cars, trucks, airplanes, unmanned aerial vehicles (UAVs) such as multi-copters (see Figure 1.1 for a quadcopter), satellites, or even autonomous driving rovers on the Moon or Mars.

In our context, the *ego-vehicle* is that vehicle where the computer vision system operates in; *ego-motion* describes the ego-vehicle's motion in the real world.

1.1.1 Applications

Computer vision solutions are today in use in manned vehicles for improved safety or comfort, in autonomous vehicles (e.g., robots) for supporting motion or action control, and also for misusing UAVs for killing people remotely. The UAV technology has also good potentials for helping to save lives, to create three-dimensional (3D) models of

Computer Vision in Vehicle Technology: Land, Sea, and Air, First Edition.
Edited by Antonio M. López, Atsushi Imiya, Tomas Pajdla and Jose M. Álvarez.



(a) (b)

Figure 1.1 (a) Quadcopter. (b) Corners detected from a flying quadcopter using a modified FAST feature detector. Courtesy of Konstantin Schauwecker

the environment, and so forth. Underwater robots and unmanned sea-surface vehicles are further important applications of vision-augmented vehicles.

1.1.2 Traffic Safety and Comfort

Traffic safety is a dominant application area for computer vision in vehicles. Currently, about 1.24 million people die annually worldwide due to traffic accidents (WHO 2013), this is, on average, 2.4 people die *per minute* in traffic accidents. How does this compare to the numbers Western politicians are using for obtaining support for their "war on terrorism?" Computer vision can play a major role in solving the true real-world problems (see Figure 1.2). Traffic-accident fatalities can be reduced by controlling traffic flow (e.g., by triggering automated warning signals at pedestrian crossings or intersections with bicycle lanes) using stationary cameras, or by having cameras installed in vehicles (e.g., for detecting safe distances and adjusting speed accordingly, or by detecting obstacles and constraining trajectories).

Computer vision is also introduced into modern cars for improving driving comfort. Surveillance of blind spots, automated distance control, or compensation of unevenness of the road are just three examples for a wide spectrum of opportunities provided by computer vision for enhancing driving comfort.

1.1.3 Strengths of (Computer) Vision

Computer vision is an important component of intelligent systems for vehicle control (e.g., in modern cars, or in robots). The Mars rovers "Curiosity" and "Opportunity" operate based on computer vision; "Opportunity" has already operated on Mars for

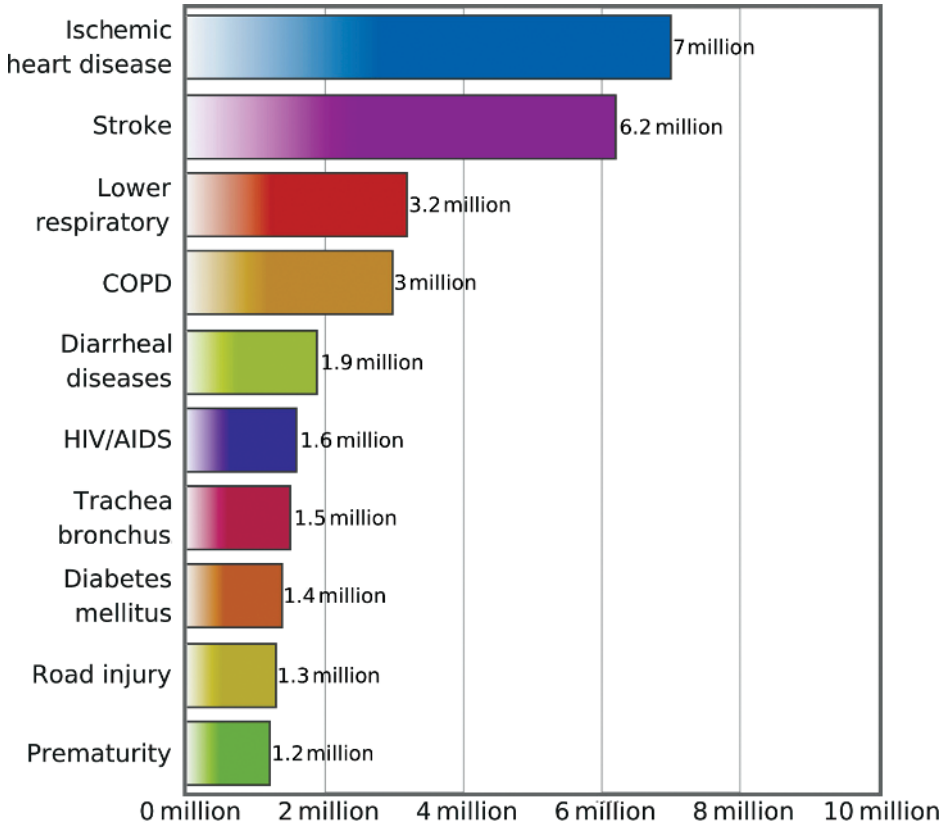


Figure 1.2 The 10 leading causes of death in the world. Chart provided online by the World Health Organization (WHO). Road injury ranked number 9 in 2011

more than ten years. The visual system of human beings provides a proof of existence that vision alone can deliver nearly all of the information required for steering a vehicle. Computer vision aims at creating comparable automated solutions for vehicles, enabling them to navigate safely in the real world. Additionally, computer vision can also work constantly "at the same level of attention," applying the same rules or programs; a human is not able to do so due to becoming tired or distracted.

A human applies accumulated knowledge and experience (e.g., supporting intuition), and it is a challenging task to embed a computer vision solution into a system able to have, for example, intuition. Computer vision offers many more opportunities for future developments in a vehicle context.

1.1.4 Generic and Specific Tasks

There are *generic visual tasks* such as calculating distance or motion, measuring brightness, or detecting corners in an image (see Figure 1.1b). In contrast, there are

specific visual tasks such as detecting a pedestrian, understanding ego-motion, or calculating the *free space* a vehicle may move in safely in the next few seconds. The borderline between generic and specific tasks is not well defined.

Solutions for generic tasks typically aim at creating one self-contained *module* for potential integration into a complex computer vision system. But there is no general-purpose corner detector and also no general-purpose stereo matcher. *Adaptation* to given circumstances appears to be the general way for an optimized use of given modules for generic tasks.

Solutions for specific tasks are typically structured into multiple modules that interact in a complex system.

Example 1.1.1 Specific Tasks in the Context of Visual Lane Analysis *Shin et al. (2014) review visual lane analysis for driver-assistance systems or autonomous driving. In this context, the authors discuss specific tasks such as "the combination of visual lane analysis with driver monitoring..., with ego-motion analysis..., with location analysis..., with vehicle detection..., or with navigation...." They illustrate the latter example by an application shown in Figure 1.3: lane detection and road sign reading, the analysis of GPS data and electronic maps (e-maps), and two-dimensional (2D) visualization are combined into a real-view navigation system (Choi et al. 2010).*

Figure 1.3 Two screenshots for real-view navigation. Courtesy of the authors of Choi et al. (2010)

1.1.5 Multi-module Solutions

Designing a multi-module solution for a given task does not need to be more difficult than designing a single-module solution. In fact, finding solutions for some single modules (e.g., for motion analysis) can be very challenging. Designing a multi-module solution requires:

1. that modular solutions are available and known,
2. tools for evaluating those solutions in dependency of a given situation (or *scenario*; see Klette et al. (2011) for a discussion of scenarios) for being able to select (or adapt) solutions,
3. conceptual thinking for designing and controlling an appropriate multi-module system,
4. a system optimization including a more extensive testing on various scenarios than for a single module (due to the increase in combinatorial complexity of multi-module interactions), and
5. multiple modules require control (e.g., when many designers separately insert processors for controlling various operations in a vehicle, no control engineer should be surprised if the vehicle becomes unstable).

1.1.6 Accuracy, Precision, and Robustness

Solutions can be characterized as being *accurate*, *precise*, or *robust*. *Accuracy* means a systematic closeness to the true values for a given scenario. *Precision* also considers the occurrence of random errors; a precise solution should lead to about the same results under comparable conditions. *Robustness* means approximate correctness for a set of scenarios that includes particularly challenging ones: in such cases, it would be appropriate to specify the defining scenarios accurately, for example, by using video descriptors (Briassouli and Kompatsiaris 2010) or data measures (Suaste et al. 2013). Ideally, robustness should address *any* possible scenario in the real world for a given task.

1.1.7 Comparative Performance Evaluation

An efficient way for a comparative performance analysis of solutions for one task is by having different authors testing their own programs on identical benchmark data. But we not only need to evaluate the programs, we also need to evaluate the benchmark data used (Haeusler and Klette 2010, 2012) for identifying their challenges or relevance.

Benchmarks need to come with *measures* for quantifying performance such that we can compare accuracy on individual data or robustness across a diversity of different input data.

Figure 1.4 illustrates two possible ways for generating benchmarks, one by using computer graphics for rendering sequences with accurately known ground truth,[1] and the other one by using high-end sensors (in the illustrated case, ground truth is provided by the use of a laser range-finder).[2]

[1] For EISATS benchmark data, see www.mi.auckland.ac.nz/EISATS.
[2] For KITTI benchmark data, see www.cvlibs.net/datasets/kitti/.

(a) (b)

Figure 1.4 Examples of benchmark data available for a comparative analysis of computer vision algorithms for motion and distance calculations. (a) Image from a synthetic sequence provided on EISATS with accurate ground truth. (b) Image of a real-world sequence provided on KITTI with approximate ground truth

But those evaluations need to be considered with care since everything is not comparable. Evaluations depend on the benchmark data used; having a few summarizing numbers may not be really of relevance for particular scenarios possibly occurring in the real world. For some input data we simply can not answer how a solution performs; for example, in the middle of a large road intersection, we cannot answer which lane border detection algorithm performs best for this scenario.

1.1.8 There Are Many Winners

We are not so naive to expect an all-time "winner" when comparatively evaluating computer vision solutions. Vehicles operate in the real world (whether on Earth, the Moon, or on Mars), which is so diverse that not all of the possible event occurrences can be modeled in underlying constraints for a designed program. Particular solutions perform differently for different scenarios, and a winning program for one scenario may fail for another. We can only evaluate how particular solutions perform for particular scenarios. At the end, this might support an optimization strategy by adaptation to a current scenario that a vehicle experiences at a time.

1.2 Notation and Basic Definitions

The following basic notations and definitions (Klette 2014) are provided.

1.2.1 Images and Videos

An *image I* is defined on a set

$$\Omega = \{(x, y) : \ 1 \leq x \leq N_{\text{cols}} \wedge 1 \leq y \leq N_{\text{rows}}\} \subset \mathbb{Z}^2 \tag{1.1}$$

of pairs of integers (*pixel locations*), called the image *carrier*, where N_{cols} and N_{rows} define the number of columns and rows, respectively. We assume a left-hand coordinate system with the coordinate origin in the upper-left corner of the image, the x-axis to the right, and the y-axis downward. A *pixel* of an image I combines a location $p = (x, y)$ in the carrier Ω with the value $I(p)$ of I at this location.

A *scalar image* I takes values in a set $\{0, 1, \ldots, 2^a - 1\}$, typically with $a = 8$, $a = 12$, or $a = 16$. A *vector-valued image* I has scalar values in a finite number of channels or bands. A video or image sequence consists of *frames* $I(.,.,t)$, for $t = 1, 2, \ldots, T$, all being images on the same carrier Ω.

Example 1.2.1 Three Examples *In case of an RGB color image $I = (R, G, B)$, we have pixels $(p, I(p)) = (p, R(p), G(p), B(p))$.*

A geometrically rectified gray-level stereo image or frame $I = (L, R)$ consists of two channels L and R, usually called left *and* right *images; this is implemented in the multi-picture object (mpo) format for images (CIPA 2009).*

For a sequence of gray-level stereo images, we have pixel $(p, t, L(p, t), R(p, t))$ in frame t, which is the combined representation of pixels $(p, t, L(p, t))$ and $(p, t, R(p, t))$ in $L(.,.,t)$ and $R(.,.,t)$, respectively, at pixel location p and time t.

1.2.1.1 Gauss Function

The zero-mean *Gauss function* is defined as follows:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{1.2}$$

A convolution of an image I with the Gauss function produces smoothed images

$$L(p, \sigma) = [I \star G_\sigma](p) \tag{1.3}$$

also known as *Gaussians*, for $\sigma > 0$. (We stay with symbol L here as introduced by Lindeberg (1994) for "layer"; a given context will prevent confusion with the left image L of a stereo pair.)

1.2.1.2 Edges

Step-edges in images are detected based on first- or second-order derivatives, such as values of the *gradient* ∇I or the *Laplacian* ΔI given by

$$\nabla I = \mathbf{grad}\ I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]^\top \quad \text{or} \quad \Delta I = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{1.4}$$

Local maxima of L_1- or L_2-magnitudes $\|\nabla I\|_1$ or $\|\nabla I\|_2$, or zero-crossings of values ΔI are taken as an indication for a step-edge. The gradient or Laplacian is commonly preceded by smoothing, using a convolution with the zero-mean Gauss function.

Alternatively, *Phase-congruency edges* in images are detected based on local frequency-space representations (Kovesi 1993).

1.2.1.3 Corners

Let I_{xx}, I_{xy}, I_{yx}, and I_{yy} denote the second-order derivatives of image I. *Corners* in images are localized based on high curvature of intensity values, to be identified by two large eigenvalues of the *Hessian matrix*

$$\mathbf{H}(p) = \begin{bmatrix} I_{xx}(p) & I_{xy}(p) \\ I_{xy}(p) & I_{yy}(p) \end{bmatrix} \tag{1.5}$$

at a pixel location p in a scalar image I (see Harris and Stephens (1988)). Figure 1.1 shows the corners detected by FAST. Corner detection is often preceded by smoothing using a convolution with the zero-mean Gauss function.

1.2.1.4 Scale Space and Key Points

Key points or *interest points* are commonly detected as maxima or minima in a $3 \times 3 \times 3$ subset of the *scale space* of a given image (Crowley and Sanderson 1987; Lindeberg 1994). A finite set of differences of Gaussians

$$D_{\sigma,a}(p) = L(p, \sigma) - L(p, a\sigma) \tag{1.6}$$

produces a *DoG scale space*. These differences are approximations to Laplacians of increasingly smoothed versions of an image (see Figure 1.5 for an example of such Laplacians forming an *LoG scale space*).

1.2.1.5 Features

An *image feature* is finally a *location* (an *interest point*), defined by a key point, edge, corner, and so on, together with a *descriptor*, usually given as a data vector (e.g., in case of scale-invariant feature transform (SIFT) of length 128 representing local gradients), but possibly also in other formats such as a graph. For example, the descriptor of a step-edge can be mean and variance of gradient values along the edge, and the descriptor of a corner can be defined by the eigenvalues of the Hessian matrix.

1.2.2 Cameras

We have an $X_wY_wZ_w$ *world coordinate system*, which is not defined by a particular camera or other sensor, and a *camera coordinate system* $X_sY_sZ_s$ (index "s" for "sensor"), which is described with respect to the chosen world coordinates by means of an affine transform, defined by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$.

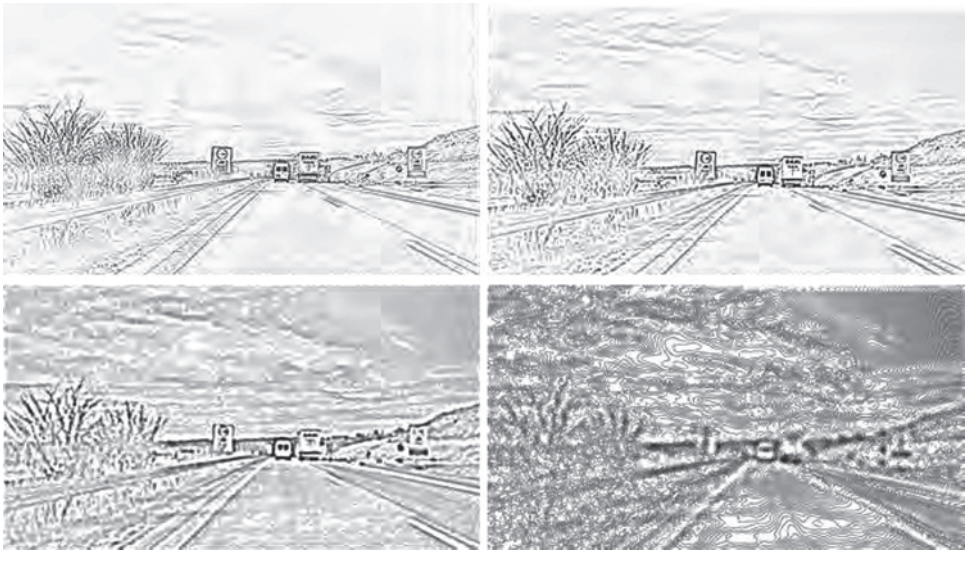


Figure 1.5 Laplacians of smoothed copies of the same image using `cv::GaussianBlur` and `cv::Laplacian` in `OpenCV`, with values 0.5, 1, 2, and 4, for parameter σ for smoothing. Linear scaling is used for better visibility of the resulting *Laplacians*. Courtesy of Sandino Morales

A point in 3D space is given as $P_w = (X_w, Y_w, Z_w)$ in world coordinates or as $P_s = (X_s, Y_s, Z_s)$ in camera coordinates. In addition to the *coordinate notation* for points, we also use *vector notation*, such as $P_w = [X_w, Y_w, Z_w]^T$ for point P_w.

1.2.2.1 Pinhole-type Camera

The Z_s-axis models the *optical axis*. Assuming an ideal pinhole-type camera, we can ignore radial distortion and can have *undistorted projected points* in the image plane with coordinates x_u and y_u. The distance f between the $x_u y_u$ image plane and the projection center is the *focal length*.

A visible point $P = (X_s, Y_s, Z_s)$ in the world is mapped by *central projection* into pixel location $p = (x_u, y_u)$ in the undistorted image plane:

$$x_u = \frac{fX_s}{Z_s} \quad \text{and} \quad y_u = \frac{fY_s}{Z_s} \tag{1.7}$$

with the origin of $x_u y_u$ image coordinates at the intersection point of the Z_s-axis with the image plane.

The intersection point (c_x, c_y) of the optical axis with the image plane in xy coordinates is called the *principal point*. It follows that $(x, y) = (x_u + c_x, y_u + c_y)$. A pixel location (x, y) in the 2D xy image coordinate system has 3D coordinates $(x - c_x, y - c_y, f)$ in the $X_s Y_s Z_s$ camera coordinate system.

1.2.2.2 Intrinsic and Extrinsic Parameters

Assuming multiple cameras C_i, for some indices i (e.g., just C_L and C_R for binocular stereo), camera calibration specifies intrinsic parameters such as edge lengths e_x^i and e_y^i of camera sensor cells (defining the aspect ratio), a skew parameter s^i, coordinates of the principal point $\mathbf{c}^i = (c_x^i, c_y^i)$ where optic axis of camera i and image plane intersect, the focal length f^i, possibly refined as f_x^i and f_y^i, and lens distortion parameters starting with κ_1^i and κ_2^i. In general, it can be assumed that lens distortion has been calibrated before and does not need to be included anymore in the set of intrinsic parameters. Extrinsic parameters are defined by rotation matrices and translation vectors, for example, matrix $\mathbf{R}^{ij}$ and vector $\mathbf{t}^{ij}$ for the affine transform between camera coordinate systems $X_s^i Y_s^i Z_s^i$ and $X_s^j Y_s^j Z_s^j$, or matrix $\mathbf{R}^i$ and vector $\mathbf{t}^i$ for the affine transform between camera coordinate system $X_s^i Y_s^i Z_s^i$ and $X_w Y_w Z_w$.

1.2.2.3 Single-Camera Projection Equation

The camera projection equation in homogeneous coordinates, mapping a 3D point $P = (X_w, Y_w, Z_w)$ into image coordinates $p^i = (x^i, y^i)$ of the ith camera, is as follows:

$$k \begin{bmatrix} x^i \\ y^i \\ 1 \end{bmatrix} = \begin{bmatrix} f^i/e_x^i & s^i & c_x^i & 0 \\ 0 & f^i/e_y^i & c_y^i & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}^i & -[\mathbf{R}^i]^\top \mathbf{t}^i \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{1.8}$$

$$= [\mathbf{K}^i | \mathbf{0}] \cdot \mathbf{A}^i \cdot [X_w, Y_w, Z_w, 1]^\top \tag{1.9}$$

where $k \neq 0$ is a scaling factor. This defines a 3×3 matrix $\mathbf{K}^i$ of intrinsic camera parameters and a 4×4 matrix $\mathbf{A}^i$ of extrinsic parameters (of the affine transform) of camera i. The 3×4 *camera matrix* $\mathbf{C}^i = [\mathbf{K}^i | \mathbf{0}] \cdot \mathbf{A}^i$ is defined by 11 parameters if we allow for an arbitrary scaling of parameters; otherwise it is 12.

1.2.3 Optimization

We specify one popular optimization strategy that has various applications in computer vision. In an abstract sense, we assign to each pixel a *label* l (e.g., an optical flow vector $\mathbf{u}$, a disparity d, a segment identifier, or a surface gradient) out of a set L of possible labels (e.g., all vectors pointing from a pixel p to points in a Euclidean distance to p of less than a given threshold). Labels $(u, v) \in \mathbb{R}^2$ are thus in the 2D continuous plane.

1.2.3.1 Optimizing a Labeling Function

Labels are assigned to all the pixels in the carrier Ω by a *labeling function* $f : \Omega \to L$. Solving a labeling problem means to identify a labeling f that approximates somehow an optimum of a defined *error* or *energy*

$$E_{\text{total}}(f) = E_{\text{data}}(f) + \lambda \cdot E_{\text{smooth}}(f) \tag{1.10}$$

where $\lambda > 0$ is a weight. Here, $E_{\text{data}}(f)$ is the *data-cost term* and $E_{\text{smooth}}(f)$ is the *smoothness-cost term*. A decrease in λ works toward reduced smoothing of calculated labels. Ideally, we search for an optimal (i.e., of minimal total error) f in the set of all possible labelings, which defines a *total variation* (TV).

We detail Eq. (1.10) by adding costs at pixels. In a current image, label $f_p = f(p)$ is assigned by the value of labeling function f at pixel position p. Then we have that

$$E_{\text{total}}(f) = \sum_{p \in \Omega} E_{\text{data}}(p, f_p) + \lambda \cdot \sum_{p \in \Omega} \sum_{q \in A(p)} E_{\text{smooth}}(f_p, f_q) \tag{1.11}$$

where A is an adjacency relation between pixel locations.

In optical flow or stereo vision, label f_p (i.e., optical flow vector or disparity) defines a pixel q in another image (i.e., in the following image, or in the left or right image of a stereo pair); in this case, we can also write $E_{\text{data}}(p, q)$ instead of $E_{\text{data}}(p, f_p)$.

1.2.3.2 Invalidity of the Intensity Constancy Assumption

Data-cost terms are defined for windows that are centered at the considered pixel locations. The data in both windows, around the start pixel location p, and around the pixel location q in the other image, are compared for understanding "data similarity."

For example, in the case of stereo matching, we have $p = (x, y)$ in the right image R and $q = (x + d, y)$ in the left image L, for disparity $d \geq 0$, and the data in both $(2k + 1) \times (2k + 1)$ windows are identical if and only if the data-cost measure

$$E_{\text{SSD}}(p, d) = \sum_{i=-l}^{l} \sum_{j=-k}^{k} [R(x + i, y + j) - L(x + d + i, y + j)]^2 \tag{1.12}$$

results in value 0, where SSD stands for *sum of squared differences*.

The use of such a data-cost term would be based on the *intensity constancy assumption* (ICA), that is, intensity values around corresponding pixel locations p and q are (basically) identical within a window of specified size. However, the ICA is invalid for real-world recording. Intensity values at corresponding pixels and in their neighborhoods are typically impacted by lighting variations, or just by image noise. There are also impacts of differences in local surface reflectance, differences in cameras when comparing images recorded by different cameras, or effects of perspective distortion (the local neighborhood around a surface point is differently projected into different cameras). Thus, energy optimization needs to apply better data measures compared to SSD, or other measures are also defined based on the ICA.

1.2.3.3 Census Data-Cost Term

The census-cost function has been identified as being able to compensate successfully bright variations in input images of a recorded video (Hermann and Klette 2009;

Hirschmüller and Scharstein 2009). The *mean-normalized census-cost function* is defined by comparing a $(2l+1)\times(2k+1)$ window centered at pixel location p in frame I_1 with a window of the same size centered at a pixel location q in frame I_2. Let $\bar{I}_i(p)$ be the mean of the window around p for $i=1$ or $i=2$. Then we have that

$$E_{\mathrm{MCEN}}(p,q)=\sum_{i=-l}^{l}\sum_{j=-k}^{k}\rho_{ij} \tag{1.13}$$

with

$$\rho_{ij}=\begin{cases}0 & I_1(p+(i,j))<\bar{I}_1(p) \text{ and } I_2(q+(i,j))<\bar{I}_2(q)\\ & \text{or}\quad I_1(p+(i,j))>\bar{I}_1(p) \text{ and } I_2(q+(i,j))>\bar{I}_2(q)\\ 1 & \text{otherwise}\end{cases} \tag{1.14}$$

Note that value 0 corresponds to consistency in both comparisons. If the comparisons are performed with respect to values $I_1(p)$ and $I_2(q)$, rather than the means $\bar{I}_1(p)$ and $\bar{I}_2(q)$, then we have the *census-cost function* $E_{\mathrm{CEN}}(p,q)$ as a candidate for a data-cost term.

Let $\mathbf{a}_p$ be the vector listing results $\mathrm{sgn}(I_1(p+(i,j))-\bar{I}_1(p))$ in a left-to-right, top-to-bottom order (with respect to the applied $(2l+1)\times(2k+1)$ window), where sgn is the signum function; $\mathbf{b}_q$ lists values $\mathrm{sgn}(I_2(q+(i,j))-\bar{I}_2(q))$. The mean-normalized census data-cost $E_{\mathrm{MCEN}}(p,q)$ equals the Hamming distance between vectors $\mathbf{a}_p$ and $\mathbf{b}_q$.

1.3 Visual Tasks

This section briefly outlines some of the visual tasks that need to be solved by computer vision in vehicles.

1.3.1 Distance

Laser range-finders are increasingly used for estimating distance mainly based on the time-of-flight principle. Assuming sensor arrays of larger density in the near future, laser range-finders will become a standard option for cost-efficient accurate distance calculations. Combining stereo vision with distance data provided by laser range-finders is a promising multi-module approach toward distance calculations.

Stereo vision is the dominant approach in computer vision for calculating distances. *Corresponding pixels* are here defined by projections of the same surface point in the scene into the left and right images of a stereo pair. After having recorded stereo pairs rectified into canonical stereo geometry, one-dimensional (1D) correspondence search can be limited to identical image rows.

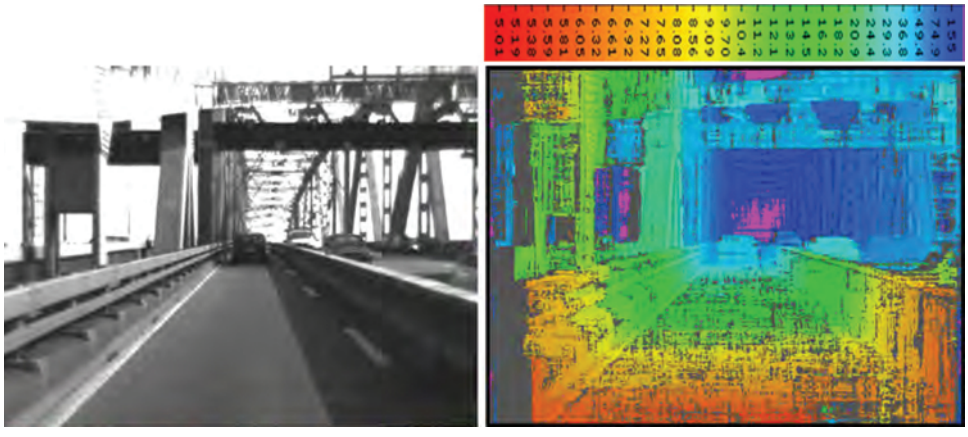


Figure 1.6 (a) Image of a stereo pair (from a test sequence available on EISATS). (b) Visualization of a depth map using the color key shown at the top for assigning distances in meters to particular colors. A pixel is shown in gray if there was low confidence for the calculated disparity value at this pixel. Courtesy of Simon Hermann

1.3.1.1 Stereo Vision

We address the detection of corresponding points in a stereo image $I = (L, R)$, a basic task for distance calculation in vehicles using binocular stereo.

Corresponding pixels define a *disparity*, which is mapped based on camera parameters into *distance* or *depth*. There are already very accurate solutions for stereo matching, but challenging input data (rain, snow, dust, sunstroke, running wipers, and so forth) still pose unsolved problems (see Figure 1.6 for an example of a depth map).

1.3.1.2 Binocular Stereo Vision

After camera calibration, we have two virtually identical cameras C^L and C^R, which are perfectly aligned defining *canonical stereo geometry*. In this geometry, we have an identical copy of the camera on the left translated by *base distance b* along the X_s-axis of the $X_sY_sZ_s$ camera coordinate system of the left camera. The projection center of the left camera is at $(0, 0, 0)$ and the projection center of the cloned right camera is at $(b, 0, 0)$. A 3D point $P = (X_s, Y_s, Z_s)$ is mapped into undistorted image points

$$p_u^L = (x_u^L, y_u^L) = \left(\frac{f \cdot X_s}{Z_s}, \frac{f \cdot Y_s}{Z_s} \right) \tag{1.15}$$

$$p_u^R = (x_u^R, y_u^R) = \left(\frac{f \cdot (X_s - b)}{Z_s}, \frac{f \cdot Y_s}{Z_s} \right) \tag{1.16}$$

in the left and right image planes, respectively. Considering p_u^L and p_u^R in homogeneous coordinates, we have that

$$[p_u^R]^\top \cdot \mathbf{F} \cdot p_u^L = 0 \tag{1.17}$$

for the 3×3 *bifocal tensor* $\mathbf{F}$, defined by the configuration of the two cameras. The dot product $\mathbf{F} \cdot p_u^L$ defines an *epipolar line* in the image plane of the right camera; any stereo point corresponding to p_u^L needs to be on that line.

1.3.1.3 Binocular Stereo Matching

Let B be the *base image* and M be the *match image*. We calculate corresponding pixels p^B and q^M in the xy image coordinates of carrier Ω following the optimization approach as expressed by Eq. (1.11). A labeling function f assigns a disparity f_p to pixel location p, which specifies a corresponding pixel $q = p^f$.

For example, we can use the census data-cost term $E_{\text{MCEN}}(p, p^f)$ as defined in Eq. (1.13), and for the smoothness-cost term, either the Potts model, linear truncated cost, or quadratic truncated costs is used (see Chapter 5 in Klette (2014)). Chapter 6 of Klette (2014) discusses also different algorithms for stereo matching, including *belief-propagation matching* (BPM) (Sun et al. 2003) and *dynamic-programming stereo matching* (DPSM). DPSM can be based on scanning along the epipolar line only using either an ordering or a smoothness constraint, or it can be based (for symmetry?) on scanning along multiple scanlines using a smoothness constraint along those lines; the latter case is known as *semi-global matching* (SGM) if multiple scanlines are used for error minimization (Hirschmüller, 2005). A variant of SGM is used in Daimler's stereo vision system, available since March 2013 in their Mercedes cars (see also Chapter 2 by U. Franke in this book).

Iterative SGM (iSGM) is an example for a modification of baseline SGM; for example, error minimization along the horizontal scanline should in general contribute more to the final result than optimization along other scanlines (Hermann and Klette, 2012). Figure 1.7 also addresses confidence measurement; for a comparative discussion of confidence measures, see Haeusler and Klette (2012). *Linear BPM* (linBPM) applies the MCEN data-cost term and the linear truncated smoothness-cost term (Khan et al. 2013).

1.3.1.4 Performance Evaluation of Stereo Vision Solutions

Figure 1.8 provides a comparison of iSGM to linBPM on four frame sequences each of 400 frames length. It illustrates that iSGM performs better (with respect to the used measure, see the following section for its definition) on the `bridge` sequence that is characterized by many structural details in the scene, but not as good as linBPM on the other three sequences. For sequences `dusk` and `midday`, both performances are highly correlated, but not for the other two sequences. Of course, evaluating on only

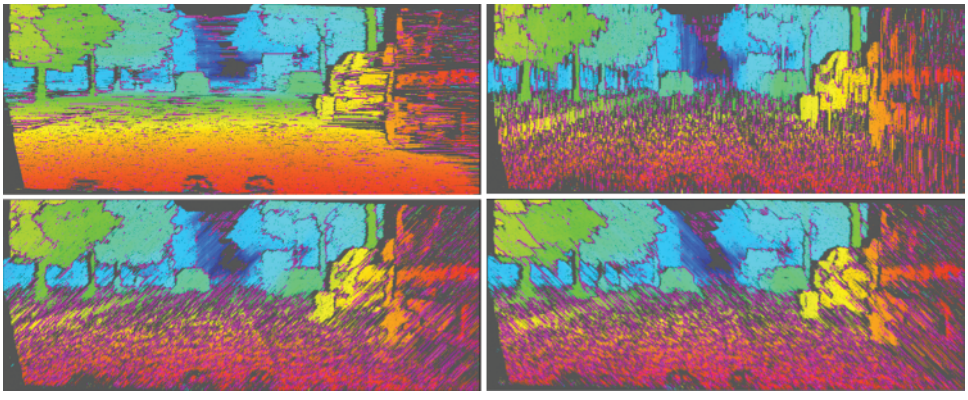


Figure 1.7 Resulting disparity maps for stereo data when using only *one* scanline for DPSM with the SGM smoothness constraint and a 3×9 MCEN data-cost function. *From top to bottom and left to right*: Left-to-right horizontal scanline, and lower-left to upper-right diagonal scanline, top-to-bottom vertical scanline, and upper-left to lower-right diagonal scanline. Pink pixels are for low-confidence locations (here identified by inhomogeneous disparity locations). Courtesy of Simon Hermann; the input data have been provided by Daimler A.G.

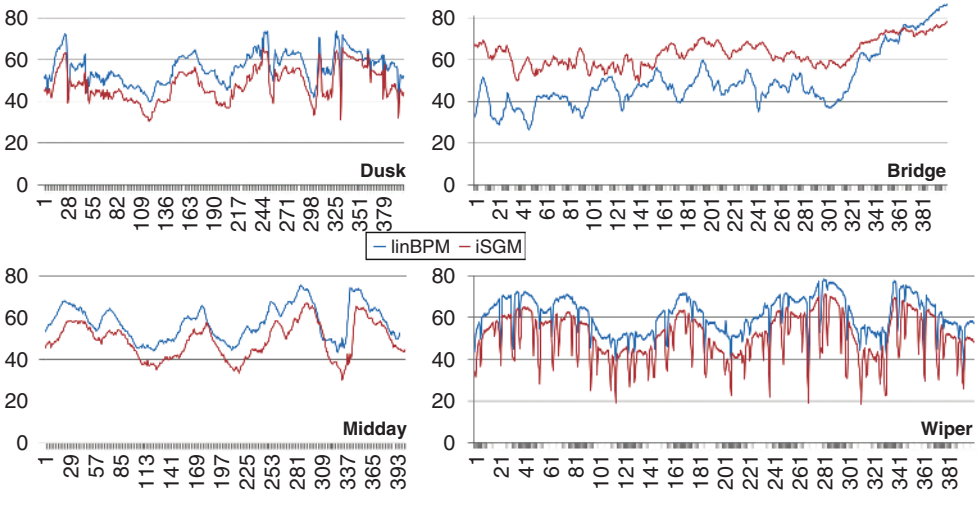


Figure 1.8 Normalized cross-correlation results when applying the third-eye technology for stereo matchers iSGM and linBPM for four real-world trinocular sequences of Set 9 of EISATS. Courtesy of Waqar Khan, Veronica Suaste, and Diego Caudillo

a few sequences of 400 frames each is insufficient for making substantial evaluations, but it does illustrate performance.

The diagrams in Figure 1.8 are defined by the *normalized cross-correlation* (NCC) between a recorded third-frame sequence and a virtual sequence calculated based on

the stereo matching results of two other frame sequences. This *third-eye technology* (Morales and R 2009) also uses masks such that only image values are compared which are close to step-edges (e.g., see Figure 1.5 for detected edges at bright pixels in LoG scale space) in the third frame. It enables us to evaluate performance on any calibrated trinocular frame sequence recorded in the real world.

Example 1.3.1 Environment Reconstruction *3D road-side visualization or 3D environment modeling is the application where a 3D reconstruction from a moving platform can be used (Xiao et al. 2009), possibly in combination with 3D reconstructions from a flying platform such as a multi-copter.*

There are unresolved issues in the required very high accuracy of ego-motion analysis for mapping 3D results obtained at time t in a uniform world coordinate system. This is in particular apparent when trying to unify results from different runs through the same street (Zeng and Klette 2013). Figure 1.9 shows the 3D results from a single run (for a site at Tamaki campus, Auckland).

Figure 1.9 (a) Reconstructed cloud of points. (b) Reconstructed surface based on a single run of the ego-vehicle. Courtesy of Yi Zeng

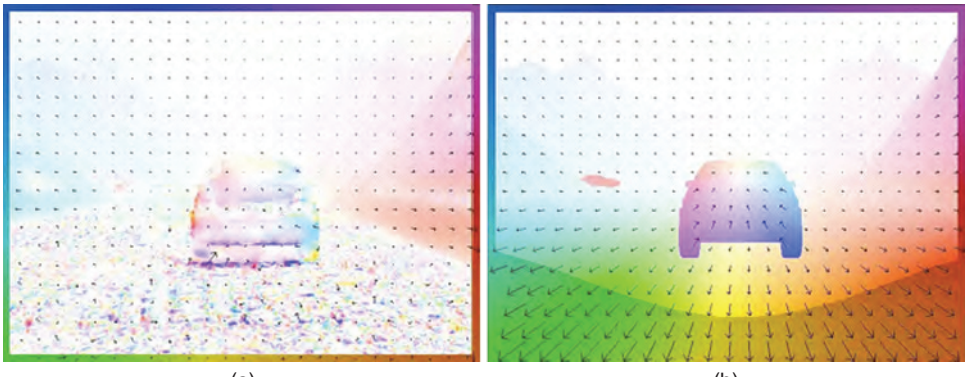
1.3.2 Motion

A sequence of video frames $I(.,.,t)$, all defined on the same carrier Ω, is recorded with a time difference δt between two subsequent frames; frame t is recorded at time $t \cdot \delta t$ counted from the start of the recording.

The projection of a static or moving surface point into pixel $p_t = (x_t, y_t)$ in frame t and into pixel $p_{t+1} = (x_{t+1}, y_{t+1})$ in frame $t + 1$ defines a pair of *corresponding pixels* represented by a *motion vector* $[x_{t+1} - x_t, y_{t+1} - y_t]^\top$ from p_t to p_{t+1} in Ω.

1.3.2.1 Dense or Sparse Motion Analysis

Dense motion analysis aims at calculating approximately correct motion vectors for "basically" every pixel location $p = (x, y)$ in frame t (see Figure 1.10 for an example).



(a) (b)

Figure 1.10 Visualization of optical flow using the color key shown around the border of the image for assigning a direction to particular colors; the length of the flow vector is represented by saturation, where value "white" (i.e., undefined saturation) corresponds to "no motion." (a) Calculated optical flow using the original Horn–Schunck algorithm. (b) Ground truth for the image shown in Figure 1.4a. Courtesy of Tobi Vaudrey

Sparse motion analysis is designed for having accurate motion vectors at a few selected pixel locations.

Motion analysis is a difficult 2D correspondence problem, and it might become easier by having recorded high-resolution images at a higher frame rate in future. For example, motion analysis is approached by a single-module solution by *optical flow* calculation, or as a multi-module solution when combining image segmentation with subsequent estimations of motion vectors for image segments.

1.3.2.2 Optical Flow

Optical flow $\mathbf{u}(p,t) = [u(p,t), v(p,t)]^\top$ is the result of dense motion analysis. It represents motion vectors between corresponding pixels $p = (x, y)$ in frames $I(.,.,t)$ and $I(.,.,t+1)$. Figure 1.10 shows the visualization of an optical flow map.

1.3.2.3 Optical Flow Equation and Image Constancy Assumption

The derivation of the *optical flow equation* (Horn and Schunck 1981)

$$0 = u(p,t) \cdot I_x(p,t) + v(p,t) \cdot I_y(p,t) + I_t(p,t) \tag{1.18}$$

for $p \in \Omega$ and first-order derivatives I_x, I_y, and I_t follows from the ICA, that is, by assuming that corresponding 3D world points are represented in frame t and $t+1$ by the same intensity. This is actually not true for computer vision in vehicles. Light intensities change frequently due to lighting artifacts (e.g., driving below trees), changing angles to the Sun, or simply due to sensor noise. However, the optical flow

equation is often used as a *data-cost term* in an optimization approach (minimizing energy as defined in Eq. (1.10)) for solving the optical flow problem.

1.3.2.4 Examples of Data and Smoothness Costs

If we accept Eq. (1.18) due to Horn and Schunck (and thus the validity of the ICA) as data constraint, then we derive

$$E_{\mathrm{HS}}(f) = \sum_{p\in\Omega} [\ u(p,t)\cdot I_x(p,t) + v(p,t)\cdot I_y(p,t) + I_t(p,t)\ ]^2 \tag{1.19}$$

as a possible data-cost term for any given time t.

We introduced above the zero-mean-normalized census-cost function E_{MCEN}. The sum $E_{\mathrm{MCEN}}(f) = \sum_{p\in\Omega} E_{\mathrm{MCEN}}(p, p+f(p))$ can replace $E_{\mathrm{HS}}(p,q)$ in an optimization approach as defined by Eq. (1.10) (see Hermann and Werner (2013)). This corresponds to the invalidity of the ICA for video data recorded in the real world.

For the smoothness-error term, we may use

$$E_{\mathrm{FO\_L_2}}(f) = \sum_{p\in\Omega} \left[\frac{\partial u(p,t)}{\partial x}\right]^2 + \left[\frac{\partial u(p,t)}{\partial y}\right]^2 + \left[\frac{\partial v(p,t)}{\partial x}\right]^2 + \left[\frac{\partial v(p,t)}{\partial y}\right]^2 \tag{1.20}$$

This smoothness-error term applies squared penalties to first-order derivatives in the L_2 sense. Applying a smoothness term in an approximate L_1 sense reduces the impact of outliers (Brox et al. 2004).

Terms E_{HS} and $E_{\mathrm{FO\_L_2}}$ define the TVL_2 optimization problem as originally considered by Horn and Schunck (1981).

1.3.2.5 Performance Evaluation of Optical Flow Solutions

Apart from using data with provided ground truth (see EISATS and KITTI and Figure 1.4), there is also a way for evaluating calculated flow vectors on recorded real-world video assuming that the recording speed is sufficiently large; for calculated flow vectors for frames $I(.,.,t)$ and $I(.,.,t+2)$, we calculate an image "half-way" using the mean of image values at corresponding pixels and we compare this calculated image with frame $I(.,.,t+1)$ (see Szeliski (1999)). Limitations for recording frequencies of current cameras make this technique not yet practically appropriate, but it is certainly appropriate for fundamental research.

1.3.3 Object Detection and Tracking

In general, an *object detector* is defined by applying a classifier for an object detection problem. We assume that any decision made can be evaluated as being either correct or false.

1.3.3.1 Measures for Object Detection

Let tp or fp denote the numbers of true-positives or false-positives, respectively. Analogously we define tn and fn for the negatives; tn is not a common entry for performance measures.

Precision (PR) is the ratio of true-positives compared to all detections. *Recall* (RC) (or *sensitivity*) is the ratio of true-positives compared to all potentially possible detections (i.e., to the number of all visible objects):

$$\mathrm{PR} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}} \qquad \text{and} \qquad \mathrm{RC} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}} \tag{1.21}$$

The *miss rate* (MR) is the ratio of false-negatives compared to all objects in an image. *False-positives per image* (FPPI) is the ratio of false-positives compared to all detected objects in an image:

$$\mathrm{MR} = \frac{\mathrm{fn}}{\mathrm{tp} + \mathrm{fn}} \qquad \text{and} \qquad \mathrm{FPPI} = \frac{\mathrm{fp}}{\mathrm{tp} + \mathrm{fp}} \tag{1.22}$$

In case of multiple images, the mean of measures can be used (i.e., averaged over all the processed images).

How to decide whether a detected object is true-positive? Assume that objects in images have been locally identified manually by bounding boxes, serving as the ground truth. All detected objects are matched with these *ground-truth boxes* by calculating ratios of areas of overlapping regions

$$a_o = \frac{\mathcal{A}(D \cap T)}{\mathcal{A}(D \cup T)} \tag{1.23}$$

where $\mathcal{A}$ denotes the area of a region in an image, D is the detected bounding box of the object, and T is the area of the bounding box of the matched ground-truth box. If a_o is larger than a threshold T, say $T = 0.5$, then the detected object is taken as a true-positive.

Example 1.3.2 Driver Monitoring *Besides measurements for understanding the steadiness of driver's movement of the steering wheel, cameras are also an appropriate tool for understanding the state of the driver (e.g., drowsiness detection, or eye gaze).*

Face and eye detection (Viola and Jones 2001b) or head pose analysis (Murphy-Chutorian and Trivedi 2009) are basic tasks in this area (Figure 1.11). Challenging lighting conditions still define unsatisfactorily solved scenarios (e.g., see Rezaei and Klette (2012) for such scenarios).

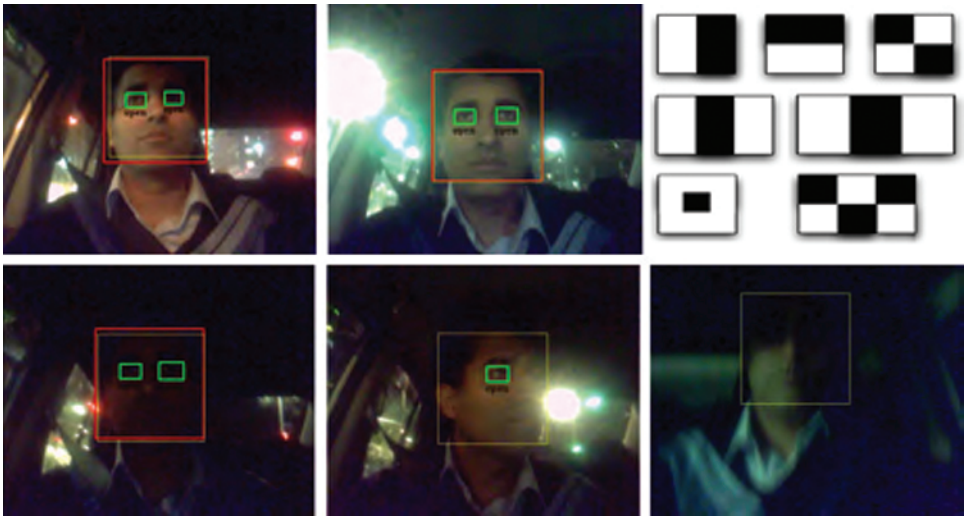


Figure 1.11 Face detection, eye detection, and face tracking results under challenging lighting conditions. Typical Haar-like features, as introduced in Viola and Jones (2001b), are shown in the upper right. The illustrated results for challenging lighting conditions require additional efforts. Courtesy of Mahdi Rezaei

Driver awareness *can be defined by relating driver monitoring results to environment analysis for the given traffic scenario. The driver not only needs to pay attention to driving, eye gaze or head pose (Rezaei and Klette 2011) should also correspond (for some time) to those outside regions where safety-related events occur.*

1.3.3.2 Object Tracking

Object tracking is an important task for understanding the motion of a mobile platform or of other objects in a dynamic environment. The mobile platform with the installed system is also called the *ego-vehicle* whose *ego-motion* needs to be calculated for understanding the movement of the installed sensors in the three-dimensional (3D) world.

Calculated features in subsequent frames $I(.,.,t)$ can be tracked (e.g., by using RANSAC for identifying an affine transform between feature points) and then used for estimating ego-motion based on bundle adjustment. This can also be combined with another module using nonvisual sensor data such as GPS or of an inertial measurement unit (IMU). For example, see Geng et al. (2015) for an integration of GPS data.

Other moving objects in the scene can be tracked using repeated detections or by following a detected object in frame $I(.,.,t)$ to frame $I(.,.,t+1)$. A *Kalman filter* (e.g., linear, general, or unscented) can be used for building a model for the motion as well as for involved noise. A *particle filter* can also be used based on extracted weights for potential moves of a particle in particle space. Kalman and particle filters are introduced, with references to related original sources, in Klette (2014).

1.3.4 Semantic Segmentation

When segmenting a scene, ideally obtained segments should correspond to defined objects in the scene, such as a house, a person, or a car in a road scene. These segments define *semantic segmentation*. Segmentation for vehicle technology aims at semantic segmentation (Floros and Leibe 2012; Ohlich et al. 2012) with temporal consistency along a recorded video sequence. Appearance is an important concept for semantic segmentation (Mohan 2014). The concept of super pixels (see, e.g., Liu et al. (2012)) might be useful for achieving semantic segmentation. Temporal consistency requires tracking of segments and similarity calculations between tracked segments.

1.3.4.1 Environment Analysis

There are *static* (i.e., fixed with respect to the Earth) or *dynamic* objects in a scenario which need to be detected, understood, and possibly further analyzed.

A flying helicopter (or just multi-copter) should be able to detect power lines or other potential objects defining a hazard. Detecting traffic signs or traffic lights, or understanding lane borders of highways or suburban roads are examples for driving vehicles. Boats need to detect buoys and beacons.

Pedestrian detection became a common subject for road-analysis projects. After detecting a pedestrian on a pathway next to an inner-city road, it would be helpful to understand whether this pedestrian intends to step onto the road in the next few seconds.

After detecting more and more objects, we may have the opportunity to model and understand a given environment.

Example 1.3.3 Use of Stereo Analysis and Optical Flow Calculations *Modules for solving stereo matching and optical flow calculation can be used for designing a system for video segmentation. For example, following Hermann et al. (2011), stereo matching for images $L(.,.,t)$ and $R(.,.,t)$ of frame t results in a depth map that is segmented by*

1. *preprocessing for removing noisy (i.e., isolated) depth values and irrelevant depth values (e.g., in the sky region),*
2. *estimating a ground manifold using* v-disparities*—depth values identified as being in the ground manifold are also removed—and*
3. *performing a segmentation procedure (e.g., simple region growing) on the remaining depth values.*

Resulting segments are likely to be of similar shape and location as those obtained for stereo frame $t + 1$ by the same procedure. For each segment obtained for frame t, the mean optical flow vector for pixels in this segment defines the expected move of this segment into a new position in frame $t + 1$. Those expected segments (of frame t after expected moves into frame $t + 1$) are compared with the actual segments of frame $t + 1$ for identifying correspondences, for example, by applying a set-theoretical metric, which represents the ratio between overlap and total area of both segments.

1.3.4.2 Performance Evaluation of Semantic Segmentation

There is a lack of provided ground truth for semantic segmentations in traffic sequences. Work reported in current publications on semantic segmentation, such as Floros and Leibe (2012) and Ohlich et al. (2012), can be used for creating test databases. There is also current progress in available online data; see www.cvlibs.net/datasets/kitti/eval_road.php, www.cityscapes-dataset.net, and (Ros et al. 2015) for a study for such data.

Barth et al. (2010) proposed a method for segmentation, which is based on evaluating pixel probabilities of whether they are in motion in the real world or not (using scene flow and ego-motion). Barth et al. (2010) also provides ground truth for image segmentation in Set 7 of EISATS, illustrated by Figure 1.12. Figure 1.12 also shows resulting SGM stereo maps and segments obtained when following the multi-module approach briefly sketched earlier.

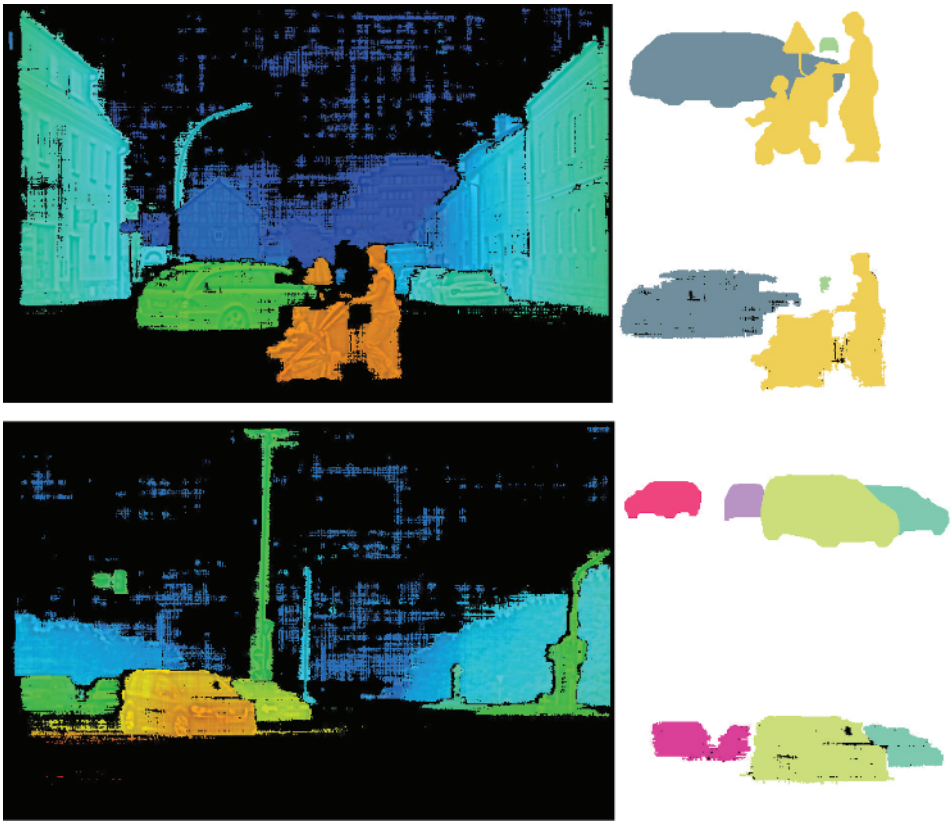


Figure 1.12 Two examples for Set 7 of EISATS illustrated by preprocessed depth maps following the described method (Steps 1 and 2). Ground truth for segments is provided by Barth et al. (2010) and shown on top in both cases. Resulting segments using the described method are shown below in both cases; courtesy of Simon Hermann

Modifications in the involved modules for stereo matching and optical flow calculation influence the final result. There might be dependencies between performances of contributing programs.

1.4 Concluding Remarks

The vehicle industry worldwide has assigned major research and development resources for offering competitive solutions for vision-based components for vehicles. Research at academic institutions needs to address future or fundamental tasks, challenges that are not of immediate interest for the vehicle industry, for being able to continue to contribute to this area.

The chapter introduced basic notation and selected visual tasks. It reviewed work in the field of computer vision in vehicles. There are countless open questions in this area, often related to

1. adding further alternatives to only a few existing robust solutions for one generic or specific task,
2. a comparative evaluation of such solutions,
3. ways of analyzing benchmarks for their particular challenges,
4. the design of more complex systems, and
5. ways to test such complex systems.

Specifying and solving a specific task might be a good strategy to define fundamental research, ahead of currently extremely intense industrial research and development within the area of computer vision for vehicles. Aiming at robustness including challenging scenarios and understanding interactions in dynamic scenes between multiple moving objects are certainly examples where further research is required.

Computer vision can help to solve true problems in society or industry, thus contributing to the prevention of social harms or atrocities; it is a fundamental ethical obligation of researchers in this field not to contribute to those, for example, by designing computer vision solutions for the use in UAVs for killing people. Academics identify *ethics in research* often with subjects such as plagiarism, competence, or objectivity, and a main principle is also social responsibility. Computer vision in road vehicles can play, for example, a major role in reducing casualties in traffic accidents, which are counted by hundreds of thousands of people worldwide each year; it is a very satisfying task for a researcher to contribute to improved road safety.

Acknowledgments

The author thanks Simon Hermann, Mahdi Rezaei, Konstantin Schauwecker, Junli Tao, and Garry Tee for comments on drafts of this chapter.