Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 29858, 17 pages
doi:10.1155/2007/29858

Research Article

Adaptive Probabilistic Tracking Embedded in Smart Cameras for Distributed Surveillance in a 3D Model

Sven Fleck, Florian Busch, and Wolfgang Straßer

Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, Sand 14, 72076 Tübingen, Germany

Received 27 April 2006; Revised 10 August 2006; Accepted 14 September 2006

Recommended by Moshe Ben-Ezra

Tracking applications based on distributed and embedded sensor networks are emerging today, both in the fields of surveillance and industrial vision. Traditional centralized approaches have several drawbacks, due to limited communication bandwidth, computational requirements, and thus limited spatial camera resolution and frame rate. In this article, we present network-enabled smart cameras for probabilistic tracking. They are capable of tracking objects adaptively in real time and offer a very bandwidthconservative approach, as the whole computation is performed embedded in each smart camera and only the tracking results are transmitted, which are on a higher level of abstraction. Based on this, we present a distributed surveillance system. The smart cameras' tracking results are embedded in an integrated 3D environment as live textures and can be viewed from arbitrary perspectives. Also a georeferenced live visualization embedded in Google Earth is presented.

1. INTRODUCTION

In typical computer vision systems today, cameras are seen only as simple sensors. The processing is performed after transmitting the complete raw sensor stream via a costly and often distance-limited connection to a centralized processing unit (PC). We think it is more natural to also embed the processing in the camera itself: what algorithmically belongs *to* the camera is also physically performed *in* the camera. The idea is to compute the information where it becomes available—directly at the sensor—and transmit only results that are on a higher level of abstraction. This follows the emerging trend of self-contained and networking capable smart cameras.

Although it could seem obvious to experts in the computer vision field, that a smart camera approach brings various benefits, the state-of-the-art surveillance systems in industry still prefer centralized, server-based approaches instead of maximally distributed solutions. For example, the surveillance system installed in London and soon to be installed in New York consists of over 200 cameras, each sending a 3.8 Mbps video stream to a centralized processing center consisting of 122 servers [1].

The contribution of this paper is not a new smart camera or a new tracking algorithm or any other isolated component of a surveillance system. Instead, it will demonstrate both the idea of 3D surveillance which integrates the results of the tracking system in a unified, ubiquitously available 3D model using a distributed network of smart cameras, and also the system aspect that comprises the architecture and the whole computation pipeline from 3D model acquisition, camera network setup, distributed embedded tracking, and visualization, embodied in one complete system.

Tracking plays a central role for many applications including robotics (visual servoing, RoboCup), surveillance (person tracking), and also human-machine interface, motion capture, augmented reality, and 3DTV. Traditionally in surveillance scenarios, the raw live video stream of a huge number of cameras is displayed on a set of monitors, so the security personnel can respond to situations accordingly. For example, in a typical Las Vegas casino, approximately 1 700 cameras are installed [2]. If you want to track a suspect on his way, you have to manually follow him within a certain camera. Additionally, when he leaves one camera's view, you have to switch to an appropriate camera manually and put yourself in the new point of view to keep up tracking. A more

intuitive 3D visualization where the person's path tracked by a distributed network of smart cameras is integrated in one consistent world model, independent of all cameras, which is not yet available.

Imagine a distributed, intersensor surveillance system that reflects the world and its events in an integrated 3D world model which is available ubiquitously within the network, independent of camera views. This vision includes a hassle-free and automated method for acquiring a 3D model of the environment of interest, an easy plug "*n*" play style of adding new smart camera nodes to the network, the distributed tracking and person handover itself, and the integration of all cameras' tracking results in one consistent model. We present two consecutive systems to come closer to this vision.

First, in Section 2, we present a network-enabled smart camera capable of embedded probabilistic real-time object tracking in *image domain*. Due to the embedded and decentralized nature of such a vision system, besides real-time constraints, the robust and fully autonomous operation is an essential challenge, as no user interaction is available during the tracking operation. This is achieved by these concepts. Using particle filtering techniques enables the robust handling of multimodal probability density functions (pdfs) and nonlinear systems. Additionally, an adaptivity mechanism increases the robustness by adapting to slow appearance changes of the target.

In the second part of this article (Section 3), we present a complete surveillance system capable of tracking in *world model domain*. The system consists of a virtually arbitrary number of camera nodes, a server node, and a visualization node. Two kinds of visualization methods are presented: a 3D point-based rendering system, called XRT, and a live visualization plug-in for Google Earth [3]. To cover the whole system, our contribution does not stop from presenting an easy method for 3D model acquisition of both indoor and outdoor scenes as content for the XRT visualization node by the use of our mobile platform—the Wägele. Additionally, an application for self-localization in indoor and outdoor environments based on the tracking results of this distributed camera system is presented.

1.1. Related work

1.1.1. Smart cameras

A variety of smart camera architectures designed in academia [4, 5] and industry exist today. What all smart cameras share is the combination of a sensor, an embedded processing unit, and a connection, which is nowadays often a network unit. The processing means can be roughly classified in DSPs, general purpose processors, FPGAs, and a combination thereof. The idea of having Linux running embedded on the smart camera gets more and more common (Matrix Vision, Basler, Elphel).

From the other side, the surveillance sector, IP-based cameras are emerging where the primary goal is to transmit live video streams to the network by self-contained camera units with (often wireless) Ethernet connection and embedded processing that deals with the image acquisition, compression (MJPEG or MPEG4), a webserver, and the TCP/IP stack and offer a plug "*n*" play solution. Further processing is typically restricted to, for example, user definable motion detection. All the underlying computation resources are normally hidden from the user.

The border between the two classes gets more and more fuzzy, as the machine vision-originated smart cameras get (often even GigaBit) Ethernet connection and on the other hand the IP cameras get more computing power and user accessability to the processing resources. For example, the ETRAX100LX processors of the Axis IP cameras are fully accessible and also run Linux.

1.1.2. Tracking: particle filter

Tracking is one key component of our system; thus, it is essential to choose a state-of-the-art class of tracking algorithm to ensure robust performance. Our system is based on particle filters. Particle filters have become a major way of tracking objects [6, 7]. The IEEE special issue [8] gives a good overview of the state of the art. Utilized visual cues include shape [7] and color [9–12] or a fusion of cues [13, 14]. For comparison purposes, a Kalman filter was implemented too. Although it requires very little computation time as only one hypothesis is tracked at a time, it turned out what theoretically was already apparent: the Kalman filter-based tracking was not that robust compared to the particle filter-based implementation, as it can only handle unimodal pdfs and linear systems. Also extended Kalman filters are not capable of handling multiple hypothises and are thus not that robust in cases of occlusions. It became clear at a very early stage of the project that a particle filter-based approach would succeed better, even on limited computational resources.

1.1.3. Surveillance systems

The IEEE Signal Processing issue on surveillance [15] surveys the current status of surveillance systems, for example, Foresti et al. present "Active video-based surveillance systems," Hampur et al. describe their multiscale tracking system. On CVPR05, Boult et al. gave an excellent tutorial of surveillance methods [16]. Siebel and Maybank especially deal with the problem of multicamera tracking and person handover within the ADVISOR surveillance system [17]. Trivedi et al. presented a distributed video array for situation awareness [18] that also gives a great overview about the current state of the art of surveillance systems. Yang et al. [19] describe a camera network for real-time people counting in crowds. The Sarnoff Group presented an interesting system called "video flashlight" [20] where the output of traditional cameras are used as live textures mapped onto the ground/walls of a 3D model.

However, the idea of a surveillance system consisting of a distributed network of smart cameras and live visualization embedded in a 3D model has not been covered yet.

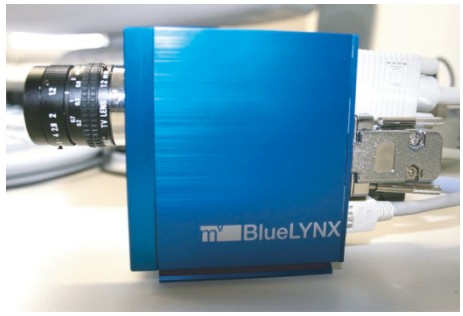


Figure 1: Our smart camera system.

2. SMART CAMERA PARTICLE FILTER TRACKING

We first describe our camera hardware before we go into the details of particle filter-based tracking in camera domain which was presented at ECV05 [21].

2.1. Smart camera hardware description

Our work is based on mvBlueLYNX 420CX smart cameras from Matrix Vision [22] as shown in Figure 1. Each smart camera consists of a sensor, an FPGA, a processor, and a networking interface. More precisely, it contains a single CCD sensor with VGA resolution (progressive scan, 12 MHz pixel clock) and an attached Bayer color mosaic. A Xilinx Spartan-IIE FPGA (XC2S400E) is used for low-level processing. A 200 MHz Motorola MPC 8241 PowerPC processor with MMU & FPU running embedded Linux is used for the main computations. It further comprises 32 MB SDRAM (64 Bit, 100 MHz), 32 MB NAND-FLASH (4 MB Linux system files, approx. 40 MB compressed user filesystem), and 4 MB NOR-FLASH (bootloader, kernel, safeboot system, system configuration parameters). The smart camera communicates via a 100 Mbps Ethernet connection, which is used both for field upgradeability and parameterization of the system and for transmission of the tracking results during runtime. For direct connection to industrial controls, 16 I/Os are available. XGA analog video output in conjunction with two serial ports are available, where monitor and mouse are connected for debugging and target initialization purposes. The form factor of the smart camera is (without lens) $(w \times h \times l)$: $50 \times 88 \times 75\ \text{mm}^3$. It consumes about 7 W power. The camera is not only intended for prototyping under laboratory conditions, it is also designed to meet the demands of harsh real world industrial environments.

2.2. Particle filter

Particle filters can handle multiple hypotheses and nonlinear systems. Following the notation of Isard and Blake [7], we define Z_t as representing all observations $\{z_1, \ldots, z_t\}$ up to time t, while X_t describes the state vector at time t with dimension k. Particle filtering is based on the Bayes rule to obtain the posterior $p(X_t \mid Z_t)$ at each time-step using all available information:

$$p(X_t \mid Z_t) = \frac{p(z_t \mid X_t)\, p(X_t \mid Z_{t-1})}{p(z_t)}, \tag{1}$$

whereas this equation is evaluated recursively as described below. The fundamental idea of particle filtering is to approximate the probability density function (pdf) over X_t by a weighted sample set $\mathbf{S}_t$. Each sample $\mathbf{s}$ consists of the state vector X and a weight π, with $\sum_{i=1}^{N} \pi^{(i)} = 1$. Thus, the ith sample at time t is denoted by $\mathbf{s}_t^{(i)} = (X_t^{(i)}, \pi_t^{(i)})$. Together they form the sample set $\mathbf{S}_t = \{\mathbf{s}_t^{(i)} \mid i = 1, \ldots, N\}$. Figure 2 shows the principal operation of a particle filter with 8 particles, whereas its steps are outlined below.

(i) Choose samples step

First, a cumulative histogram of all samples' weights is computed. Then, according to each particle's weight $\pi_{t-1}^{(i)}$, its number of successors is determined according to its relative probability in this cumulative histogram.

(ii) Prediction step

Our state has the form $X_t^{(i)} = (x, y, v_x, v_y)_t^{(i)}$. In the prediction step, the new state X_t is computed:

$$p(X_t \mid Z_{t-1}) = \int p(X_t \mid X_{t-1})\, p(X_{t-1} \mid Z_{t-1})\, dX_{t-1}. \tag{2}$$

Different motion models are possible to implement $p(X_t \mid X_{t-1})$. We use three simple motion models (whereas the specification of how many samples belong to each model can be parameterized): a random position model, a zero velocity model, and a constant velocity model ($X_t = AX_{t-1} + w_{t-1}$), each enriched with a Gaussian diffusion w_{t-1} to spread the samples and to allow for target moves differing from each motion model. A combined mode is also implemented where nonrandom samples belong either to a zero motion model or a constant velocity model. This property is handed down to each sample's successor.

(iii) Measurement step

In the measurement step, the new state X_t is weighted according to the new measurement z_t (i.e., according to the new sensor image),

$$p(X_t \mid Z_t) = p(z_t \mid X_t)\, p(X_t \mid Z_{t-1}). \tag{3}$$

The measurement step (3) complements the prediction step (2). Together they form the Bayes formulation (1).

2.3. Color histogram-based particle filter

Measurement step in context of color distributions

As already mentioned, we use a particle filter on the color histograms. This offers rotation invariant performance and

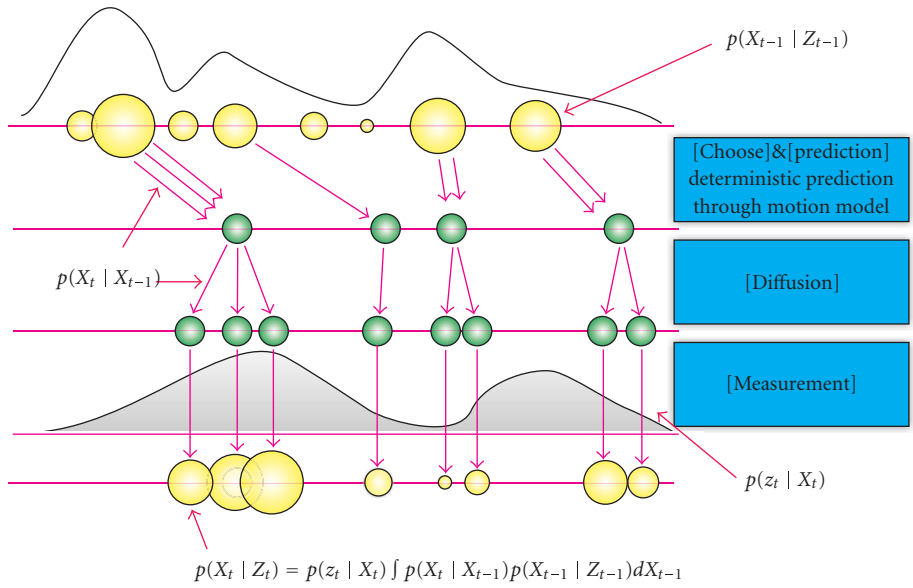


FIGURE 2: Particle filter iteration loop. The size of each sample $X_t^{(i)}$ corresponds to its weight $\pi_t^{(i)}$.

robustness against partial occlusions and nonrigidity. In contrast to using standard RGB space, we use an HSV color model: a 2D Hue-Saturation histogram (HS) in conjunction with a 1D Value histogram (V) is designed as representation space for (target) appearance. This induces the following specializations of the abstract measurement step described above.

From patch to histogram

Each sample $\mathbf{s}_t^{(i)}$ induces an image patch $P_t^{(i)}$ around its spatial position in image space, whereas the patch size (H_x, H_y) is user-definable. To further increase the robustness of the color distribution in case of occlusion or in case of present background pixels in the patch, an importance weighting dependent on the spatial distance from the patch's center is used. We employ the following weighting function:

$$k(r) = \begin{cases} 1 - r^2, & r < 1, \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

with r denoting the distance from the center. Utilizing this kernel leads to the color distribution for the image location of sample $\mathbf{s}_t^{(i)}$:

$$p_t^{(i)}[b] = f \sum_{w \in P_t^{(i)}} k\left(\frac{\| w - \tilde{X}_t^{(i)} \|}{a} \right) \delta[I(w) - b], \tag{5}$$

with bin number b, pixel position w on the patch, bandwidth $a = \sqrt{H_x^2 + H_y^2}$, and normalization f, whereas $\tilde{X}_t^{(i)}$ denotes the subset of $X_t^{(i)}$ which describes the (x, y) position in the image. The δ-function assures that each summand is assigned to the corresponding bin, determined by its image intensity I, whereas I stands for HS or V, respectively. The target representation is computed similarly, so a comparison to each sample can now be carried out in histogram space.

From histogram to new weight π

Now we compare the target histogram with each sample's histogram. For this, we use the popular Bhattacharyya similarity measure [9], both on the 2D HS and the 1D V histograms, respectively:

$$\rho[p_t^{(i)}, q_t] = \sum_{b=1}^{B} \sqrt{p_t^{(i)}[b] q_t[b]}, \tag{6}$$

with $p_t^{(i)}$ and q_t denoting the ith sample and target histograms at time t (resp., in Hue-Saturation (HS) and Value (V) space). Thus, the more similar a sample to the target appears, the larger ρ becomes. These two similarities ρ_{HS} and ρ_V are then weighted using alpha blending to get a unified similarity. The number of bins is variable, as well as the weighting factor. The experiments are performed using $10 \times 10 + 10 = 110$ bins ($H \times S + V$) and a $70 : 30$ weighting between HS and V. Then, the Bhattacharyya distance

$$d_t^{(i)} = \sqrt{1 - \rho[p_t^{(i)}, q]} \tag{7}$$

is computed. Finally, a Gaussian with user-definable variance σ is applied to receive the new observation probability for sample $\mathbf{s}_t^{(i)}$:

$$\pi_t^{(i)} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{d_t^{(i)2}}{2\sigma^2} \right). \tag{8}$$

Hence, a high Bhattacharyya similarity ρ leads to a high probability weight π and thus the sample will be favored more in the next iteration. Figure 3 illustrates how the variance

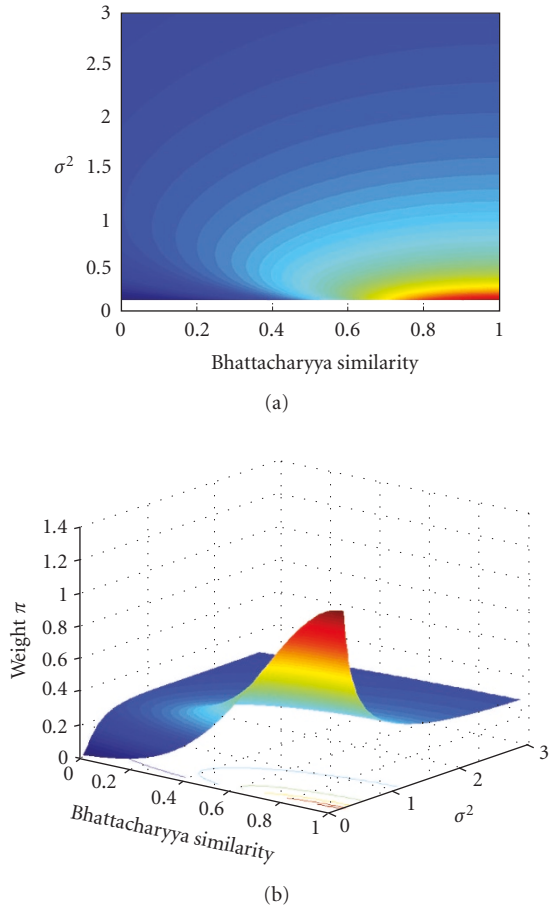


Figure 3: Mapping of Bhattacharyya similarity ρ to weight π for different variances σ^2.

σ^2 affects the mapping between ρ and the resulting weight π. A smaller variance leads to a more aggressive behavior in that samples with higher similarities ρ are pushed more extremely.

2.4. Self-adaptivity

To increase the tracking robustness, the camera automatically adapts to slow appearance (e.g., illumination) changes during runtime. This is performed by blending the appearance at the most likely position with the actual target reference appearance in histogram space:

$$q_t[b] = \alpha \times p_t^{(j)}[b] + (1 - \alpha) \times q_{t-1}[b] \tag{9}$$

for all bins $b \in \{1, \ldots, B\}$ (both in HS and V) using the mixture factor $\alpha \in [0, 1]$ and the maximum likelihood sample j, that is, $\pi_{t-1}^{(j)} = \max_{\{i=1,\ldots,N\}}\{\pi_{t-1}^{(i)}\}$. The rate of adaption α is variable and is controlled by a diagnosis unit that measures the actual tracking confidence. The idea is to *adapt wisely*, that is, the more confident the smart camera about actually tracking the target itself is, the less the risk of overlearning is and the more it to the actual appearance of the target adapts. The degree of unimodality of the resulting pdf $p(X_t \mid Z_t)$ is one possible interpretation of confidence. For example, if the target object is not present, this will result in a very uniform pdf. In this case the confidence is very low and the target representation is not altered at all to circumvent overlearning. As a simple yet efficient implementation of the confidence measure, the absolute value of the pdf's peak is utilized, which is approximated by the sample with the largest weight $\pi^{(j)}$.

2.5. Smart camera tracking architecture

Figure 4 illustrates the smart camera architecture and its output. In Figure 5 the tracking architecture of the smart camera is depicted in more details.

2.5.1. Smart camera output

The smart camera's output per iteration consists of:

(i) the pdf $p(X_t \mid Z_t)$, approximated by the sample set $\mathbf{S}_t = \{(X_t^{(i)}, \pi_t^{(i)}),\ i = 1, \ldots, N\}$; this leads to $(N * (k + 1))$ values,

(ii) the mean state $E[\mathbf{S_t}] = \sum_{i=1}^{N} \pi_t^{(i)} X_t^{(i)}$, thus one value,

(iii) the maximum likelihood state $X_t^{(j)}$ with $j \mid \pi_t^{(j)} = \max_{i=1}^{N}\{\pi_t^{(i)}\}$ in conjunction with the confidence $\pi_t^{(j)}$, resulting in two values,

(iv) optionally, a region of interest (ROI) around the sample with maximum likelihood can be transmitted too.

The whole output is transmitted via Ethernet using sockets. As only an approximation of the pdf $p(X_t \mid Z_t)$ is transmitted along with the mean and maximum likelihood state of the target, our tracking camera needs only about 15 kB/s bandwidth when using 100 samples, which is less than 0.33% of the bandwidth that the transmission of raw images for external computation would use. On the PC side, the data can be visualized on the fly or saved on hard disk for offline evaluation.

2.6. Particle filter tracking results

Before we illustrate some results, several benefits of this smart camera approach are described.

2.6.1. Benefits

(i) Low-bandwidth requirements

The raw images are processed directly on the camera. Hence, only the approximated pdf of the target's state has to be transmitted from the camera using relatively few parameters. This allows using standard networks (e.g., Ethernet) with virtually unlimited range. In our work, all the output amounts to $(N * (k + 1) + 3)$ values per frame. For example, using $N = 100$ and constant velocity motion model ($k = 4$) leads to 503 values per frame. This is quite few data compared to transmitting all pixels of the raw image. For example (even undemosaiced) VGA resolution needs about 307 k pixel values per frame. Even at (moderate) 15 fps this already leads to 37 Mbps transmission rate, which is about 1/3 of the standard 100 Mbps bandwidth. Of course, modern IP

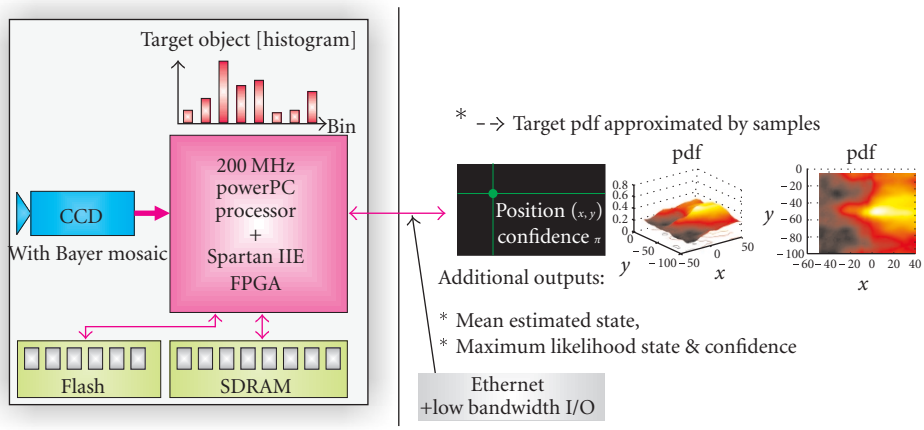


Figure 4: Smart camera architecture.

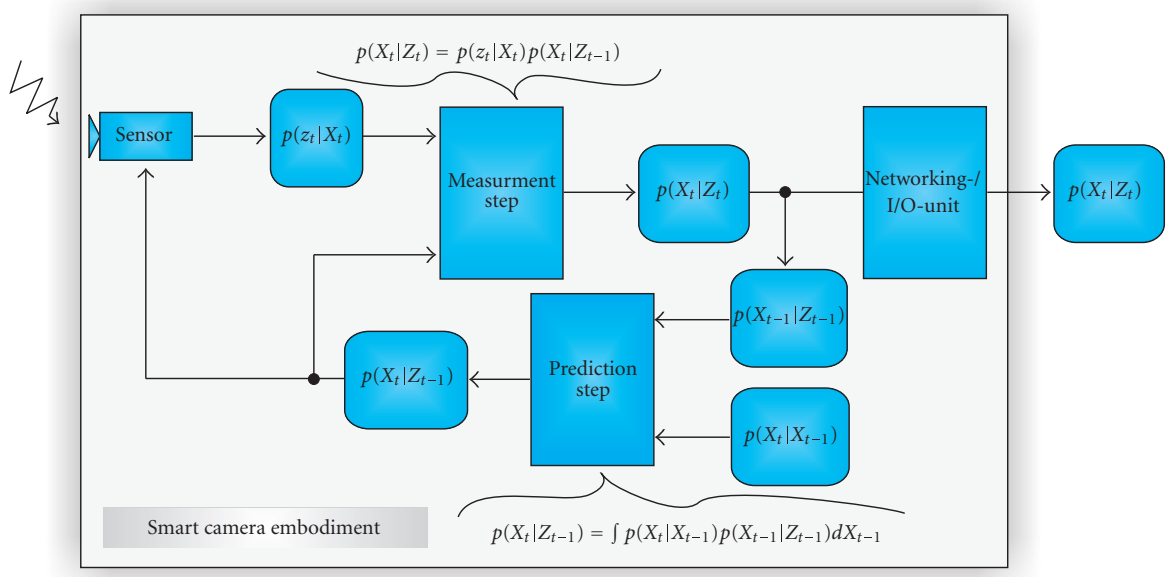


Figure 5: Smart camera tracking architecture.

cameras offer, for example, MJPEG or MPEG4/H.264 compression which drastically reduces the bandwidth. However, if the compression is not almost lossless, introduced artefacts could disturb the further video processing. The smart camera approach instead performs the processing embedded in the camera on the raw, unaltered images. Compression also requires additional computational resources which is not required with the smart camera approach.

(ii) No additional computing outside the camera has to be performed

No networking enabled external processing unit (a PC or a networking capable machine control in factory automation) has to deal with low-level processing any more which algorithmically belongs to a camera. Instead it can concentrate on higher-level algorithms using all smart cameras' outputs as basis. Such a unit could also be used to passively supervise all outputs (e.g., in case of a PDA with WiFi in a surveillance application). Additionally, it becomes possible to connect the output of such a smart camera directly to a machine control unit (that does not offer dedicated computing resources for external devices), for example, to a robot control unit for visual servoing. For this, the mean or the maximum likelihood state together with a measure for actual tracking confidence can be utilized directly for real-time machine control.

(iii) Higher resolution and framerate

As the raw video stream does not need to comply with the camera's output bandwidth any more, sensors with higher

spatial or temporal resolutions can be used. Due to the very close spatial proximity between sensor and processing means, higher bandwidth can be achieved more easily. In contrast, all scenarios with a conventional vision system (camera + PC) have major drawbacks. First, transmitting the raw video stream in full spatial resolution at full frame rate to the external PC can easily exceed today's networking bandwidths. This applies all the more when multiple cameras come into play. Connections with higher bandwidths (e.g., CameraLink) on the other hand are too distance-limited (besides the fact that they are typically host-centralized). Second, if only regions-of-interest (ROIs) around samples induced by the particle filter were transmitted, the transmission between camera and PC would become part of the particle filter's feedback loop. Indeterministic networking effects provoke that the particle filter's prediction of samples' states (i.e., ROIs) is not synchronous with the real world any more and thus measurements are done at wrong positions.

(iv) Multicamera systems

As a consequence of the above benefits, this approach offers optimal scaling for multicamera systems to work together in a decentralized way which enables large-scale camera networks.

(v) Small, self-contained unit

The smart camera approach offers a self-contained vision solution with a small form factor. This increases the reliability and enables the installation at size-limited places and on robot hands.

(vi) Adaptive particle filter's benefits

A Kalman filter implementation on a smart camera would also offer these benefits. However, there are various drawbacks as it can only handle unimodal pdfs and linear models. As the particle filter approximates the—potentially arbitrarily shaped—pdf $p(X_t \mid Z_t)$ somewhat efficiently by samples, the bandwidth overhead is still moderate whereas the tracking robustness gain is immense. By adapting to slow appearance changes of the target with respect to the tracker's confidence, the robustness is further increased.

2.6.2. Experimental results

We will outline some results which are just an assortment of what is also available for download from the project's website [23] in higher quality. For our first experiment, we initialize the camera with a cube object. It is trained by presenting it in front of the camera and saving the according color distribution as target reference. Our smart camera is capable of robustly following the target over time at a framerate of over 15 fps. For increased computational efficiency, the tracking directly runs on the raw and thus still Bayer color-filtered pixels exist. Instead of first doing expensive Bayer demosaicing and finally only using the histogram which still contains no spatial information, we interpret each four-pixel Bayer neighborhood as one pixel representing RGB intensity (whereas the two-green values are averaged), leading to QVGA resolution as tracking input. In the first experiment, a cube is tracked which is moved first vertically, then horizontally, and afterwards in a circular way. The final pdf $p(X_t \mid Z_t)$ at time t from the smart camera is illustrated in Figure 6, projected in x and y directions. Figure 7 illustrates several points in time in more detail. Concentrating on the circular motion part of this cube sequence, a screenshot of the samples' actual positions in conjunction with their weights is given. Note that we do not take advantage of the fact that the camera is mounted statically; that is, no background segmentation is performed as a preprocessing step.

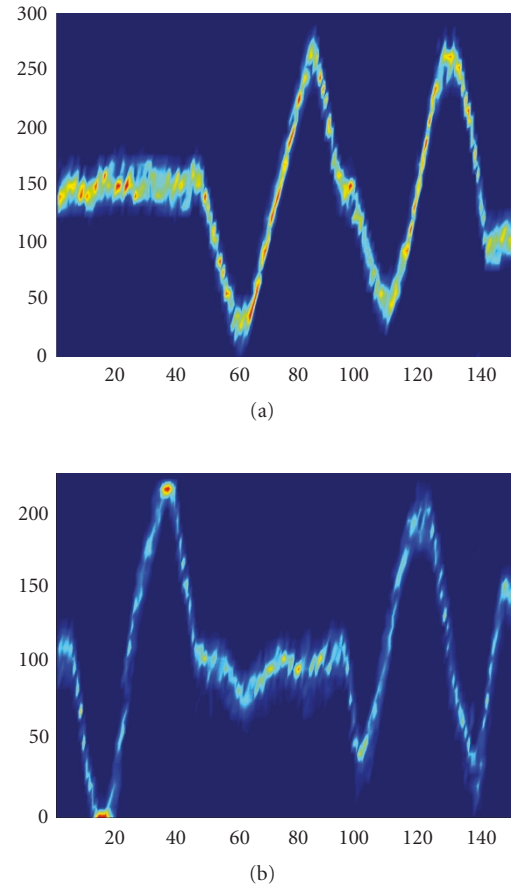
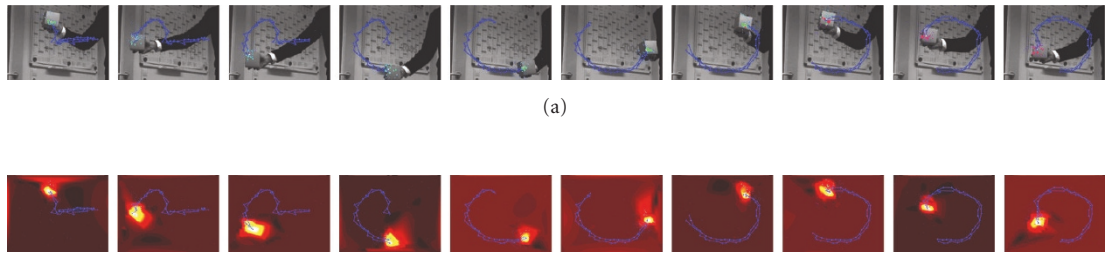


Figure 6: Experiment no. 1: pdf $p(X_t \mid Z_t)$ over iteration time t. (a) x-component, (b) y-component.

In the second experiment, we evaluate the performance of our smart camera in the context of surveillance. The smart camera is trained with a person's face as target. It shows that the face can be tracked successfully in real time too. Figure 8 shows some results during the run.

3. 3D SURVEILLANCE SYSTEM

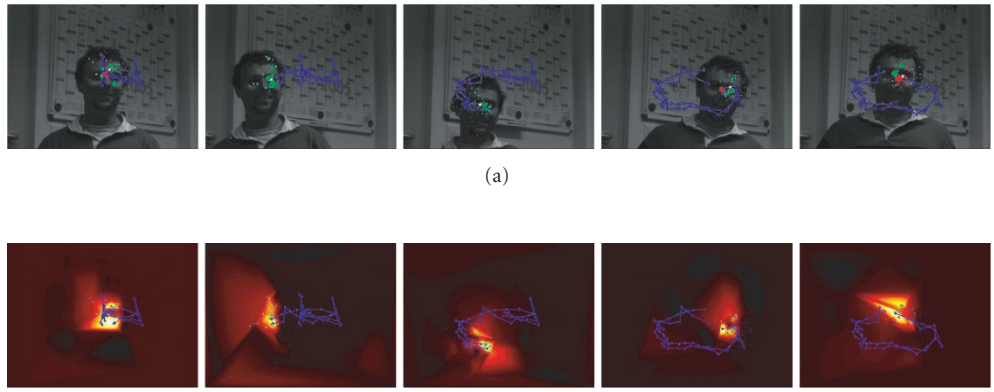
To enable tracking in world model domain, decoupled from cameras (instead of in the camera image domain), we now



(a)

(b)

Figure 7: Circular motion sequence of experiment no. 1. Image (a) and approximated pdf (b). Samples are shown in green; the mean state is denoted as yellow star.



(a)

(b)

Figure 8: Experiment no. 2: face tracking sequence. Image (a) and approximated pdf (b) at iteration no. 18, 35, 49, 58, 79.

extend the system described above as follows. It is based on our ECV06 work [24, 25].

3.1. Architecture overview

The top-level architecture of our distributed surveillance and visualization system is given in Figure 9. It consists of multiple networking-enabled camera nodes, a server node and a 3D visualization node. In the following, all components are described on top level, before each of them is detailed in the following sections.

Camera nodes

Besides the preferred realization as smart camera, our system also allows for using standard cameras in combination with a PC to form a camera node for easier migration from deprecated installations.

Server node

The server node acts as server for all the camera nodes and concurrently as client for the visualization node. It manages configuration and initialization of all camera nodes, collects the resulting tracking data, and takes care of person handover.

Visualization node

The visualization node acts as server, receiving position, size, and texture of each object currently tracked by any camera from the server node. Two kinds of visualization nodes are implemented. The first is based on the XRT point cloud-rendering system developed at our institute. Here, each object is embedded as a sprite in a rendered 3D point cloud of the environment. The other option is to use Google Earth as visualization node. Both the visualization node and the server node can run together on a single PC.

3.2. Smart camera node in detail

The smart camera tracking architecture as one key component of our system is illustrated in Figure 10 and comprises the following components: a background modeling and auto init unit, multiple instances of a particle filter-based tracking unit, $2D \rightarrow 3D$ conversion units, and a network unit.

3.2.1. Background modeling and autoinit

In contrast to Section 2, we take advantage of the fact that each camera is mounted statically. This enables the use of a background model for segmentation of moving objects. The background modeling unit has the goal to model the actual

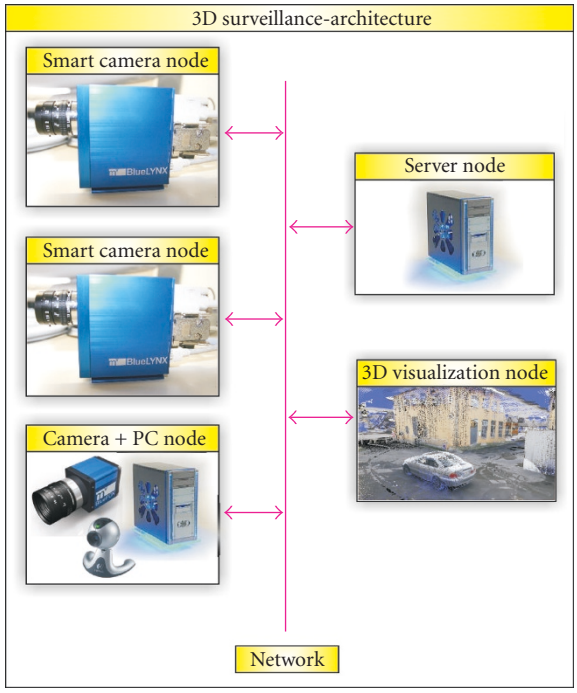


Figure 9: 3D surveillance system architecture.

background in real time, that is, foreground objects can be extracted very *robustly*. Additionally, it is important, that the background model *adapts* to slow appearance (e.g., illumination) changes of the scene's background. Elgammal et al. [26] give a nice overview of the requirements and possible cues to use within such a background modeling unit in the context of surveillance. Due to the embedded nature of our system, the unit has to be very computationally efficient to meet the real-time demands. State-of-the-art background modeling algorithms are often based on layer extraction, (see, e.g., Torr et al. [27]) and mainly target segmentation accuracy. Often a graph cut approach is applied to (layer) segmentation, (see, e.g., Xiao and Shah [28]) to obtain high-quality results.

However, it became apparent, that these algorithms are not efficient enough for our system to run concurrently together with multiple instances of the particle filter unit. Hence we designed a robust, yet efficient, background algorithm that meets the demands, yet works with the limited computational resources available on our embedded target. It is capable of running at 20 fps at a resolution of $320 * 240$ pixels on the mvBlueLYNX 420CX that we use. The background modeling unit works on a per-pixel basis. The basic idea is that a model for the background b_t and an estimator for the noise process η_t at the current time t is extracted from a set of n recent images $i_t, i_{t-1}, \ldots, i_{t-n}$. If the difference between the background model and the current image, $|b_t - i_t|$, exceeds a value calculated from the noisiness of the pixel, $f_1(\eta_t) = c_1 * \eta_t + c_2$, where c_1 and c_2 are constants, the pixel is marked as moving. This approach, however, would require storing n complete images. If n is set too low ($n < 500$), a car stopping at a traffic light, for example, would become part of the background model and leave a ghost image of the road as a detected object after moving on because the background model would have already considered the car as part of the scenery itself, instead of an object. Since the amount of memory necessary to store $n = 500$ images consisting of $320 * 240$ RGB pixels is $500 * 320 * 240 * 3 = 115200000$ bytes (over 100 MB), it is somewhat impractical.

Instead we only buffer $n = 20$ images but introduce a confidence counter j_t that is increased if the difference between the oldest and newest images $|i_t - i_{t-n}|$ is smaller than $f_2(\eta_t) = c_1 * \eta_t + c_3$, where c_1 and c_3 are constants, or reset otherwise. If the counter reaches the threshold τ, the background model is updated. The noisiness estimation η_t is also modeled by a counter that is increased by a certain value (default: 5) if the difference in RGB color space of the actual image to the oldest image in the buffer exceeds the current noisiness estimation. The functions f_1 and f_2 are defined as linear functions mainly due to computational cost considerations and to limit the number of constants (c_1, c_2, c_3) which need to be determined experimentally. Other constants, such as τ which represents a number of frames and thus directly relates to time, are simply chosen by defining the longest amount of time an object is allowed to remain stationary before it becomes part of the background.

The entire process is illustrated in Figure 11. The current image i_t (a) is compared to the oldest image in the buffer i_{t-n} (b) and if the resulting difference $|i_t - i_{t-n}|$ (c) is higher than the threshold $f_2(\eta_t) = c_1 * \eta_t + c_3$ calculated from the noisiness η_t (d), the confidence counter j_t (e) is reset to zero, otherwise it is increased. Once the counter reaches a certain level, it triggers the updating of the background model (f) at this pixel. Additionally, it is reset back to zero for speed purposes (to circumvent adaption and thus additional memory operations at every frame). For illustration purposes, the time it takes to update the background model is set to 50 frames (instead of 500 or higher in a normal environment) in Figure 11 (see first rising edge in (f)). The background is updated every time the confidence counter (e) reaches 50. The fluctuations of (a) up until $t = 540$ are not long enough to update the background model and are hence marked as moving pixels in (g). This is correct behavior as the fluctuations simulate objects moving past. At $t = 590$ the difference (c) kept low for 50 frames sustained, so the background model is updated (in (f)) and the pixel is no longer marked as moving (g). This simulates an object that needs to be incorporated into the background (like a parked car). The fluctuations towards the end are then classified as moving pixels (e.g., people walking in front of the car).

Segmentation

Single pixels are first eliminated by a 4-neighborhood erosion. From the resulting mask of movements, areas are constructed via a region growing algorithm: the mask is scanned for the first pixel marked as moving. An area is constructed around it and its borders checked. If a moving pixel is found on it, the area expands in that direction. This is done iteratively until no border pixel is marked. To avoid breaking up of

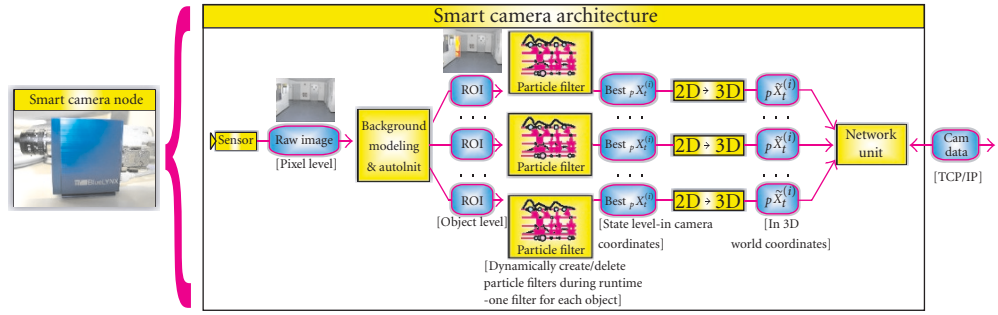


Figure 10: Smart camera node's architecture.

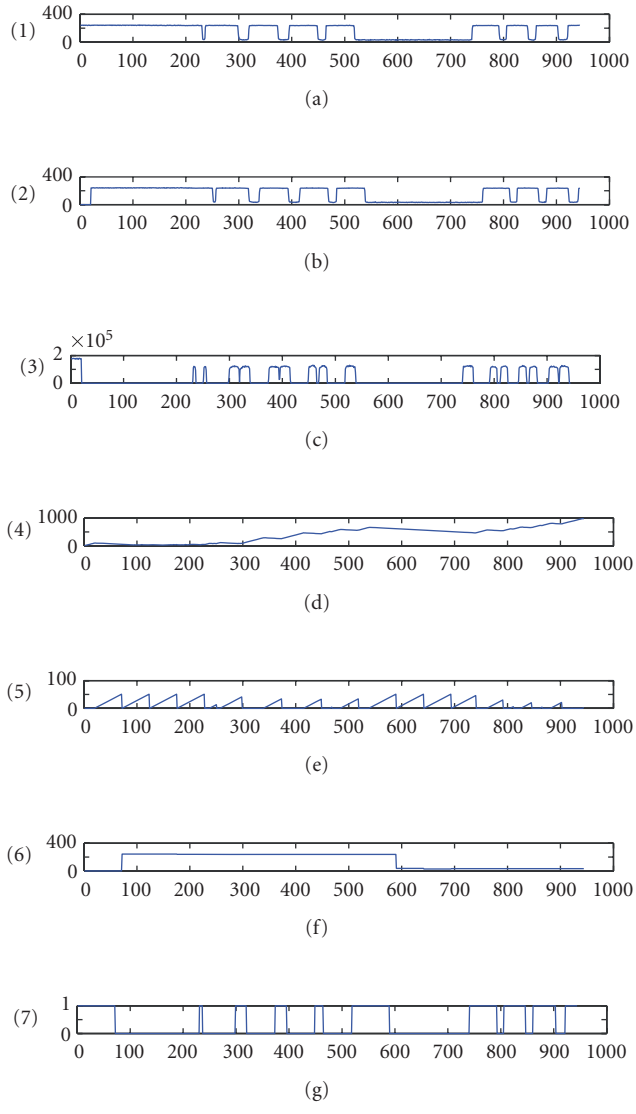


Figure 11: High-speed background modeling unit in action. Per pixel: (a) raw pixel signal from camera sensor. (b) 10 frames old raw signal. (c) Difference between (a) and (b). (d) Noise process. (e) Confidence counter: increased if pixel is consistent with background within a certain tolerance, reset otherwise. (f) Background model. (g) Trigger event if motion is detected.

objects into smaller areas, areas near each other are merged. This is done by expanding the borders a certain amount of pixels beyond the point where no pixels were found moving any more. Once an area is completed, the pixels it contains are marked "nonmoving" and the algorithm starts searching for the next potential area. This unit thus handles the transformation from raw pixel level to object level.

Auto initialization and destruction

If the region is not already tracked by an existing particle filter, a new filter is instantiated with the current appearance as target and assigned to this region. An existing particle filter that has not found a region of interest near enough over a certain amount of time is deleted.

This enables the tracking of multiple objects, where each object is represented by a separate color-based particle filter. Two particle filters that are assigned the same region of interest (e.g., two people that walk close to each other after meeting) are detected in a last step and one of them is eliminated if the object does not split up again after a certain amount of time.

3.2.2. Multiobject tracking—color-based particle filters

Unlike in Section 2, a particle filter engine is instantiated for each person/object p. Due to the availability of the background model several changes were made.

(i) The confidence for adaption comes from the background model as opposed to the pdf's unimodality.
(ii) The state $X_t^{(i)}$ also comprises the object's size.
(iii) The likeliness between sample and ROI influences the measurement process and calculation of $\pi_t^{(i)}$.

3.2.3. 2D → 3D conversion unit

3D tracking is implemented by converting the 2D tracking results in image domain of the camera to a 3D world coordinate system with respect to the (potentially georeferenced) 3D model, which also enables global, intercamera handling and handover of objects.

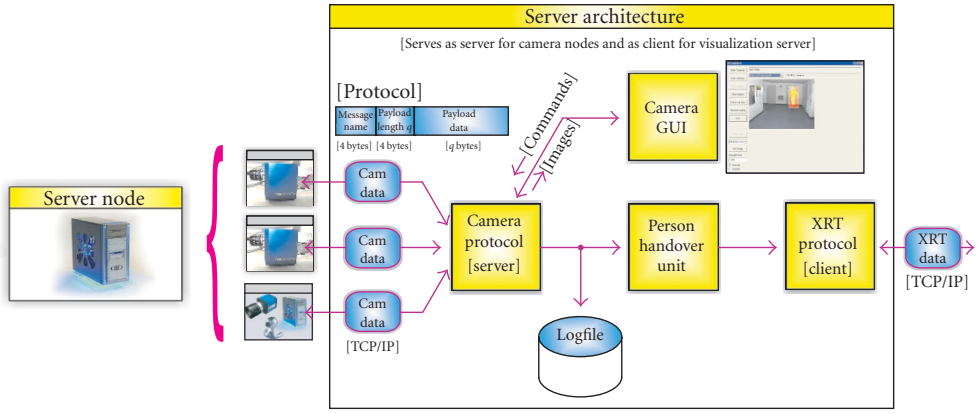


Figure 12: Server architecture.

Since both external and internal camera parameters are known (manually calibrated by overlaying a virtual rendered image with a live camera image), we can convert 2D pixel coordinates into world coordinate view rays. The view rays of the lower left and right corner of the object are intersected with the fixed ground plane. The distance between them determines the width and the mean determines the position of the object. The height (e.g., of a person) is calculated by intersecting the view ray from the top center pixel with the plane perpendicular to the ground plane that goes through the two intersection points from the first step. If the object's region of interest is above the horizon, the detected position lies behind the camera and it will be ignored. The extracted data is then sent to the server along with the texture of the object.

3.2.4. Parameter selection

The goal is to achieve the best tracking performance possible in terms of robustness, precision, and framerate within the given computational resources of the embedded target. As the surveillance system is widely parameterizable and many options affect computational demands, an optimal combination has to be set up. This optimization problem can be subdivided in a background unit and a tracking unit parameter optimization problem. There are basically three levels of abstraction that affect computational time (bottom up): pixel level, sample level, and object level operations.

Some parameters, for example, noise, are adapted automatically during runtime. Other parameters that do not affect the computational resources have been set only under computer vision aspects taking the environment (indoor versus outdoor) into account. All background unit parameters and most tracking parameters belong to this class. Most of these parameters have been selected using visual feedback from debug images.

Within the tracking unit, especially the number of particles N is crucial for runtime. As the measurement step is the most expensive part of the particle filter, its parameters have to be set with special care. These consist of the number of histogram bins and the per particle subsampling size which determines the number of pixels over which a histogram is built. In practice, we set these parameters first and finally set N which linearly affects the tracking units runtime to get the desired sustained tracking framerate.

3.3. Server node in detail

The server node is illustrated in Figure 12. It consists of a camera protocol server, a camera GUI, a person handover unit and a client for the visualization node.

3.3.1. Camera protocol server

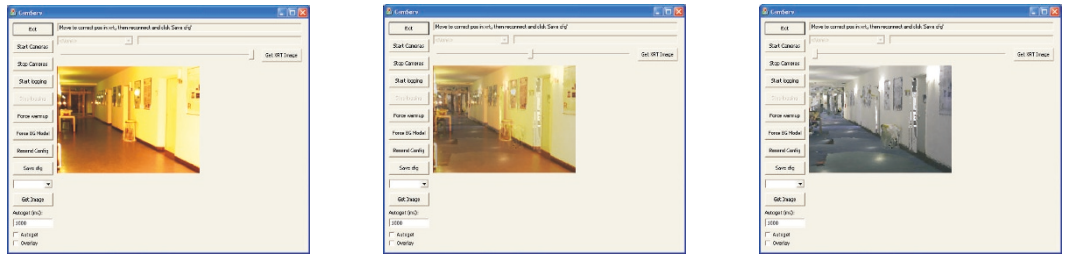
The camera server implements a binary protocol for communication with each camera node based on TCP/IP. It serves as sink for all camera nodes' tracking result streams which consist of the actual tracking position and appearance (texture) of every target per camera node in world coordinates. This information is forwarded both to the person handover unit and to a log file that allows for debugging and playback of recorded data. Additionally, raw camera images can be acquired from any camera node for the camera GUI.

3.3.2. Camera GUI

The camera GUI visualizes all the results of any camera node. The segmented and tracked objects are overlayed over the raw sensor image. The update rate can be manually adjusted to save bandwidth. Additionally, the camera GUI supports easy calibration relative to the model by blending the rendered image of a virtual camera over the current live image as basis for optimal calibration, as illustrated in Figure 13.

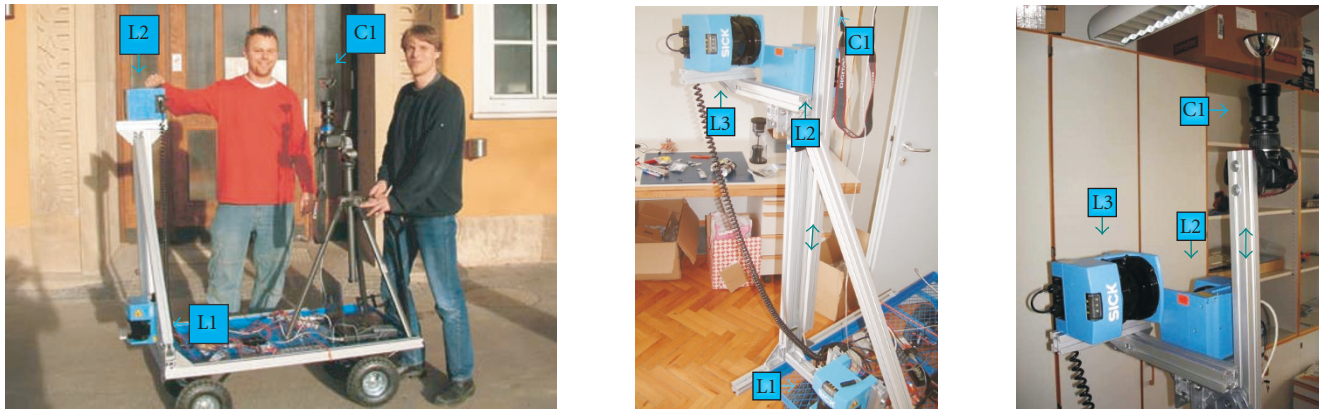
3.3.3. Person handover unit

To achieve a seamless intercamera tracking decoupled from each respective sensor node, the person handover unit merges objects tracked by different camera nodes if they are in spatial proximity. Obviously, this solution is far from perfect, but we are currently working to improve the handover process by integrating the object's appearance, that



(a) (b) (c)

Figure 13: Camera GUI. Different blending levels are shown: (a) real raw sensor image and (c) rendered scene from same viewpoint.



(a) (b) (c)

Figure 14: Two setups of our mobile platform. (a) Two laser scanners, L1 and L2, and one omnidirectional camera C1. (b), (c) Three laser scanners, L1, L2, and L3, and omnidirectional camera closely mounted together.

is, comparing color distributions over multiple spatial areas of the target using the Bhattacharyya distance on color-calibrated cameras. Additionally, movements over time will be integrated to further identify correct handovers. At the moment, the unit works as follows: after a new object has been detected by a camera node, its tracking position is compared to all other objects that are already being tracked. If an object at a similar position is found, it is considered the same object and statically linked to it using its global id.

3.4. Visualization node in detail

A practical 3D surveillance system also comprises an easy way of acquiring 3D models of the respective environment. Hence, we briefly present our 3D model acquisition system that provides the content for the visualization node which is described afterwards.

3.4.1. 3D model acquisition for 3D visualization

The basis for 3D model acquisition is our mobile platform which we call the Wägele[1] [29]. It allows for an easy acquisition of indoor and outdoor scenes. 3D models are acquired just by moving the platform through the scene to be captured. Thereby, geometry is acquired continuously and color images are taken in regular intervals. Our platform (see Figure 14) comprises an 8 megapixel omnidirectional camera (C1 in Figure 14) in conjunction with three laser scanners (L1–L3 in Figure 14) and an attitude heading sensor (A1 in Figure 14). Two flows are implemented to yield 3D models: a computer vision flow based on graph cut stereo and a laser scanner based-modeling flow.

After a recording session, the collected data is assembled to create a consistent 3D model in an automated offline processing step. First a 2D map of the scene is built and all scans of the localization scanner (and the attitude heading sensor) are matched to this map. This is accomplished by probabilistic scan matching using a generative model developed by Biber and Straßer [30]. After this step the position and orientation of the Wägele is known for each time step. This data is then fed into the graph cut stereo pipeline and the laser scanner pipeline. The stereo pipeline computes dense depth maps using pairs of panoramic images taken from different positions. The laser flow projects the data from laser scanners L2 and L3 into space using the results of the localization step. L2 and L3 together provide a full 360° vertical slice of the environment. The camera C1, then, yields the texture for the 3D models. More details can be found in [29, 31].

[1] *Wägele*—Swabian for a little cart.

Figure 15: Outdoor setup. (a), (h) Renderings of the acquired model in XRT visualization system. (b) Dewarped example of an omnidirectional image of the model acquisition platform. (c), (f) Live view of camera nodes with overlayed targets currently tracking. (d), (e) Rendering of resulting person of (c) in XRT visualization system from two viewpoints. (i)–(k) More live renderings in XRT.

3.4.2. 3D visualization framework—XRT

The visualization node gets its data from the server node and renders the information (all objects currently tracked) embedded in the 3D model. The first option is based on the experimental rendering toolkit (XRT) developed by Michael Wand et al. at our institute which is a modular framework for real-time point-based rendering. The viewpoint can be chosen arbitrarily. Also a fly-by mode is available that moves the viewpoint with a tracked person/object. Objects are displayed as sprites using live textures. Resulting renderings are shown in Section 3.5.

3.4.3. Google Earth as visualization node

As noted earlier, our system also allows Google Earth [3] to be used as visualization node. As shown in the results, each object is represented with a live texture embedded in the Google Earth model. Of course, the viewpoint within Google Earth can be chosen arbitrarily during runtime, independent of the objects being tracked (and independent of the camera nodes).

3.5. 3D surveillance results and applications

Two setups have been evaluated over several days, an outdoor setup and an indoor setup in an office environment. More details and videos can be found on the project's website [23]. First, a 3D model of each environment has been acquired. Afterwards, the camera network has been set up and calibrated relative to the model. Figures 15 and 16 show some results of the outdoor setup. Even under strong gusts where the trees were heavily moving, our per pixel noise process estimator enabled robust tracking by spatially adapting to the

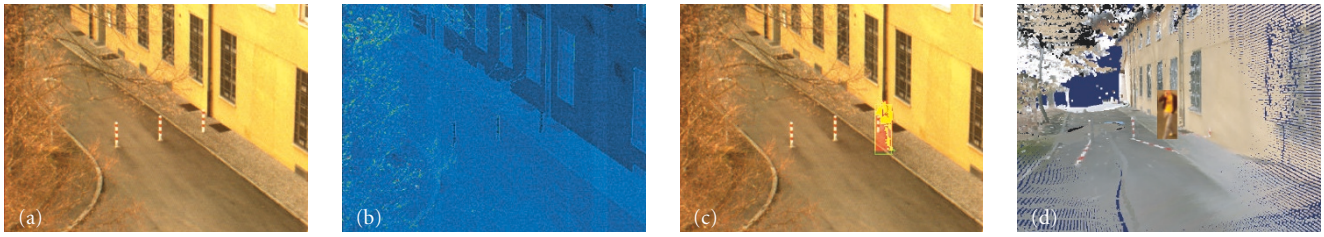


Figure 16: Outdoor experiment. (a) Background model. (b) Estimated noise. (c) Live smart camera view with overlayed tracking information. (d) Live XRT rendering of tracked object embedded in 3D model.

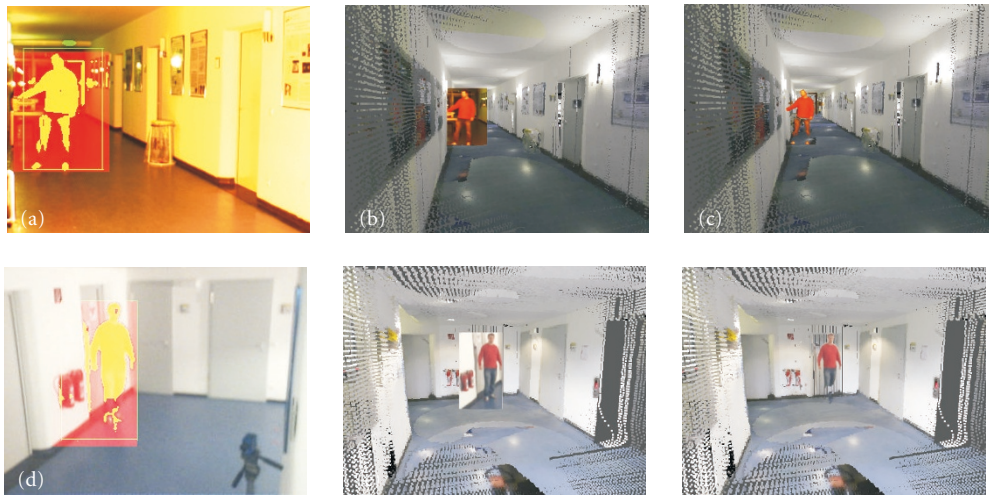


Figure 17: Indoor setup. Renderings of the XRT visualization node. (a), (d) Output of the server node (camera GUI): raw image of the camera node, overlayed with the target object on which a particle filter is running. (a), (d) Live camera views overlayed by tracking results. (b), (e) Rendering of embedded live texture in XRT visualization system. (c), (f) Same as center, but with alpha map enabled: only segmented areas are overlayed for increased realism.

respective background movements. Some indoor results are illustrated in Figure 17. To circumvent strong reflections on the floor in the indoor setup, halogen lamps are used with similar directions as the camera viewpoints.

Results of the Google Earth visualization are shown in the context of the self-localization application on our campus. Before, experimental results are described.

3.5.1. *Long term experiment*

We have set up the surveillance system with 5 camera nodes for long-term indoor operation. It runs 24/7 for 4 weeks now and shows quite promising performance. Both the 3D visualization within XRT and the Google Earth visualization were used. Figure 19 illustrates the number of persons tracked concurrently within the camera network. In Figure 20 the distribution of number of events per hour within a day is shown, accumulated over an entire week. The small peak in the early morning hours is due to a night-watchman. The peak between 11 AM and 12 PM clearly shows the regular lunchtime movements on the hallway. Many students leave in the early afternoon, however as usual in the academic world, days tend to grow longer and longer, as seen in the figure. As we do not have an automated way of detecting false tracking results, no qualitative results showing false object detection rate are available yet.

3.5.2. *Application—self-localization*

An interesting application on top of the described 3D surveillance system is the ability to perform multiperson self-localization within such a distributed network of cameras. Especially in indoor environments, where no localization mechanisms like GPS are available; our approach delivers highly accurate results without the need for special localization devices for the user. The scenario looks like this: a user with a portable computer walks into an unknown building or airport, connects to the local WiFi network, and downloads the XRT viewer or uses Google Earth. The data is then streamed to him and he can access the real-time 3D surveillance data feed, where he is embedded as an object. Choosing the follow option in XRT, the viewer automatically follows the user through the virtual scene. The user can then navigate virtually through the environment starting from his actual position.

Figure 18 illustrates a self-localization setup on our campus. The person to be localized (d) is tracked and visualized

Figure 18: (a) Live view of one camera node where the detected and currently tracked objects are overlayed. (b) Estimated noise of this camera. Note the extremely high values where trees are moving due to heavy gusts. (c) Overview of the campus in Google Earth, as the person to be localized sees it. (d) Image of the person being localized with the WiFi-enabled laptop where the visualization node runs. (e) Same as (c), (f), with XRT used as visualization instrument in conjunction with a georeferenced 3D model acquired by our Wägele platform, embedded in a low-resolution altitude model of the scene. Note that the 5 objects currently tracked are embedded as live billboard textures. (f) Close up Google Earth view of (c). Four people and a truck are tracked concurrently. The person to be localized is the one in the bottom center carrying the notebook.

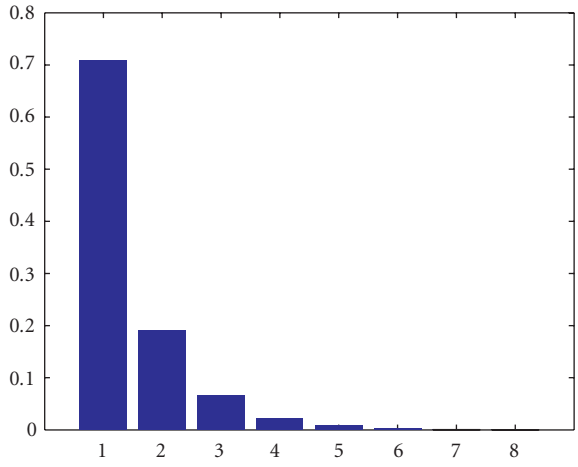


Figure 19: Distribution of number of persons concurrently tracked by the camera network over one week of operation.

in Google Earth (c), (f) and simultaneously in our XRT point-based renderer (e).

4. CONCLUSION AND FUTURE IDEAS

Our approach of such a distributed network of smart cameras offers various benefits. In contrast to a host centralized approach, the possible number of cameras can easily exceed hundreds. Neither the computation power of a host nor the physical cable length (e.g., like with CameraLink) is a limiting factor. As the whole tracking is embedded inside each smart camera node, only very limited bandwidth is necessary, which makes the use of Ethernet possible. Additionally, the possibility to combine standard cameras and PCs to form local camera nodes extends the use of PC-based surveillance over larger areas where no smart cameras are available yet. Our system is capable of tracking multiple persons in real time. The inter-camera tracking results are embedded as live textures in a consistent and georeferenced 3D world model, for example, acquired by our mobile platform or within Google Earth. This enables the intuitive 3D visualization of tracking results decoupled from sensor views and tracked persons.

One key decision in our system design was to use a distributed network of smart cameras as this provides various benefits compared to a more traditional, centralized approach as described in Section 2.6.1. Due to real-time requirements and affordability of large installations, we decided to design our algorithms not only on quality but also on computational efficiency and parameterizability. Hence, we chose to implement a rather simplistic background modeling unit and a fast particle filtering measurement step.

The detection of a person's feet turned out to not always be reliable (which affects the estimated distance from the camera), also reflections on the floor affected correct localization of objects. This effect could be attenuated by using polarization filters. Also, the system fails, if multiple objects are constantly overlapping each other (e.g., crowds).

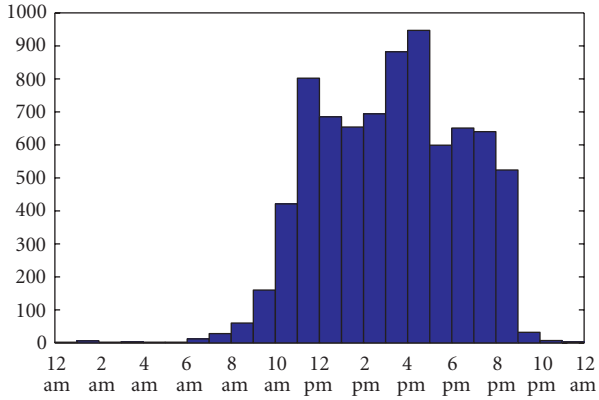


Figure 20: Events over one week, sorted by hour.

Future research includes person identification using RFID tags, long-term experiments, and the acquisition of enhanced 3D models. Additionally, integrating the acquired 3D Wägele models within Google Earth will also allow for seamless tracking in large, textured 3D indoor and outdoor environments. However, the acquired models are still far too complex.

Self-localization future ideas

A buddy list could be maintained that goes beyond of what typical chat programs offer today: the worldwide, georeferenced localization visualization of all your buddies instead of just a binary online/away classification. Additionally, an idea is that available navigation software could be used on top of this to allow for indoor navigation, for example, to route to a certain office or gate. In combination with the buddy list idea, this enables novel services like flexible meetings at unknown places, for example, within the airport a user can be navigated to his buddy whereas the buddy's location is updated during runtime. Also, security personnel can use the same service to catch a suspicious person moving inside the airport.

ACKNOWLEDGMENTS

We would like to thank Matrix Vision for their generous support and successful cooperation, Peter Biber for providing Wägele data and many fruitful discussions, Sven Lanwer for his work within smart camera-based tracking, and Michael Wand for providing the XRT rendering framework.

REFERENCES

[1] J. Mullins, "Rings of steel ii, New York city gets set to replicate London's high-security zone," to appear in *IEEE Spectrum*.

[2] "What happens in vegas stays on tape," http://www.csoonline.com/read/090105/hiddencamera_vegas_3834.html.

[3] "Google earth," http://earth.google.com/.

[4] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, 2006.

[5] W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, no. 9, pp. 48–53, 2002.

[6] A. Doucet, N. D. Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*, Springer, New York, NY, USA, 2001.

[7] M. Isard and A. Blake, "Condensation—conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[8] S. Haykin and N. de Freitas, "Special issue on sequential state estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 399–400, 2004.

[9] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, 2003.

[10] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe, "A boosted particle filter: multitarget detection and tracking," in *Proceedings of 8th European Conference on Computer Vision (ECCV '04)*, Prague, Czech Republic, May 2004.

[11] K. Nummiaro, E. Koller-Meier, and L. V. Gool, "A color based particle filter," in *Proceedings of the 1st International Workshop on Generative-Model-Based Vision (GMBV '02)*, Copenhagen, Denmark, June 2002.

[12] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," in *Proceedings of 7th European Conference on Computer Vision (ECCV '02)*, pp. 661–675, Copenhaguen, Denmark, June 2002.

[13] P. Pérez, J. Vermaak, and A. Blake, "Data fusion for visual tracking with particles," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 495–513, 2004, issue on State Estimation.

[14] M. Spengler and B. Schiele, "Towards robust multi-cue integration for visual tracking," in *Proceedings of the 2nd International Workshop on Computer Vision Systems*, vol. 2095 of *Lecture Notes in Computer Science*, pp. 93–106, Vancouver, BC, Canada, July 2001.

[15] "Surveillance works: look who's watching," *IEEE Signal Processing Magazine*, vol. 22, no. 3, 2005.

[16] T. Boult, A. Lakshmikumar, and X. Gao, "Surveillance methods," in *Proceedings of IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR '05)*, San Diego, Calif, USA, June 2005.

[17] N. Siebel and S. Maybank, "The advisor visual surveillance system," in *Proceedings of the ECCV Workshop on Applications of Computer Vision (ACV '04)*, Prague, Italy, May 2004.

[18] M. M. Trivedi, T. L. Gandhi, and K. S. Huang, "Distributed interactive video arrays for event capture and enhanced situational awareness," *IEEE Intelligent Systems*, vol. 20, no. 5, pp. 58–65, 2005, special issue on Homeland Security, 2005.

[19] D. B. Yang, H. H. González-Baños, and L. J. Guibas, "Counting people in crowds with a real-time network of simple image sensors," in *Proceedings of the 9th IEEE International Conference on Computer Vision (ICCV '03)*, vol. 1, pp. 122–129, Nice, France, October 2003.

[20] H. S. Sawhney, A. Arpa, R. Kumar, et al., "Video flashlights—real time rendering of multiple videos for immersive model visualization," in *Proceedings of the 13th Eurographics Workshop on Rendering (EGRW '02)*, pp. 157–168, Pisa, Italy, June 2002.

[21] S. Fleck and W. Straßer, "Adaptive probabilistic tracking embedded in a smart camera," in *Proceedings of the IEEE CVPR Embedded Computer Vision Workshop (ECV '05)*, vol. 3, p. 134, San Diego, Calif, USA, June 2005.

[22] "Matrix vision," http://www.matrix-vision.com/.

[23] "Project's website," http://www.gris.uni-tuebingen.de/~sfleck/smartsurv3d/.

[24] S. Fleck, F. Busch, P. Biber, and W. Straßer, "3d surveillance—a distributed network of smart cameras for real-time tracking and its visualization in 3d," in *Proceedings of IEEE CVPR Embedded Computer Vision Workshop (ECV '06)*, 2006.

[25] S. Fleck, F. Busch, P. Biber, and W. Straßer, "3d surveillance—a distributed network of smart cameras for real-time tracking and its visualization in 3d," in *Proceedings of Conference on Computer Vision and Pattern Recognition Workshop (CVPRW '06)*, p. 118, June 2006.

[26] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1162, 2002.

[27] P. H. S. Torr, R. Szeliski, and P. Anandan, "An integrated Bayesian approach to layer extraction from image sequences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 3, pp. 297–303, 2001.

[28] J. Xiao and M. Shah, "Motion layer extraction in the presence of occlusion using graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1644–1659, 2005.

[29] P. Biber, S. Fleck, M. Wand, D. Staneker, and W. Straßer, "First experiences with a mobile platform for flexible 3d model acquisition in indoor and outdoor environments—the wägele," in *Proceedings of the ISPRS Working Group V/4 Workshop 3D-ARCH 2005: Virtual Reconstruction and Visualization of Complex Architectures (ISPRS '05)*, Mestre-Venice, Italy, August 2005.

[30] P. Biber and W. Straßer, "The normal distributions transform: a new approach to laser scan matching," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, vol. 3, pp. 2743–2748, Las Vegas, Nev, USA, October 2003.

[31] S. Fleck, F. Busch, P. Biber, H. Andreasson, and W. Straßer, "Omnidirectional 3d modeling on a mobile robot using graph cuts," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '05)*, pp. 1748–1754, Barcelona, Spain, April 2005.