LoCaF: Detecting Real-World States with Lousy Wireless Cameras

Benjamin Meyer, Richard Mietz, Kay Römer
Institute of Computer Engineering, University of Lübeck, Germany
Email: {meyer,mietz,roemer}@iti.uni-luebeck.de

The definitive version was published in the Proceedings of IEEE DCOSS 2012.

***Abstract*—The Internet of Things (IoT) integrates wireless sensors to provide online and real-time access to the state of things and places. However, many interesting real-world states are difficult to detect with traditional scalar sensors. Tiny wireless camera sensor nodes are an interesting alternative as a single camera can observe a large area in great detail. However, low image resolution, poor image quality, and low frame rates as well as varying lighting conditions in outdoor scenarios make the detection of real-world states using these lousy cameras a challenging problem. In this paper we introduce a framework that addresses this problem by providing an end-to-end solution that includes energy-efficient image capture, image enhancement to mitigate low picture quality, object detection with low frame rates, inference of high-level states, and publishing of these states on the IoT. The framework can be flexibly configured by end-users without programming skills and supports a variety of different applications.**

I. Introduction

An increasing number of sensors embedded into appliances such as mobile phones, sensor nodes, and sensor networks are being connected to the Internet. The resulting Internet of Things (IoT) provides online and real-time access to the state of real-world objects and places (e.g., length of waiting queues or traffic jams; occupancy of rooms, buses, or parking spots). Experts predict that over the next ten years, the IoT will grow to include tens of billions of embedded sensors, thus surpassing the number of general-purpose computers that connect to the Internet today [1]. This massive growth is only possible if end-users can connect embedded sensors and publish their output on the IoT without help from experts – in analogy to the existing Web 2.0, where users not only consume, but actively contribute information.

Currently, the majority of wireless sensors measure scalar properties such as temperature, brightness, or acceleration. However, detecting high-level states of places with such simple sensors is often difficult if not impossible, or requires many sensors. Examples include measuring the length of waiting queues and traffic jams, or the occupancy of rooms and parking lots. Therefore, miniaturized wireless cameras (e.g., [2]–[4]) are recently gaining momentum as a single camera can observe a large scene and provides more detail than a scalar sensor.

However, extracting accurate high-level states of places from images captured by these cameras is a difficult task. Due to the small size of those cameras, their optics are simple (e.g., small aperture, fixed focus) and image quality is low. Further, they need to capture and wirelessly transmit images for months or years on a single battery. Therefore, image resolution and framerate are very low (typically a few images per minute with a resolution of few tens of kilopixels per image). Finally, the observed scenes are often exposed to unpredictable and varying lighting conditions.

Due to those difficulties, general-purpose solutions are difficult to provide. Rather, the focus has so far been on custom solutions for specific applications whose development requires substantial expertise and effort in embedded programming (to record and extract images from wireless cameras), image processing (to extract high-level states), and networking (to publish those states on the IoT). However, if the IoT is to scale to billions of devices, we need flexible and reusable approaches that support a wide range of different applications and can be used without programming skills.

To this end, this paper contributes LoCaF (Lousy-Camera Framework) – a flexible framework to extract and publish high-level states of places on the IoT using low-quality wireless cameras. LoCaF has been specifically designed to address the challenges resulting from resource and energy constraints, poor image quality and low frame rates and aims to support a wide range of applications under different lighting conditions. Specifically, it includes energy-efficient image capture, image-enhancing filters, object detectors, a rule language for state inference, and methods to publish states on the Web. All components can be easily configured and composed into a workflow by users to meet the requirements of a specific application.

The design rationale, architecture, and components of LoCaF are introduced in Sect. II. Implementation aspects are briefly discussed in Sect. III. An evaluation of two different application scenarios is given in Sect. IV. We discuss related work in Sect. V before concluding the paper with Sect. VI.

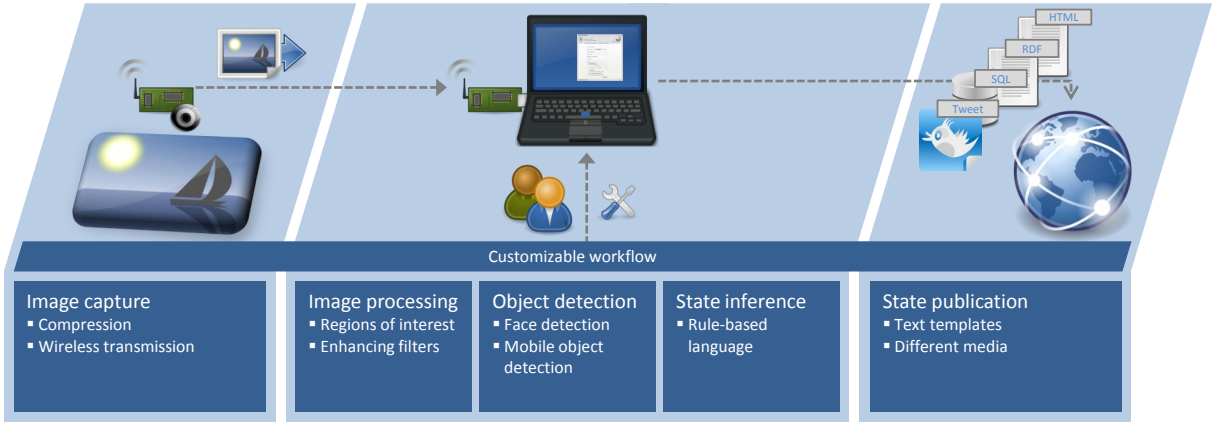

Fig. 1: LoCaF architecture.

II. System Architecture

Two typical applications we aim to support with LoCaF (and which will also be considered in the evaluation in Sect. IV) are monitoring of the occupancy of multiple parking spots in a parking lot and monitoring of the occupancy of a room, each with a single wireless camera. For the parking application, the positions of the different spots have to be identified in the image, the image has to be enhanced to compensate variable lighting conditions and to enhance texture in low-res images, detect movement of cars into and out of spots using only few images, infer a high-level state (i.e., occupied or free) for each spot, and publish this state on a Web page. The workflow for the room occupancy application is similar, but differs in important details. For example, instead of cars in an outdoor environment, the presence of people has to be detected in an indoor setting.

Consequently, and as summarized in Fig. 1, LoCaF provides functionality for capturing series of images in an energy-efficient way (Sect. II-A), for enhancing images to deal with low image quality (Sect. II-B), for detecting objects with low frame rates (Sect. II-C), for inferring high-level states from detected objects (Sect. II-D), and for publishing those states on the IoT (Sect. II-E). In order to support a wide range of different applications, alternative solutions for each task need to be provided that can be flexibly configured and combined into a workflow by end users without programming skills.

As illustrated in Fig. 1, LoCaF involves three types of computing platforms: low-quality camera nodes (left) that capture images and communicate wirelessly with a base station (middle) that connects to the Internet (right) where high-level states of places extracted from images are published. A key question in the design of the LoCaF architecture is *where* (i.e., on which of the involved computing platforms) to execute which of the above tasks. As wireless communication typically dominates the power consumption of sensor nodes, a common approach is to extract high-level states from raw sensor time series on the sensor nodes to reduce the amount of data that needs to be transmitted wirelessly. While some wireless camera platforms include substantial memory and processing resources (e.g., [3], [4]) to allow execution of basic image processing algorithms on the camera, we intend to support a wide range of platforms, in particular also ones at the lower end of the spectrum (e.g., [5]). The latter are typically equipped with a simple microcontroller and few tens of kilobytes of RAM. Here, advanced image processing (which is necessary to deal with low image quality) on the camera node is infeasible, i.e., images need to be transmitted to the base station for further processing. However, many camera modules do support lossy compression of images in hardware with small power consumption to substantially reduce the size of the transmitted images. Therefore, LoCaF performs all image processing on the base station, which also simplifies reconfiguration of the image processing workflow at runtime as the code executing on the camera node does not have to be modified.

In the following sections we describe each of the tasks performed by LoCaF shown on the bottom of Fig. 1 and discuss how the specific challenges resulting from constrained energy and resources as well as from low image quality and frame rate are addressed. For each task, complementary approaches are supported to cover the application space. As discussed in Sect. III, a graphical user interface supports easy selection and configuration of suitable approaches and their composition into a workflow in order to realize a specific application.

A. Image recording

As motivated above, wireless camera nodes do record and compress images and send them to the base station for further processing. However, as wireless communication is expensive in terms of energy consumption, the number of transmitted images needs to be minimized. To this end, LoCaF supports *event-triggered* and *time-triggered* image capture. The latter mode is the simpler one and an image is captured at a regular interval defined by the user.

The event-triggered mode exploits more energy-efficient sensors to detect if the scene might have changed and only then

captures an image. In particular, we exploit a passive infrared (PIR) sensor to detect movement in the scene. The energy consumption of the PIR sensor is negligible compared to the energy consumption of the radio and camera. A challenge in this context is that not every movement event reported by the PIR sensor represents a significant change in the scene. Therefore, a state machine is used which only triggers an image recording if a sufficient number of motion events is observed in a certain time interval.

B. Image processing

The images received for the camera are affected by unpredictable and varying lighting conditions and other disturbances (e.g., moving leaves or flags) that may impair state detection. Further, image resolution and quality are low, not only due to the low-quality cameras but also due to the use of lossy compression algorithms such as JPEG. Therefore, LoCaF provides four classes of image processing algorithms to enhance image quality in preparation for object detection: region selection, lighting compensation, texture enhancement, and contrast enhancement.

Region selection. Often, the potential locations of objects in the image are known in advance (e.g., parking spots in a parking lot) while other image regions (e.g., driveways) need to be ignored during object detection. Therefore, the user can mark several polygon regions of interest on an image and give them descriptive names. Later object detection will be constrained to those regions. During runtime all pixels outside selected regions are colored black to obtain a binary threshold image using the *Scan-Line-Fill* algorithm [6], which performs a line-by-line scanning on an image for recognizing coherent blobs.

Lighting compensation. The illumination has a large influence on the image scene and a sudden change of the lighting conditions in the image (e.g., shadows) can lead to inaccurate object detection. To solve the problem of inhomogeneous lighting, a homomorphic filter [7] is included in the framework. The filter is based on an illumination-reflectance model, which describes the image production as a process of lighting and absorption. As a result of the algorithm, shaded regions are lightened and overexposed regions are adjusted to a normal value such that the brightness is normalized and the contrast is increased. Thereby, changing lighting conditions have a smaller effect on state detection.

Texture enhancement. Due to the low image resolution, structures in the scene are often poorly visible in the image. Therefore, an anisotropic filter [8] is implemented to enhance the texture quality of the small images generated by the sensor node to improve the detection of distant objects. It smoothes the image to reduce noise, but preserves important surface features such as sharp edges or corners by applying direction-dependent smoothing. Thereby, objects seem sharply focused from a large distance in the observed scene.

Contrast enhancement. Additionally, LoCaF provides algorithms to enhance the quality of the pictures with regard to the ability to distinguish between different objects traversing the scene. Because the contrast in the scene is pivotal for the ability to recognize objects, different types of histogram equalizations are integrated to reallocate the intensity values of the whole image such that neighboring pixels have a statistically better contrast. For generating better results, the equalization is performed adaptively by computing the reallocation function in local neighborhoods of pixels. This so-called adaptive histogram equalization *(*AHE) generates an image with higher contrast at the object borders and is a perfect preparation for a segmentation algorithm. To avoid the generation of artificial features in homogeneous image segments by the local histogram equalization, the generated contrast can be limited. This is the so-called contrast-limited adaptive histogram equalization (CLAHE) [9].

C. Object detection

High-level states of places are typically related to the presence or absence of certain objects (e.g., people in a waiting queue, cars in a parking lot). If the appearance of those objects is known in advance, algorithms (e.g., face recognition) can be designed that detect objects using only a single image frame. While LoCaF includes such algorithms as described below, we aim to support a wide range of applications and therefore need to detect objects in a more general way based on their mobility. However, this typically requires multiple image frames. As the frame rate is low, we use an algorithm that requires only few consecutive frames. Finally, we need to perform blob detection to obtain number and area of detected objects in the image, which serves as input for state inference.

Face detection. We used the efficient face detection algorithm developed by Viola and Jones [10] which provides high detection accuracy. It exploits typical features of the human face, e.g., the eye regions typically appear darker than the nose and cheek regions. As illustrated in Fig. 2, these features are detected using so-called *Haar base functions*, which consist of the difference of the sums of pixel values in rectangular regions of the same size. To efficiently compute these features, a special image representation called integral image is used. A pixel (x, y) in the integral image is defined as the sum of all pixels above and to the left of (x, y) in the original image. In this representation, Haar features can be computed by adding and subtracting few pixels in the integral image.

The features are computed for each possible position in the image. To improve computation speed, a cascade structure of multiple features in order of increasing complexity is used, i.e., if a feature is found at a certain image position, further features are evaluated at that position to increase confidence in the detection.

As a result, all faces in an image are found and all pixels belonging to detected faces are marked as foreground pixels. However, if persons are far from the camera, wear sunglasses, or are not directly looking into the camera, they cannot be recognized by the algorithm and the more general algorithm described below has to be used instead.

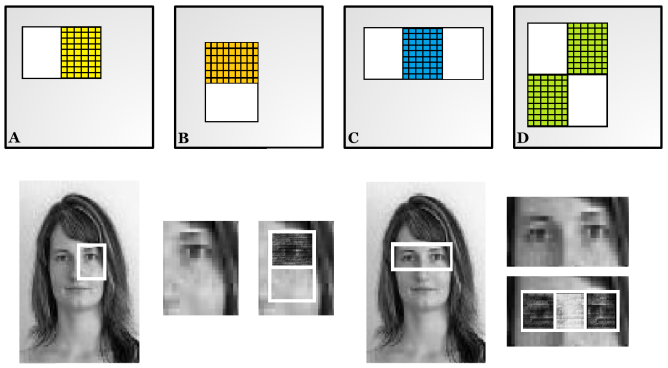

Fig. 2: Haar feature types: For classification the sums of pixel values from the patterned rectangles are subtracted from the white rectangles. The features are based on the gray level differences in the human face such as the difference between the shaded eye section and the nose.

Mobile object detection. This algorithm exploits the mobility of objects on a pseudo-static background to detect them. It is based on an adaptive background subtraction algorithm originally developed by Stauffer and Grimson [11]. Different from previous algorithms, the image pixels are modeled as a mixture of a small number of Gaussians. Each pixel in a new image is assigned to one of those Gaussians by checking if the pixel value is within 2.5 times the standard deviation. Each Gaussian is also assigned a weight based on the number of assigned pixels using a moving average filter with learning rate α. If a pixel cannot be represented by any Gaussian, the mean value of the closest Gaussian is set to the pixel value, variance is set to a large value, and weight is set to an initial small value.

All Gaussians with a weight above a certain threshold, respectively all pixels assigned to them, are considered as background, while all other pixels are considered as foreground. As static image parts increase the weight of their Gaussians over time, static image parts will be classified as background. All foreground pixels are assumed to belong to detected objects.

The algorithm has the advantages that foreground objects are learned very fast (i.e., only few images are needed) and the objects are not restricted to a given shape. If an object is entering a scene, it will be recognized until it is not moving for a certain time, depending on the value of α. Also, uniform background movements such as waving trees or clouds are categorized as background processes depending on the number of used Gaussians.

Blob detection. In general, object detection algorithms mark individual pixels as belonging to an object (i.e., foreground). We still need to group these pixels into continuous objects to obtain their number and size, which forms the input for state inference described below. For this, a blob detection algorithm is used which groups continuous regions of foreground pixels into a blob [12]. The number of distinct blobs and the fraction of the area of the region (defined by the user as described in Sect. II-B) containing them are then passed as input to the state inference.

D. State inference

As a result of object detection, we obtain the number of distinct objects and the area they cover for each image region defined by the user. These are important hints on the state of a place. For example, to detect the number of persons in a room with a ceiling-mounted camera, we can count distinct objects as the persons won't overlap in the image. However, if we want to measure the length of a waiting queue with a wall-mounted camera, we would rather leverage covered area as persons will overlap in the image and cannot be identified as distinct objects. That is, how exactly number and area of objects map to a high-level state of a place, very much depends on the application.

Therefore, LoCaF provides a rule-based language to map number and area of objects to a set of discrete high-level states. One state is computed for each image region defined by the user. The language supports *state-based* and *event-based* mapping. With state-based mapping, the high-level state is only a function of the current number and area of detected objects and does not depend on the history. For example, the state of the room is occupied if there is at least one human (i.e., mobile object) present, otherwise it is empty.

The syntax for this approach is $\{$count|area$\}$:map:l:u:$state$ where $state$ is the user-defined name of the high-level state, l and r are inclusive lower and exclusive upper bounds on the number or area of detected objects in that image region. Listing 1 encodes the above example (-1 stands for infinity). Further rules can be added to define more fine-grained occupancy states (e.g., empty, single person, small group, full).

Listing 1: Example of two *map rules* for a meeting room.

```
1 count:map:0:1:free
2 count:map:1:-1:occupied
```

With event-based mapping, the high-level state is a function of the previous high-level state and an event, where the event is defined by a change in number or area of detected objects. This approach is motivated by the fact that objects are only detected as long as they move at least occasionally by the algorithm described in Sect. II-C. For example, a car will only be detected while entering and leaving a parking spot, but not while standing still. Here, the detection of the mobile car (i.e., the event) triggers a state change to occupied if the state was empty before and vice versa. The event is defined by giving a threshold on the number or area of detected objects. The event triggers if the number or area of objects rises from a value below the threshold to a value above (i.e., rising edge).

The syntax for this approach is $\{$count|area$\}$:switch:$prevstate$:t:$state$ where t is the threshold on either number or area of detected objects, $state$ is the name of the state to switch to when the current state is $prevstate$ and the threshold is exceeded. Listing 2 encodes the above example. Further rules can be added for additional states.

Listing 2: Example of a *switch rule* for a parking spot.

```
area:switch:free:80:occupied
area:switch:occupied:80:free
```

E. State publishing

Finally, the computed state needs to be published. Depending on the application, states may have to be published in different formats (e.g., plain text, HTML) over different media (e.g., Web, file, Tweet, database).

LoCaF supports text-based templates (e.g., plain text, HTML, RDF) which contain placeholders \$region$N$, \$state$N$, \$time$N$, where the name of the region, its state, and a timestamp are filled in. Regions are consecutively numbered and N is the number of the requested region. Alternatively, a loop construct may be used, see Listing 3 for an example. The text inside the loop is duplicated for each region and the variables are replaced with the information of that region.

Listing 3: Example of a HTML-template making use of the loop-construct and variables.

```
<?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
 <head><title>State of entities</title></head> <body>
    <h1>Region states</h1>
$forLoop[<p>Region $region has $blobs blobs and is in state $state.</p>]
 </body>
</html>
```

LoCaF creates an instance of this template whenever a state changes and then publishes the filled-in template by uploading it via FTP (user needs to specify the server address as well as login data) or by posting it as a Tweet on the Twitter microblogging platform. For each region, a separate Tweet is posted due to the limitation of 140 characters per Tweet. Alternatively, LoCaF can insert states into an SQL database (user needs to specify database address and login data). For every region a new row consisting of region name, time, and state is appended to a predefined database table.

III. Implementation

We use an iSense sensor node [5] equipped with a low-cost camera, capable of taking colored JPEG-compressed pictures with a resolution up to 640x480 pixels, and a PIR sensor. The captured images are split into multiple packets and transferred wirelessly to a gateway sensor module connected via USB to a netbook.

The main framework component executing on the netbook is written in C++ and uses the *Open Computer Vision* library with algorithms and data structures for image processing. It features a graphical user interface shown in Fig. 3 which allows the user to configure the image capture mode and resolution. For a resolution of 320x240 pixels, up to 15 frames can be captured per minute.

Further, the interface allows the user to define an image processing workflow by selecting appropriate algorithms, setting their parameters, and by defining the order in which they shall be executed. As most filters are non-linear, filters can be added multiple times to the workflow and their order can also be changed later. Likewise, rules for state inference can be entered and the publishing mode can be selected and configured. The

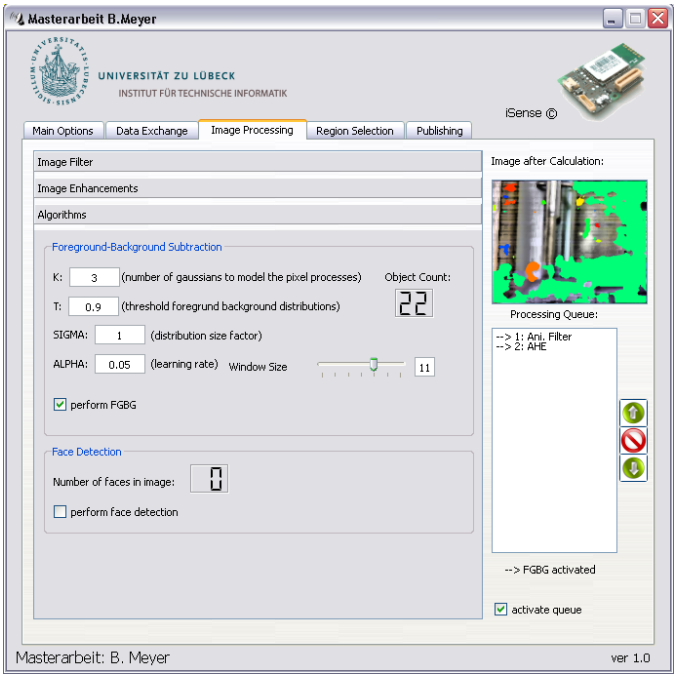

Fig. 3: Screenshot of the GUI. The current tab shows how to customize the filters (left) and the sequence of filters to apply (right).

GUI also displays raw and processed images to allow for visual inspection of processing results. All settings can be changed during operation to optimize detection accuracy.

IV. Evaluation

For the evaluation of our framework, we investigate two scenarios. In the first scenario we install the camera sensor node in a lecture hall in our university observing students during a lecture. An outdoor scenario where the camera is monitoring a parking spot is our second use case.

A. Lecture Hall Scenario

While the ultimate goal of this scenario would be to infer a high-level state (e.g., empty, full) of a lecture hall, here we investigate the exact number of students as the high-level state.

To recognize as many students as possible, we install the camera node centered in front of the seat rows. We use the background subtraction approach to detect objects (i.e., students) instead of the face recognition because the faces of students in the back rows are only a few pixel in height and width and can thus hardly be detected by the face recognition algorithm. Initially we use the raw images taken by the camera without any enhancing filters. In a second step, we play back the recorded images but this time use different combinations of image enhancement filters to analyze their impact on detection accuracy.

We split the scenario into three phases. In the first minutes, shortly before the lecture starts, students are entering the lecture hall resulting in high mobility. The algorithm has to detect many objects and ideally distinguish nearby students

instead of treating them as a single object. In the second phase, during the lecture, the students are sitting and listening to the professor. Very little movement is expected and thus the correct number of students is difficult to estimate. After the lecture, the students leave the room, again resulting in substantial mobility, until the room is empty.

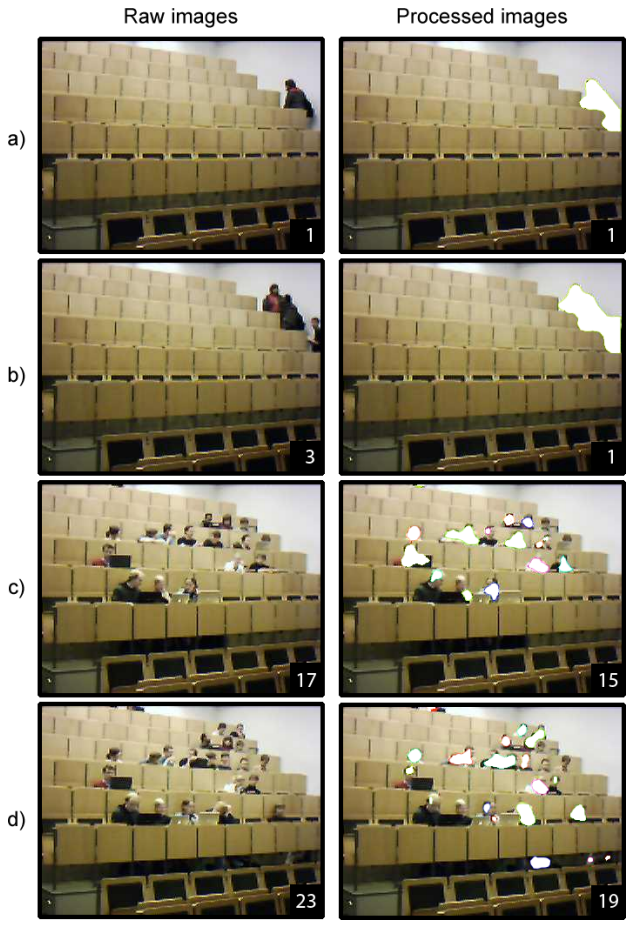

Fig. 4: Raw images and images with blobs (manually colored white for better visibility) from object detection. The number in lower right corner indicates the actual number of persons in raw images and number of blobs (estimated number of persons) in processed images.

Figure 4 shows a selection of raw images along with processed images overlaid with blobs from the object detection phase. As can be seen in row a) in Figure 4, the object detection clearly identifies the first student entering the lecture hall. Figure 5 compares the estimated and ground truth number of students over time. One can see that the algorithm over- as well as underestimates the correct number. The underestimation is due to the detection of two or more students as one object (row b) in Figure 4). The overestimation can be explained by wrong detections resulting from foreground pixels from an old image not yet being detected as background, although no object is related to these pixels anymore (row c) in Figure 4) or due to the legs of students being detected as independent objects because part of the body is hidden by the table (row

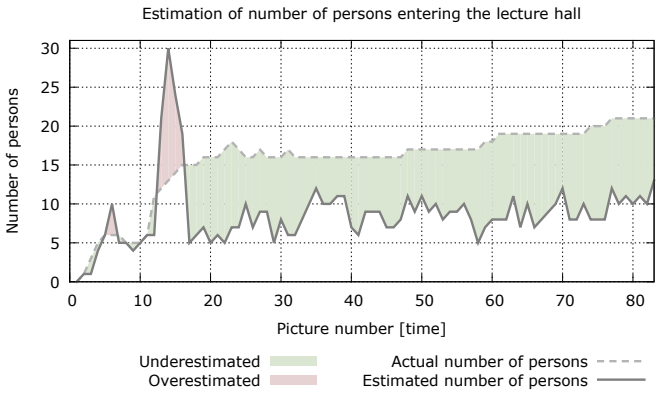

Fig. 5: Estimation of number of persons entering the lecture hall.

d) in Figure 4). Although the absolute estimation error is relatively high with a maximum overestimation error of 130%, a maximum underestimation error of 70%, and an average error of 48%, high-level states such as empty or occupied (cf. Listing 1) could be correctly identified. As discussed below, the performance can be substantially improved by enhancing the raw image with different filters.

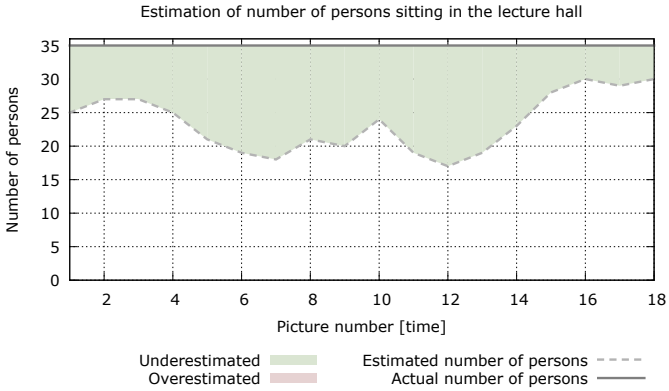

Fig. 6: Estimation of number of persons sitting in the lecture hall.

In the second phase the number of students stays constant. As Figure 6 shows, the estimation is always lower than the actual number. The underestimation is due to some students not being detected as moving objects but as background as well as some students hiding behind a laptop thus not being captured by the camera. The average estimation error of 54% is higher than in the first phase.

The last phase, when the lecture is over, is characterized by students abruptly leaving the room. In contrast to the first phase, where students came in dribs and drabs, all students walk out at the same time. One can see from Figure 7 that these movements are a challenge for the algorithm. The background subtraction needs some time (i.e., frames) to adapt to the background again. At point 11 in time the state inference

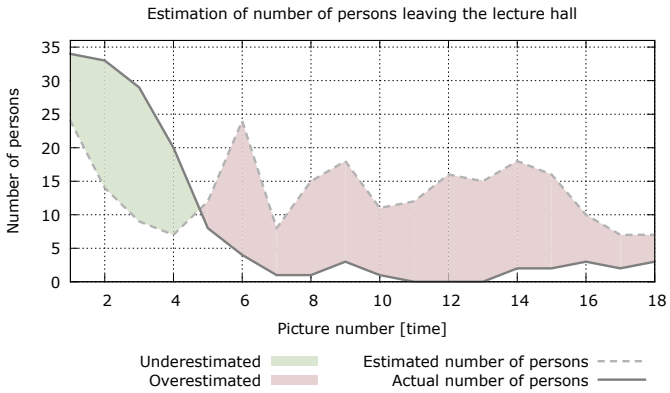

Fig. 7: Estimation of number of persons leaving the lecture hall.

would, with the rules of Listing 1, incorrectly tag the lecture hall as *occupied* although it is already *free*. However, this wrong state would only last for some minutes until enough pictures are received and the algorithm has adapted to the static background again. At the end, one can see this adaptation when estimated people count decreases.

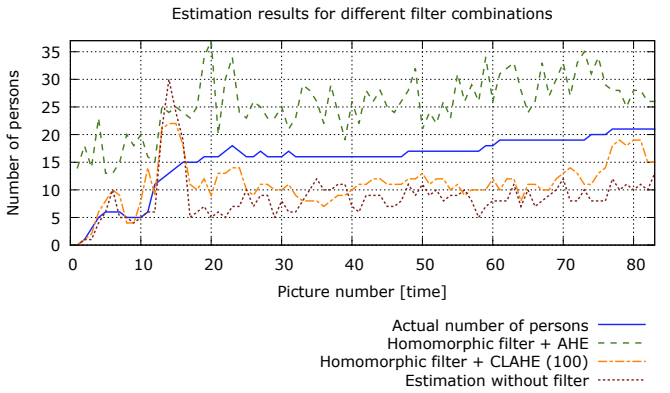

Fig. 8: Estimation using different combinations of filters.

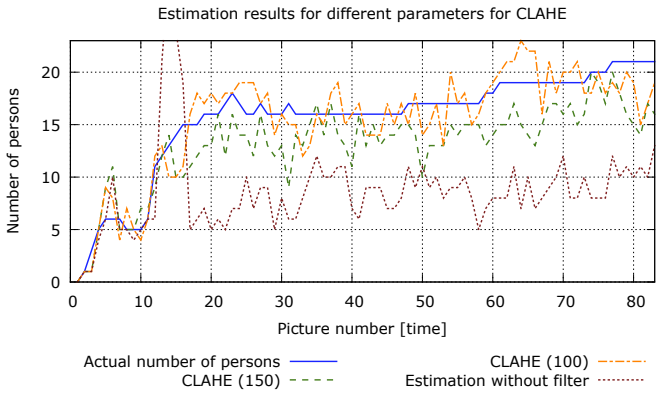

Fig. 9: Estimation using same filters but different parameters.

To investigate how image enhancement techniques can increase the accuracy of estimation, we play back the previously recorded images, but apply different combinations of filters respectively parameters before object detection is executed.

The quality of estimation is strongly dependent on the choice of filter combinations as one can see in Figure 8. While a homomorphic filter followed by AHE overestimates, a homomorphic filter with CLAHE mostly underestimates the actual number of persons. This is caused by the different calculation of the reallocation functions in the image plane. AHE enhances existing image contours in the whole image. Thereby, fragments in homogeneous image parts could occur and lead to overestimations in the object detection. CLAHE has a limitation for the enhanced contrast and prevents the negative effect of AHE in homogeneous images, depending on its parameters. From the results we conclude that CLAHE is more practical in homogeneous environments and AHE in more cluttered environments.

Not only is the right orchestration of filters crucial for good results, but also the correct parameterization of the used filters. Figure 9 compares the results from estimation with CLAHE filters but with a different number of used color bins. The results are not as widely spread as in Figure 8, but still show that different parameters can affect the result significantly. Both plots show that a wisely chosen workflow can reduce the estimation error significantly. The best result, an average error of 12%, is achieved with the CLAHE filter and a parameter of 100 bins for the 255 color values.

As the user interface shows the processed images after each filter and supports reconfiguration of the filter chain during operation, a good configuration can be found empirically.

B. Parking Spot Scenario

In the parking spot scenario we evaluate LoCaF under outdoor conditions. We install the camera on the top deck of a car park, monitoring two adjacent parking spots. Figure 10 shows raw images of a parking car along with the processed images without any filters. As one can see, nearly the complete car is detected. Only some parts of the windows and lights are not detected due to transparency of these parts. Thus, as can be seen in row b), two blobs are detected for the car. We define a region containing only the left parking spot and use the *switch rule* approach with the rule set from Listing 2. As the car parks, the object area threshold of 80% is exceeded in the defined region and the state switches to *occupied*. As the car is static, the area covered by mobile objects drops below the threshold again. As soon as the car drives away, the threshold is exceeded again and the state of the parking spot switches back to *free*. A similar rule but with number of objects instead of covered area and 1 as a threshold would not perform well as also small detected objects such as a pedestrian walking by would cause a state change resulting in a false state classification of the parking spot.

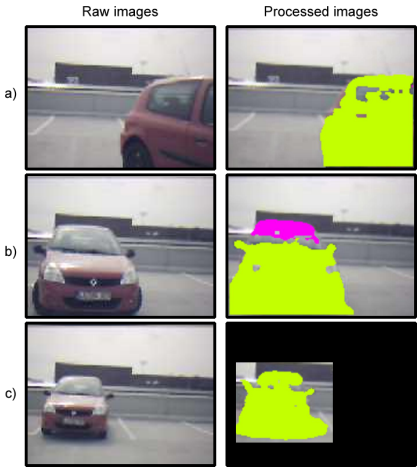

Fig. 10: Raw images and images with blobs from the object detection of the Parking-spot scenario. The processed image in row c) shows also the region defined for the left parking spot.

V. Related Work

Recently, camera sensor networks are gaining momentum and several papers have been published on camera sensor nodes, object detection, state inference, integrating sensors into and publishing data on the IoT. However, the large majority of existing work focuses on specific applications. As LoCaF aims at providing a flexible framework that supports multiple applications in and end-to-end fashion, we focus our discussion on related work that integrates multiple of the above aspects.

A. Platforms

Smart camera nodes differ largely in their computational and storage resources. While some systems (e.g., Citric [3], Cyclops [4]) provide powerful hardware, capable of processing images on the node, other systems such as iSense [5] do only offer a microcontroller with few kilobytes of RAM. Some systems such as [13] are based on a two-tier architecture with low-cost camera sensor nodes (Cyclops) on the lower tier and more powerful webcams and computers on the upper tier. The upper tier only wakes up and performs object detection and recognition if the lower tier detects an event of interest. Our framework intends to support all types of cameras, especially also ones with very constrained computational resources, and therefore minimizes processing on the camera nodes.

B. Middleware

A number of middleware systems have been proposed to integrate sensors into the Web and to process the resulting data streams. However, they either do not support image sensors, or require that the user implements image processing operations himself.

In [14], Aberer et al. present Global Sensor Networks (GSN), a scalable infrastructure to integrate heterogeneous sensor networks with the Internet. They introduce the virtual sensor abstraction which aggregates sensor time series from multiple other sensors in real time using a streaming SQL query. Thus, the system is suitable for end users who are capable of formulating SQL queries. However, the system is intended for scalar sensors – image processing is not supported.

Pachube [15] is an online platform for managing feeds from sensors. An API allows creating, deleting, updating, and viewing data streams for sensors. Thus, a user can either actively publish sensor data or passively consume data published by others to create new applications. Hence, Pachube offers a middleware layer to create IoT applications. However, processing of sensor data to detect objects in images, for example, is left to the user.

With the web frontend of ParaImpu [16], the user can create mashups, called *connections*, by virtually wiring *sensors* and *actuators*. The user can either use real sensors or virtual input such as posts from Twitter. Similar to *sensors*, *actuators* can be created for real actuators or websites such as Twitter, Facebook, or Google Calendar. The user can write rules to define how received data from the *sensor* should be filtered and mapped to an output for the *actuator*. Although the system allows combining virtual as well as real sensors and actuators, there is no possibility for using other than text input.

IrisNet [17] employs a two-tier architecture to process data from several multimedia sensors. The lower level consists of the sensors connected to *Sensing Agents (SA)* responsible for data collection and filtering. On the upper-layer, *Organizing Agents (OA)* take care of data storage and querying. Application-specific code needs to be provided by the user for the SA to extract meaningful information from raw multimedia data, for the OA to provide a storage scheme, and for an application user interface. Thus, the system is very flexible but processing of sensor data, e.g., to detect objects in images is left to the user.

Deep Vision [18] uses a network of Cyclops vision sensors, capable of doing simple moving object detection and motion flow estimation, to monitor a certain region. Additional information (e.g., location) of the vision sensors is stored in a database. The user can interact with the network by querying a central computer in a high-level SQL-like syntax. Queries are transformed to low-level image processing operations and routed to the nodes. Additionally, information from the database is retrieved. All information is collected at the central computer and returned to the user. Due to the on-node processing with resource-restricted hardware, the system only uses simple detection algorithms not suitable for use in scenarios with changing environmental conditions. Additionally, the system doesn't allow to postprocess the gathered information in terms of inferring high-level states and publishing on the Internet. Finally, the user needs to learn a query language to interact with the system.

C. Applications

Several applications exploiting smart cameras have been described in the literature. However, they are designed for a

very specific purpose and cannot be applied to a wide range of different applications and scenarios as it is the purpose of LoCaF.

Facet [19] is a software framework that runs on mobile phones with cameras to track mobile objects using multiple phones installed at fixed locations in a building. A simple background subtraction algorithm is used to detect mobile objects. The system is limited to a single application (tracking) and does not support detection and publishing of high-level states as LoCaF does. Also, the algorithms are not suitable for outdoor applications due to variable lighting conditions.

In [20], Bamis et al. describe BehaviorScope, a system using multi-modal wireless sensors such as PIR sensors and cameras deployed in houses to monitor elderly people. Depending on the application and privacy requirements, raw or preprocessed sensor data is transmitted to a central server to infer the activities of the inhabitant. Thus, the system is tailored to monitoring people, whereas LoCaF supports flexible detection and publishing of a range of different high-level states of places.

MiceNet [21] uses camera-equipped sensor nodes to track lab mice in cages. The JPEG-compressed images taken by the camera are transmitted wirelessly to a base station where further processing is performed to analyze the activity of different mice. Although the architecture and used hardware is similar to our approach, it is tailored to mice experiments and thus limited to tracking, whereas LoCaF supports flexible detection and publishing of a range of different high-level states of places.

TigerCense [22] describes a deployment of camera- and PIR-equipped sensor nodes to help researchers count and analyze the movement of tigers. The camera of a sensor node is triggered by movements detected by the PIR-sensor and images are stored on an SD-Card. As soon as a connection to the gateway exists, pictures are transmitted wirelessly and afterwards uploaded to a database in the Internet. Although the prototype is not limited to monitoring of tigers, it is very limited by only taking pictures without any image enhancement or automatic post-processing.

VI. Conclusion

Within a decade, embedded devices will likely form the majority of the participants on the Internet. The states of real-world things and places observed by embedded sensors will be available online and in real-time in the resulting Internet of Things. We have presented the LoCaF framework which allows to detect and publish high-level states of places using low-quality wireless camera sensors. Our emphasis is on providing an end-to-end solution that supports a range of different applications, to be used by domain experts without programming skills. With LoCaF, users can select and configure function blocks and combine them into a complete workflow from energy-efficient image capture to publishing of high-level states on the Web. Thereby, users can not only consume, but also actively contribute real-time data to the Internet of Things.

Acknowledgment

This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Förderkennzeichen 01BK0905, GLab). The authors alone are responsible for the content of the paper.

This research has been also partially financed by the Cluster of Excellence 306/1 Inflammation at Interfaces (Excellence Initiative, Germany, since 2006).

References

[1] N. Lomas, "Online gizmos could top 50 billion in 2020," Bloomberg Businessweek, http://www.businessweek.com/globalbiz/content/jun2009/gb20090629_492027.htm, Jun. 2009.

[2] A. Rowe, A. G. Goode, D. Goel, and I. Nourbakhsh, "Cmucam3: An open programmable embedded vision sensor," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-13, May 2007.

[3] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Y. Yang, C. Yeo, L.-C. Chang, J. D. Tygar, and S. S. Sastry, "CITRIC: A low-bandwidth wireless camera network platform," in *ICDSC*, Sep 2008, pp. 1–10.

[4] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: in situ image sensing and interpretation in wireless sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 192–204.

[5] C. Buschmann and D. Pfisterer, "isense: A modular hardware and software platform for wireless sensor networks." 6. Fachgesprch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Tech. Rep., 2007.

[6] M. R. Dunlavey, "Efficient polygon-filling algorithms for raster displays," *ACM Trans. Graph.*, vol. 2, pp. 264–273, October 1983.

[7] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[8] K. Hildebrandt and K. Polthier, "Anisotropic filtering of non-linear surface features," *Computer Graphics Forum*, vol. 23, pp. 391–400, 2004.

[9] A. M. Reza, "Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement," *J. VLSI Signal Process. Syst.*, vol. 38, no. 1, pp. 35–44, aug 2004.

[10] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, may 2004.

[11] C. S. und W.E.L Grimson, "Adaptive background mixture models for real-time tracking," *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 10.1109/CVPR.1999.784637, 1999.

[12] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[13] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu, "SensEye: a multi-tier camera sensor network," in *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*. New York, NY, USA: ACM, 2005, pp. 229–238.

[14] K. Aberer, M. Hauswirth, and A. Salehi, "Global Sensor Networks," Tech. Rep., 2006, submitted to IEEE Communications Magazine.

[15] (2011) The internet of things real-time web service and applications - pachube. [Online]. Available: https://pachube.com/

[16] A. Pintus, D. Carboni, and A. Piras, "The anatomy of a large scale social web for internet enabled objects," in *Proceedings of the Second International Workshop on Web of Things*, ser. WoT '11. New York, NY, USA: ACM, 2011.

[17] J. Campbell, P. B. Gibbons, S. Nath, P. Pillai, S. Seshan, and R. Sukthankar, "Irisnet: an internet-scale architecture for multimedia sensors," in *Proceedings of the 13th annual ACM international conference on Multimedia*. New York, NY, USA: ACM, 2005, pp. 81–88.

[18] M. Rahimi, S. Ahmadian, D. Zats, J. Garcia, M. Srivastava, and D. Estrin, "Deep vision: Experiments in exploiting vision in wireless sensor networks," submitted for publication.

[19] P. Bolliger, M. Köhler, and K. Römer, "Facet: towards a smart camera network of mobile phones," in *Proceedings of the 1st international conference on Autonomic computing and communication systems*, ser. Autonomics '07, ICST, Brussels, Belgium, 2007.

[20] A. Bamis, D. Lymberopoulos, T. Teixeira, and A. Savvides, "The behaviorscope framework for enabling ambient assisted living," *Personal Ubiquitous Comput.*, vol. 14, pp. 473–487, September 2010.

[21] G. Stamatescu, K. Römer, R. Ludwig, S. M. Ibrahim, and V. Sgarciu, "Work in progress: Micenet: Monitoring behaviour of laboratory mice with sensor networks," in *DCOSS*, 2011, pp. 1–3.

[22] R. Bagree, V. R. Jain, A. Kumar, and P. Ranjan, "Tigercense: Wireless image sensor network to monitor tiger movement," *P.J. Marron et al. (Eds.): REALWSN 2010, LNCS 6511*, pp. 13–24, 2010.