Dipl.-Ing. Bernhard Dieber

Resource-aware Reconfiguration of Visual Sensor Networks

Dissertation

zum Erlangen des akademischen Grades
Doktor der technischen Wissenschaften (Dr. techn.)

1. Begutachter: Univ. Prof. DI Dr. Bernhard Rinner
Institut für Vernetzte und Eingebettete Systeme
Alpen-Adria Universität Klagenfurt

2. Begutachter: Prof. Dr. Andrea Prati
Dipartimento di Progettazione e pianificazione in ambienti complessi
Università Iuav di Venezia

October 2013

Ehrenwörtliche Erklärung
für Masterarbeiten, Diplomarbeiten und Dissertationen

Ich erkläre ehrenwörtlich, dass ich die vorliegende wissenschaftliche Arbeit selbstständig angefertigt und die mit ihr unmittelbar verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus gedruckten, ungedruckten Quellen oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für wissenschaftliche Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die während des Arbeitsvorganges gewährte Unterstützung einschließlich signifikanter Betreuungshinweise ist vollständig angegeben.

Die wissenschaftliche Arbeit ist noch keiner anderen Prüfungsbehörde vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

(Unterschrift) (Ort, Datum)

Danksagung

Nach fünf Jahren als Dissertant gibt es sehr viele Menschen, denen ich Dank schulde. Hier kann ich leider nur einen kleinen Teil erwähnen.

Sehr dankbar bin ich allen Kolleginnen und Kollegen, die mit mir in der Pervasive Computing Group und am Institut für vernetzte und eingebettete Systeme gearbeitet haben. Hier habe ich neben tollen Kollegen auch wunderbare Freunde gewinnen können. Besonders erwähnen möchte ich hier Heidelies Aschbacher, die immer ein offenes Ohr für meine Probleme hatte und mir mit Rat und Tat zur Seite stand, sowie Thomas Winkler, der mir immer wieder neue Sichtweisen auf die Dinge eröffnet hat und für mich Freund und Vorbild war und ist. Meinen Bürokollegen Andreas Starzacher, Umair Ali Khan und Lukas Esterle danke ich für die angenehme Atmosphäre und die Freundschaft, die ihr mir entgegen gebracht habt.

Ganz besonders bedanken will ich mich bei Professor Bernhard Rinner für die Betreuung whrend meiner Dissertationsarbeit sowie für die vielen Möglichkeiten, die ich während dieser Zeit ergreifen durfte.

Ich wäre wohl nie so weit gekommen ohne die liebevolle Unterstützung meiner Eltern, die mir alle Möglichkeiten geboten haben um eine ausgezeichnete Ausbildung zu erhalten und für mich immer Stütze und Ratgeber waren.

Mein besonderer Dank gilt auch Anita Sobe, dafür dass sie eine so weite Strecke diese Weges mit mir gegangen ist.

Abstract

Cameras have become ubiquitous in our society. Not only are they built into things of everyday life like phones or cars, they are also used in surveillance of public and private spaces. Smart cameras which are able to process the captured video data onboard can be connected to large networks (visual sensor networks). Typically, visual sensor networks are composed of resource-limited devices. Here, coordination among the devices is needed to enable an efficient use of resources. Especially in cases where visual sensor networks are deployed without network or power infrastructure, resource-coordination is especially necessary to prolong the network lifetime.

This thesis investigates methods to reconfigure visual sensor networks to achieve a tradeoff between surveillance quality and resource consumption. First a formal description of the problem is presented. Based on this, the algorithmic design space of this problem is explored. A centralized approach using an evolutionary algorithm is presented for use in environments where no dynamic changes are expected. For environments with slow changes in the surveillance requirements, a distributed algorithm is described. For environments with fast changes, this algorithm is complemented with an object-handover algorithm.

Besides algorithms, software tools which support reconfiguration in visual sensor networks are presented. This includes a software framework for sequential data processing and a distributed middleware system.

In extensive evaluations the significant resource savings achievable with the presented algorithms are shown. Finally, the applicability of the developed evolutionary algorithm in a different field is shown. In a cloud server infrastructure, the load balancing is reconfigured to achieve cost savings without lowering the service quality.

Contents

Chapter 1

Introduction

Cameras have become ubiquitous in our society. Cameras are in phones, TVs and many other electronic devices. In addition, there is also a high number of fixed mounted cameras which are e.g. used for video surveillance of public and private spaces, highways or other areas. Traditionally, those cameras are delivering live images to a central place where they are either recorded or viewed by operators. As the number of cameras increases, the amount of information delivered at every time instant is becoming too high to be reliably processed by humans. The advent of so-called smart cameras[101] is targeted at the one hand at relieving human operators by automatically processing the captured videos and on the other hand to enhance surveillance systems with additional services. Unlike traditional CCTV cameras, smart cameras do not just transmit the recorded videos but they are equipped with processing capabilities and can extract relevant information from video data. This can be done automatically without the need to involve a user. The processed results are delivered to operators. This facilitates the work of an operator.

Sensor networks are a class of applications where numerous small sensors are deployed in an environment to sense certain types of parameters[3]. Those sensors form a network and exchange data in order to transmit their sensing results to a consumer. Progresses in miniaturization and wireless networking techniques have enabled the construction of large-scale sensor networks at low costs[2, 1]. A sensor network to detect a fire outbreak in a building could for example be realized using temperature sensors deployed all over the building which register a fast increase in temperature. Each sensor can be realized as an autarkic unit by incorporating energy supply and wireless networking technologies.

VSNs are a special type of sensor networks where cameras are used as sensors[93]. A camera typically has a limited Field of view (FOV), i.e., it cannot record data from 360° around it. Thus,VSNs form a subclass of sensor networks, the class of *directional* sensor networks. Smart cameras are often deployed in VSNs. Smart cameras are equipped with computational units and are able to process the collected data directly onboard[79].

VSNs have gained huge interest in the research community over the last decade. In traditional camera surveillance systems, operators have to view the streams coming from multiple cameras in order to detect relevant events. This procedure is exhausting and error prone. By composing camera net-

works from smart cameras, more intelligence can be brought to the network, thus helping users to concentrate only on important information[78].

Current VSNs rely on a certain infrastructure like wired access to the power grid or a fixed networking infrastructure. This limits their application to areas which provide this infrastructure. In order to be able to deploy a VSN without infrastructure, they must be able to operate from scarce resources.

1.1 Motivation

The motivation for this thesis is to investigate on operating VSNs without infrastructure and with no fixed energy supply. This demands for a method to optimize the resource consumption to maximize the network lifetime. In the SRSnet project, several approaches for energy optimization in a VSN have been evaluated.

1.1.1 Use Case: SRSnet

Smart Resource-aware multi-Sensor Network (SRSnet) is a cooperative research project of Lakeside Labs[1], the Groups of Pervasive Computing and Transportation Informatics at Alpen-Adria Universität Klagenfurt as well as the Italian companies EyeTech[2] and InfoFactory[3].

The major goal of this project is the development of techniques and methods to enable the deployment of VSNs in harsh environments, i.e., areas where no permanent power supply or wired networking is available. This poses several challenges that must be met.

First, the network must be able to be operated from unreliable power sources such as batteries and solar cells.

Second, the overall energy consumption of the system must be low enough to enable a suitable lifetime of the network. This requires the use of low-power hardware as well as a software-based approach to resource conservation.

Finally, the software executed at the sensor nodes must be tailored to be run on a platform with low computational power.

Project Organization

Our deployment use-case is the Carinthian Nationalpark "Hohe Tauern". In a park like that, the surveillance network is deployed in a biologically sensitive environment to autonomously detect various events such as intrusions into restricted areas, animal wildlife or perform tasks like visitor counting.

[1] http://www.lakeside-labs.com
[2] http://www.eye-tech.it
[3] http://www.infofactory.it

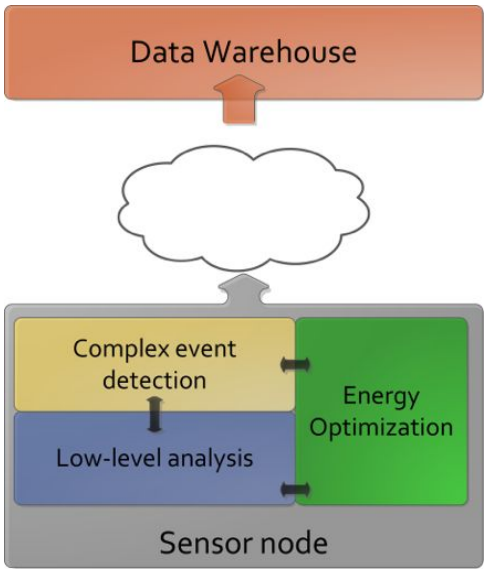


Figure 1.1: The project parts of SRSnet and their relations. Low-level analysis, complex event detection and energy optimization are executed directly on the platform. Relevant processing results are sent to the data warehouse.

To achieve this, components for sensor data processing, event detection, storage as well as algorithms for resource-aware reconfiguration must be employed.

In SRSnet this done by combining the low-level features extracted from sensor data into high-level events by using a complex event detection engine. Low- and high-level features and events are fed into a data warehouse which presents results to end-users. Sensor data processing and complex event detection are hosted at the sensor platform itself. Additionally, the component for resource-aware reconfiguration determines the parameters for the sensor data processing part during the runtime and thus, it is also hosted on the sensor node. The data warehouse is realized as a web-based service. Figure 1.1 shows how the components of SRSnet integrate.

During the three years project time of SRSnet new techniques and methods for operating a smart camera network in infrastructure-limited environments have been developed. In the first year, several test data recordings have been performed in the National Park "Hohe Tauern" as well as in the Lakeside Park in Klagenfurt. Based on the collected test data, techniques for computer vision on limited-performance hardware as well as a fast method to perform complex event detection have been developed. During the second year, a first test deployment in the National Park was done. In the final year, improvements have been applied and a final deployment was undertaken to show the final network performance.

The work in the SRSnet project is in part covered in this thesis. The

development of algorithms for the energy optimization module led to the development of the resource-aware reconfiguration algorithms presented in section 4. In addition, the energy-efficient, high-performance wireless VSN node which is presented as the PandaCam later in this thesis (section 6) was built in this project. Further, SRSnet sensor network, which contains all project parts connected by a common software framework was used in many experiments presented in section 6.

1.2 Contributions to the State of the Art

This thesis explores the problem space of finding or approximating resource-optimal configurations for VSNs. This requires finding an optimal assignment of tasks to camera nodes in a VSN with the goal of a reduced resource consumption while still providing sufficient sensing and processing performance to achieve meaningful application results. It also includes finding optimal operating parameters for image processing tasks on smart cameras.

The main contributions of this thesis include:

Formalization of the problem. A formal description of the problem space and of the optimization goals has been developed and is used as basis for the development and evaluation of algorithms[19].

Exploration of the algorithmic design space. In course of this thesis several algorithms for different areas of the problem space have been developed[76]. This includes

- A centralized algorithm for environments with no dynamic changes[19].
- A distributed algorithm for low-dynamic environments where changes occur infrequently[21].
- A combined, distributed reconfiguration and handover algorithm for highly dynamic environments[20].

Implementation, Demonstration and Evaluation. The algorithms have been tested intensively in real and simulated environments. In a test deployment the realizability of a resource-aware VSN powered by autarkic energy sources has been shown[5, 4].

Generalization. Beginning with VSNs, the algorithms and models have been applied in a different application area. In a cloud computing infrastructure [18], the central algorithm is applied for server task assignment.

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

Section 2 covers important related work. Starting from single smart cameras, the development of VSNs is described. In addition, more focus is put on resource limitation in smart cameras and smart camera networks.

In section 3, a formal definition of the reconfiguration problem is given. We introduce a formal description for coverage and resource consumption and the corresponding optimization goals for the algorithms presented later.

Section 4 describes the exploration of the algorithmic design space. Three algorithms for different environmental dynamics are presented. A centralized, evolutionary algorithm for environments with no dynamic changes is presented first. It is suitable for a-priori calculation of configurations or in settings, where it is feasible to collect all necessary input data in one central node. For applications, where surveillance requirements change slowly over time, a distributed algorithm is presented next. It exchanges messages between nodes to negotiate the best solution. Based on this algorithm, we show a third approach for applications where requirements for surveillance change fast. As an example we use object tracking, where cameras need to react fast in order to not lose track of an object

In section 5, software tools to support various aspects of reconfiguration are presented. We present a software framework which helps in development of evolutionary algorithms, a software framework for dataflow processing and a middleware system which helps in the development of distributed (VSN) applications.

We evaluate the presented algorithms in section 6. We show evaluation results from realworld and simulated scenarios for our algorithms.

A generalization of the resource-aware reconfiguration is described in section 7. We show the application of the central reconfiguration algorithm in a cloud-computing environment. The servers used as backend in a mobile applications are load-balanced according to the assignment calculated by this algorithm.

Finally, section 8 concludes the thesis and gives an outlook to related future research directions.

Chapter 2

Related Work

In this chapter, relevant related work to the field of smart camera systems and applications is discussed. First, smart cameras and their applications are described. Then, applications of VSNs and middleware systems to support VSN operation are presented.

2.1 From Smart Cameras to Visual Sensor Networks

Unlike traditional cameras, smart cameras have the ability to perform in-situ processing of the captured image and video data. Smart cameras are embedded devices consisting of a video sensor, a processing unit with attached memory and communication facilities. As the sensor captures new image data, it is directly processed in the processing unit. After that, a high-level description can be produced and disseminated using the communication facilities.

Typical tasks performed on a smart camera are the capturing of images, image registration, object detection, object tracking and reasoning about the object class, intentions and actions of tracked objects. Depending on the application scope, different algorithms have been proposed to perform all these tasks.

Smart cameras have a wide range of applications. They are used for tasks like detecting motion in simple security applications or gestures for human computer interaction [101, 85]. A more complex applications is the surveillance of crowded public spaces like train and subway stations or airports [38, 39, 74]. The goal is to detect suspicious behaviors and threats to public safeties like lost luggage (as depicted in figure 2.1).

Smart cameras are also well suited for monitoring traffic [10]. They can be deployed at a fixed location and report unusual events like car accidents and congestion[68].

In elderly care, smart cameras can be used as an unobtrusive way to ensure the welfare of inhabitants by automatically detecting e.g. falls[98, 31].

Depending on the application goals, different forms of image processing may be performed. However, in any case, the chosen procedures must be performed on a typically resource-limited system. Thus, careful evaluation of the utilized image processing algorithms is crucial.

(a) The captured image data

(b) The objects detected in the video are one piece of luggage and a person moving away from it

Figure 2.1: The basic workflow of a smart surveillance sytem for left luggage detection. The image data is captured, objects are detected and classified. After that an alert may be issued to an operator. *Still images were taken from the i-Lids dataset for AVSS 2007*

2.1.1 Smart Camera Platforms

Several hardware platforms for smart cameras have been proposed in recent years. They differ in form factor, computational performance and power consumption. Simple image processing algorithms may be performed on devices which provide only very limited processing power but in return use only little energy during operation.

Already in 1999, Moorhead and Binnie presented an approach towards a smart camera [60]. An edge detection algorithm chip is combined with a CMOS sensor. This yields high processing performance at low energy demand.

In 2002, Albani et al. [6] presented a system on chip (SoC) design for computer vision applications called VISoc. The chip includes not only a visual sensor but also a RISC-processor and a vision co-processor as well as several supporting components.

Cyclops [72] is a low-power wireless smart camera node. Its low power consumption is ensured by various techniques such as on-demand access to hardware resources like external SRAM or switching unused hardware sub-modules to low-power states. Further, it integrates components for parallel processing and extendability. A Cyclops node is typically attached to a host processor which is then responsible for data processing.

The CMUcam3 platform [82] consists of an image sensor, a frame buffer and a 60 MHz microcontroller. It provides sufficient processing power for performing simple tasks like segmentation and consumes only approximately

0.5 Watts in full operation. The image sensor delivers images of 352×288 pixels at 50 frames per second.

The Fleck platform [89] is a modular system consisting of a base board which can be extended with expansion boards like GPS or camera. Besides that, Fleck also features a real-time clock, a radio module and a trusted platform module (TPM) for security-related applications. Data processing is performed either on the built-in Atmega128 microcontroller or on an optional DSP expansion board. Flecks power consumption depends on the modules connected to it.

MeshEye [42] is a low-power stereoscopic smart camera platform for smart video surveillance. It combines a VGA camera sensor with two kilopixel imagers. The kilopixel imagers are used to perform simple object detection and their stereoscopic setup is used to determine parameters like distance and size. The corresponding image from the VGA sensor can then be used to perform high-level vision analysis. In this way, a more energy efficient approach to stereoscopic image processing can be achieved since the kilopixel images are sigificantly smaller than standard-resolution sensor data. Computations on the MeshEye platform are performed on a 50 MHz microcontroller.

Citric [14] is a platform providing a higher amount of processing power. A processor running at a maximum frequency of 624 MHz and a large RAM of 64 MB make Citric a suitable platform for more demanding computer vision tasks. It is further equipped with an IEEE 802.15.4 wireless radio enabling communication among nodes and to a backend. Despite its high-performance specifications, it consumes in average approximately 1 W during normal operation which makes it suitable for battery-powered operation.

Microsoft Kinect[1] is a special platform which has gained a lot of interest in recent years is [90, 107]. It is a self-contained vision sensor platform able to not only capture images but also provide 3D-information on the captured scene. It uses two cameras (one RGB, one infrared) in a stereo setup to gain depth information in addition to the captured image data. Contrary to platforms with similar capabilities which have been available for quite some time, the Kinect is available at a very low price making it suitable for large-scale deployment. Already, the research community has adapted the Kinect platform in various fields like robotics [27, 95] or care applications [15, 91, 62, 66].

Special Purpose Smart Cameras

In some cases, the architecture of a smart camera platform can be specially tailored to meet a certain application purpose. Depending on the intended use, different hardware components and setups can be chosen.

[1] `http://www.xbox.com/Kinect` and `http://www.kinectforwindows.org`

Kleinhorst et al. [48] present the INCA$^+$ platform, a smart camera especially aimed at face recognition applications. They divide the tasks involved in face recognition into two categories:

Low-level operations on pixel level which can be performed in a massive data-parallel fashion, like grayscaling or brightness correction

High-level tasks where parallelism is required at the task level, e.g. edge detection or scanning a database for matching faces

INCA$^+$ is a multi-processor platform consisting of a massively parallel SIMD-enabled (Single Instruction Multiple Data) processor for low-level tasks and a DSP for high-level tasks connected in series. Thus, captured images are first processed by the SIMD processor and the results are then passed to the DSP for further processing. The system can detect faces at a rate of four frames per second, and recognize faces at three frames per second.

TRICam [7] is an embedded platform for multimedia surveillance applications in remote locations under harsh conditions. It features a powerful DSPs for image processing, a FPGA for buffering frames between video sensor and the DSP as well as several communication interfaces. If necessary, it can be expanded to use up to three DSPs.

GestureCam [86] is a smart camera for hand gesture recognition. A FPGA is used to perform the data-intensive video processing required to detect hand gestures.

In [8] the authors describe an embedded platform for traffic surveillance. While typical smart cameras are equipped with either a CCD or a CMOS sensor, this platform uses a temporal contrast vision system. Instead of constantly producing image data (like traditional sensors), here each pixel of the sensor is sensitive to illumination changes and only transmits data, if a change in contrast (i.e. some kind of change or motion) is detected. This drastically reduces the amount of data to be processed. Further processing is performed on a DSP.

TrustCAM [99] is a smart camera platform especially designed for privacy-aware surveillance application. Processing is performed on a 480 MHz ARM Cortex A8 and a DSP. In addition to that, it contains a trusted platform module which is used to ensure integrity, authenticity and confidentiality of all captured image data. It is further equipped with a 802.11 WLAN radio and an XBee module for low-power low-bandwith communication.

Comparison of Smart Camera Platforms

Table 2.1 shows a comparison of smart camera systems. Computing power and energy consumption are rated from ○- low via ◔- medium to ● - high or ?- unknown if not specified. The table shows the wide range of

	Computing power	Networking	Energy	Additional
Moorhead-Binnie [60]	○	—	○	
VISoc [6]	○	—	○	
Cyclops [72]	○	on host	○	
CMUcam3 [82]	○	—	●	
Fleck [89]	μC:○ DSP: ●	ISM 433, 868, 915 MHz	●	realtime clock, TPM, expansion slots
MeshEye [42]	○	802.15.4	●	stereoscopic
Citric [14]	●	802.15.4	●	power management IC
INCA$^+$[48]	●	—	?	
TRICam [7]	●	Ethernet	?	up to three DSPs
GestureCam [86]	●	—	?	FPGA-based
Bauer et. al [8]	●	—	●	
TrustCAM [99]	●	802.11, XBee	●	

Table 2.1: Comparison of different smart camera platforms.

power and performance properties of various models ranging from models for simple tasks which feature low energy consumption and low processing power to high performance cameras for complex tasks which require more energy during the operation. In addition, the provided networking facilities differ from low-power low-bandwith 802.15.4 to wireless LAN. In addition, different processing units and application-specific extra components can be found.

2.1.2 Visual Sensor Networks: Distributed Smart Cameras

A single smart camera definitely is an enhancement over a simple video camera. However, there are problems that cannot be solved with just a

single smart camera. Occlusions in a scene are one example of problematic situations. If objects are hidden by other objects or the environment, a single camera might deliver unreliable or faulty results. Other examples are surveillance of wider areas or buildings. Many problems can be overcome by using multiple cameras and fusing the data of multiple cameras in order to achieve a global result. This is called a Visual Sensor Network (VSN) [53] or a network of distributed smart cameras [79]. Typical properties of a VSN are *i)* distributed (in-network) sensing and processing on *ii)* embedded devices with typically *iii)* low-power wireless networking technologies as communication media. Figure 2.2 shows the research field of distributed smart cameras is positioned within related fields.

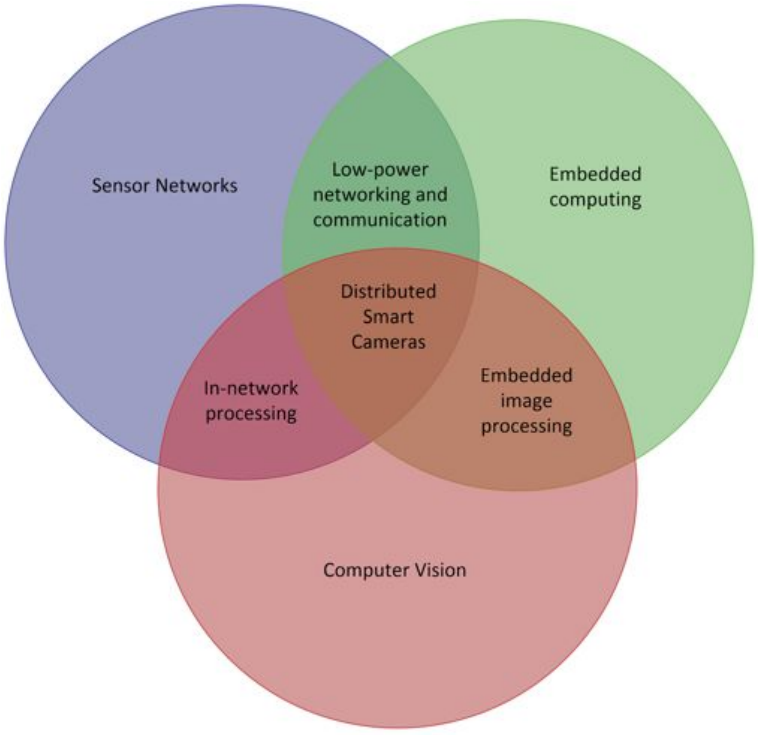


Figure 2.2: Fields contributing to distributed smart cameras

A VSN is a distributed system of smart cameras where each camera processes the captured data locally and exchanges information with other cameras in the network to reach the global goal. Many different applications of distributed smart cameras have been proposed so far [93, 1, 97].

ASAP [88] is a camera network to enable situation awareness of computing systems. It is able to survey a certain area, process the collected data and trigger certain actions based on detected events (*sense-process-actuate*). As an example, ASAP senses video data, processes it to detect and identify faces and triggers an alert if a person has entered a prohibited area. ASAP

implements an agent-based approach to abstract sensing and high-level processing. ASAP is able to incorporate heterogeneous sensors like cameras, RFID readers and fire detectors. The results of the sensor fusion can later be queried using an SQL-like query language.

In [94] a VSN for distributed object tracking is presented. A distributed Kalman consensus algorithm is used to establish a common model of the ground plane on all cameras. This knowledge is later used to perform object handoff between cameras.

SensEye [49] is a VSN aimed at improved latency and energy-efficiency. It is composed of multiple layers where each layer has different capabilities and features in terms of sensing and processing capability and energy demand. A lightweight and energy-efficient layer of CMU-cams (see section 2.1.1) attached to simple Motes, the second layer consists of Stargate boards which are more powerful, the third layer features PTZ-cameras connected to embedded PCs. The lowest layer performs simple object detection. By default, the higher and more powerful layers are in sleep mode. Whenever relevant detections are made in the lowest layer, higher layers are activated to perform more demanding tasks. By always choosing the lowest possible layer to perform a specific task, SensEye is able to be very energy efficient and yet responsive to important events.

DMCTRAC [43] is a distributed smart camera network consisting of PTZ-cameras. These cameras need additional control to ensure that cameras are always oriented optimally to fulfill a certain task. To achieve this, a master/slave approach is taken where one camera has the tracking responsibility for a certain object and is thus adjusting its PTZ parameters in order to follow it while it moves. The other cameras are acting as slaves by providing additional information and searching for new objects.

Fleck et al. [31] present an application of smart cameras in the area of assisted living. Here, especially the assistance of elderly persons is in focus of the project. A network of smart cameras which perform person tracking and activity recognition is employed. The collected data is sent to a server. Multiple visualizations use the data of the server to show the recognized events as a virtual 3D reconstruction[32], in a 2D overview map or as the raw video feed.

Kwangsu and Medioni [47] use a VSN of static and mobile (robotic) cameras in scenarios where robots for human interaction need additional information about the surroundings. The static camera nodes perform person tracking and feeds this information to mobile robots which use stereo cameras to detect command gestures of users.

2.2 Resource-limited Camera Networks

In many multi-camera networks [37], the available resources are limited. Especially in the field of *distributed smart cameras* [79] and *VSNs* [1, 93], resource limitations must be considered during system design. While data in traditional camera networks is processed at a central node, in VSNs the processing of the visual data is distributed among the individual camera nodes. This leads to minimized communication effort since no raw data must be transferred. Thus, the communication infrastructure can rely on channels with lower bandwidth which consume less energy [80].

Recently, research interest has focused on resource assignment in camera networks. First, computing platforms and sensors are one such field. As described in section 2.1.1, resource efficient camera platforms have already been proposed.

Second, a software-side approach to dynamically manage resources can be employed. Suitable functional configuration for nodes and network which minimize resource consumption during operation have to be found [93].

Maier et al. [52] describe dynamic power management method for camera networks called PoQoS. Individual camera components change their power modes based on the current performance requirements of the network. For this an genetic algorithm for multi-objective optimization is used. It presents multiple solutions for the optimization dimensions 'maximizing service quality' and 'minimizing power consumption'. A dedicated management component then picks one of the solutions and reconfigures the cameras appropriately.

In Winkler et al. [100] a multi-radio approach is proposed. The system dynamically switches between low and high power radios depending on the current requirements of the application. During normal operation, a low-power channel is used to transmit management commands. Whenever larger amounts of data need to be transferred, a high-power radio is activated.

Karuppiah et al. [46] present a hierarchical approach to resource management and reconfiguration of distributed smart cameras. In this system, fault containment units (FCU) rely on a redundant set of resources which are provided as services. The local redundancy within the FCU allows to compensate faults at the local level. If this fails, the FCU can propagate the error up the hierarchy where it might be compensated by another FCU. Within the FCU hierarchy, resources are allocated dynamically to adapt to different bandwidth requirements (e.g. raw data vs. compressed images).

Durmus et al. [26] propose the "event based fairness scheme" for fair resource management in VSNs. It allocates resources in the network according to the event category and priority to ensure fair routing of events but to also prioritize more important events. This is achieved by adapting packet queues in routing nodes and the contention window.

Shiang et al. [87] evaluate different strategies for the allocation of wireless

networking resources. Three algorithms, a centralized, a game-theoretic and a greedy approach, are proposed and an evaluation on which algorithm is best for which setup is performed.

Zou et al. [109] reconfigure routing, network flow control and video encoding to optimize for network lifetime. Based on a model of power consumption for network coding, a distributed algorithm to reconfigure the network parameters is developed.

Casares et al. [11] try to maximize processor idle time on smart cameras in order to prolong the node lifetime. Depending on scene content, the method can selectively send camera nodes to sleep to save energy.

Yu et al. [103] propose a camera selection and energy allocation strategy for battery powered camera networks. The goal is to deliver a user-requested view in a VSN. The method selects cameras and allocates resources using a stochastic model of the network lifetime.

He et al. [41] characterize relationship between power consumption and rate-distortion performance of a video encoder. Based on this the authors investigate on the optimal power allocation between encoding and wireless transmission, i.e. when increasing compression to reduce the amount of data is cheaper than transmitting more data and compressing less.

Chen et al. [13] propose an adaptive resource management mechanism for camera handoff. The method incorporates the number of currently tracked objects on a node and the resources needed to track an additional one. By prioritizing objects in the scene, it is possible to achieve a defined minimum frame rate while tracking.

In [61] the authors propose a camera control scheme using additional sensors to selectively turn on and off cameras to save energy. In this case, audio sensors which use less energy than cameras are used to detect interesting events and only then, cameras are activated to perform video surveillance.

Monari and Kroschel [59] present a framework for single-object multi-camera tracking which performs sensor selection and task migration. The framework dynamically instantiates a new distributed object tracking process for each new object which is then tracked over multiple cameras. Therefore, each instance only focuses on one single object and is not required to incorporate other properties of the scenes.

2.3 Sensor Placement and Selection

A fundamental question in building sensor networks is where to place the sensors. During operation, selecting the right subset of sensors for a specific task is another crucial aspect to providing optimal coverage at low cost[50]. The area of interest is often described by a set of critical sites (referred to as control points), and each control point has to be covered by at least k sensor nodes. Optimal node placement has been shown to be a NP-hard problem

[102]. To tackle such complexity, heuristic approaches to approximate an optimal solution have been proposed (e.g., [65]).

Camera networks having directional sensors pose additional complexity in finding an optimal placement [64, 92]. Similar to the omnidirectional case [12], often integer linear programming (ILP) models are used to represent the problem [63]. Several heuristics to approximate have been proposed [34, 40]. These approaches are based on different constraints regarding the sensors (homogeneous vs. heterogeneous; fixed vs. mobile), assumptions on the environment (static vs. dynamic), sensor coverage models and the optimization goals. For instance, simple 2d representations like segments or trapezoids are often used to model the covereage are of a sensor (e.g., [44] [64] and [40]).

Coverage in Pan Tilt Zoom (PTZ) camera networks changes during operation due to the change in the PTZ parameters. This can be exploited to achieve the maximum useful coverage of an area [55]. Changes like moving control points can be adapted to and thus e.g. continuous tracking coverage can be ensured. Here, it is especially important to employ suitable control mechanism for accurate control of the cameras. This requires sophisticated coverage models [57, 46]. Several PTZ reconfiguration algorithms have been presented. Piciarelli et al. [67] use activity maps, where areas of high activity are highlighted, to reconfigure PTZ parameters in order to cover the most active areas. Soto et al. [94] use cooperative network control based on multi-player game learning to continuously track objects in a PTZ network at a desired resolution. In [71] the authors present a tool to simulate a camera network using virtual reality. Cameras can be freely placed and moved. This can then be used to develop and improve methods for camera reconfiguration.

Placement, coverage and routing in VSNs have often been identified as being multi-objective optimization problems [81]. Rajagopalan et al. [73] employ evolutionary algorithms to optimize placement coverage, routing and aggregation for Wireless Sensor Networks (WSNs).

2.4 Middleware for Distributed Smart Cameras

Middleware systems take over typical tasks which are common in many distributed computing systems like communication or modularization. They help in building scalable and fault tolerant distributed applications.

Middleware systems specially designed for VSNs aim at providing support for developers of VSN applications in speeding up and facilitating development [77, 58]. Cougar [9] or TinyDB [51] are data centric middleware approaches. Middleton et al. [56] propose an agent-based middleware for large camera arrays that provide intelligent access to sensors and in-network processing. Other approaches using agent-oriented middlewares are Agilla

[33] DSCAgents [69] and In-Motes [35]. A CORBA implementation for smart camera systems is presented in [105]. [106] presents an embedded middleware for person tracking where computationally expensive parts are hardware accelerated using a novel system on chip design. These middleware systems focus on reliable services for ad-hoc networks and energy awareness [104]. The spectrum ranges from a virtual machine on top of TinyOS, hiding platform and operating system details, to more data-centric middleware approaches for data aggregation (i.e., shared tuplespace) and data query. Agilla [33] and In-Motes [35], for example, use an agent-oriented approach. Agents are used to implement the application logic in a modular and extensible way and agents can migrate from one node to another. Cougar [9] or TinyDB [51] follow the data-centric approach, integrating all nodes of the sensor network into a virtual database system where the data is stored distributed among several nodes.

[106] presents an embedded middleware for person tracking where computationally expensive parts are hardware accelerated using a novel system on chip design.

Detmold et. al [17] present a multi-layered middleware for distributed video surveillance. The signal processing layer at the bottom is responsible for collecting raw video streams and managing the video processing. A services layer offers facilities for topology estimation, camera management and streaming. The topmost layer is concerned with interpretation of collected video data. This middleware however, is not suitable for embedded operation since it builds on top of services like HTTP transport which cause a major overhead. Thus it builds upon an infrastructure of backend servers which handle CPU-intensive processing tasks.

In [25] the authors present a publisher-subscriber middleware for DSP-based smart cameras. It can be used for dynamic reconfiguration, i.e. for changing the tasks running in the system. Publishers provide data which is then consumed by subscribers. Both can reside either on the same DSP or on a remote processor connected via PCI.

This framework is used in [24] to perform power-aware reconfiguration on DSP-based smart cameras. Reconfiguration here means either changing the QoS-level (e.g. framerate) or locality (processor, node) of a task or changing the power levels of hardware components. Since QoS and power demands are contrary dimensions for optimization, an evolutionary algorithm for multi-dimensional optimization is employed to solve the problem.

In [45] Jovanovic et al. use the previous two approaches for dynamically changing the configuration of a VSN. They focus on exchanging and changing software tasks running on the respective cameras. This enables higher flexibility, robustness and resource utilization in a VSN.

[75] presents an agent-oriented approach to middleware for DSP-based smart cameras. Here, autonomous agents are used to transfer processing functionality between nodes. In a multi-camera tracking case study, agents

hop from one node to another in order to continuously track an object moving through the surveyed area. The authors present an evolved version of the middleware with improved performance characteristics in [70]. This middleware system is then further developed and presented in [69].

A middleware for resource-aware data management using a distributed event space is proposed in [83].

A publish-subscribe middleware for wireless sensor networks has been recently presented in [96]. A special feature of this system is that it is able to connect clusters of a sparse WSN by piggybacking data on mobile nodes (e.g. mobile phones) moving between clusters. However, its HTTP-based communication causes some overhead in the communication channel.

2.5 Differences to the State of the Art

Although the presented related work has some similarities to the approach presented in this thesis, there are significant differences.

First, we consider the coverage and task assignment problem as finding an optimal *VSN configuration* where the in-networking processing (i.e., task assignment) can be changed as well. This requires the algorithms to adapt to changes in the requirements at runtime. Thus, our algorithms can process a non-static task assignment.

Second, resource consumption and sensor coverage are modeled corresponding to the different requirements of the camera nodes. In this thesis models of two different hardware platforms are shown. They can be combined in one network. The presented algorithms then choose the models according to the platform of each sensor node individually. Resource consumption and coverage quality are modeled for different algorithms of the same class (e.g. multiple algorithms which perform background subtraction). This enables an algorithm to choose a certain algorithm according to resource and quality requirements.

Finally, efficient approximation algorithms are presented for environment with different dynamic characteristics. We demonstrate algorithms which are designed for different levels of dynamics in the environment. A centralized evolutionary algorithm for scenarios where the the task assignment does not change is presented along with two distributed algorithms which deal with infrequent and very frequent changes in the task assignment.

Chapter 3

Problem Definition

This section is adapted from [19, 20, 21]. In this chapter we define the VSN reconfiguration problem in more detail. After an overview of the challenges, we develop a formal description of the problem and define the optimization goals.

3.1 Overview and Assumptions

As shown in section 2, resource awareness is a booming research topic in the field of VSNs. Managing the resources available on embedded camera platforms is key to its efficient operation. Cameras with more tasks assigned than they can handle will not deliver the expected results and performance while cameras with a lot of free resources decrease the network efficiency. In networks with both limited computing resources and limited energy reserve, a resource management mechanism can extend the network lifetime.

In this thesis, we concentrate on resource limited VSNs with various requirements for resource management. First, we consider networks where a set of heterogeneous tasks must be performed. This means that not all cameras perform the same processing on the collected data but rather that e.g. one camera may be tracking objects while another one may perform face recognition or motion detection. Tasks typically have constraints in terms of location and quality, i.e, one task needs to be performed at a certain part of the area of interest (e.g. face detection at gates) and with a certain quality. A smart resource management mechanism needs to choose the camera to perform a certain tasks considering the camera's resource state, the quality and the location requirements of the task to perform.

Second, we consider networks where some tasks require collaboration between cameras. A typical use case is distributed object tracking, where at each point in time, one camera is responsible for tracking a specific object. Whenever this object is about to leave the camera's FOV, it must be handed over to another camera to guarantee continuous tracking. This use case poses a challenge not only in selecting a suitable camera for handover but also considering the resource constraints of each camera involved. It is not possible to hand over an object to a camera which is already at its resource limits. Thus, the resource management must be tightly integrated into the handover mechanism to guarantee flawless, resource-aware tracking performance.

Finding an optimal network configuration while at the same time sticking to various functional and resource requirements is fundamental in this area. In contrast to typical optimization problems, we focus here on a combined *sensor selection and resource allocation problem*. Objectives are (i) selecting a subset of cameras nodes in the network which provide sufficient coverage to monitor the area of interest, (ii) selecting the sensors' frame rate and resolution accordingly and (iii) assigning the required monitoring and data processing procedures to the camera nodes. This *configuration* of the camera network must satisfy both the resource requirements on the camera nodes in the network as well as the monitoring constraints of all observation points. Finally, we aim at constructing a generic framework for task assignment and resource allocation to reconfigure camera networks in various scenarios with different levels of activity.

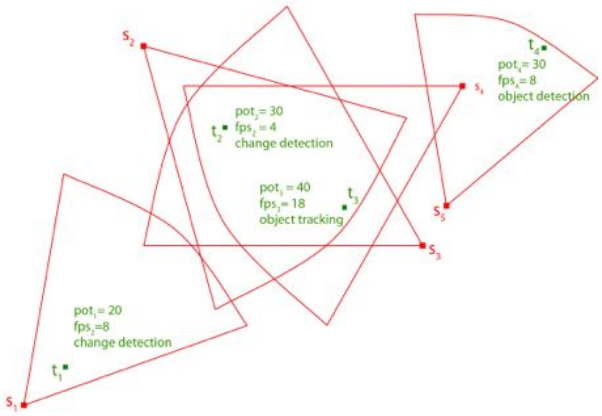
A sample of the considered configuration problem is depicted in figure 3.1(a). A set of n camera sensors S is placed on a 2D space. We represent the coverage area of each camera with a segment. The set $T = t_1, \ldots, t_m$ contains all observation points in the area. Each $t_j \in T$ has to be covered by at least one camera at a given QoS. Frame rate (fps) and pixels on target (pot) (i.e. the pixel resolution at the observation point) determine the QoS delivered for one observation point. The covering camera (s_i covering t_j) has to deliver the monitoring activity a_{t_j} of the observation point, i.e., a set of image processing procedures $\tilde{P}_{s_i}$ must be executed at s_i while not exceeding the available resources (processing, memory and energy) of the camera.

A potential solution to the example configuration problem depicted in figure 3.1(b). Cameras s_1, s_3 and s_5 are selected. t_1 is covered by s_1, thus this camera is configured to achieve at least 8 fps and 20 pot. It performs change detection. t_2 and t_3 are covered by s_3; thus, s_3 must achieve 18 fps and at least 30 pot at t_2 and 40 pot at t_3. s_3 performs change detection and object tracking procedures. Finally, s_5 covers t_4; this camera must be configured to achieve at least 8 fps and 30 pot and execute object detection procedures.

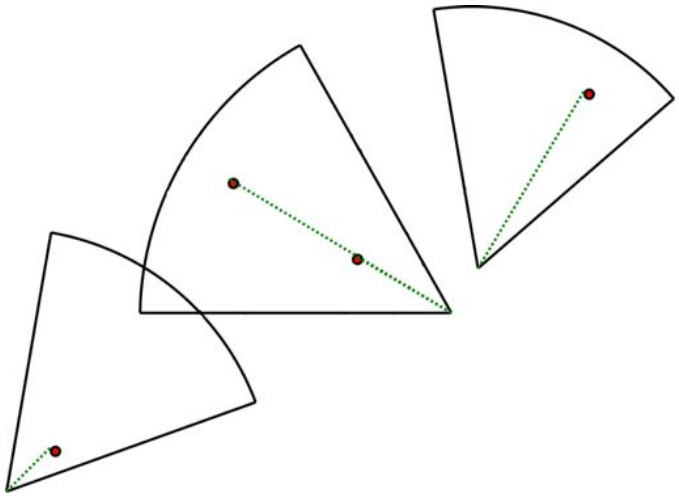
For modeling the network configuration problem we make the following assumptions[1]:

- The camera network consists of directional sensors with a fixed position and fixed field of view (FOV). The frame rate and the resolution of the image sensor can be changed within an a-priori known set of sensor configurations.
- Each camera is able to capture images (at the defined resolution and frame rate), to execute a sequence of image processing procedures and to transfer data/results to other camera nodes in the

[1] In [19] we describe an extension of our network configuration approach to optimize PTZ camera configurations. Naturally, we are able to relax some of these assumptions for this extension, i.e., fixed FOV and 2D space modeling.



(a) Example



(b) Solution

Figure 3.1: A simple example for a sensor selection and resource allocation problem. Five cameras $s_1, \ldots, s_5$ with fixed FOV (red segments) are in place. Four observation points $t_1, \ldots, t_4$ are to be covered by cameras. A suitable network configuration must be found, i.e., the (sub)set cameras required to cover all observation points as well as the assignment of processing parameters (fps, pot) and all necessary image processing procedures to the cameras in this set. Quality requirements and resource optimization criteria must be met. Potential optimization criteria include minimizing global energy usage or maximizing overall network lifetime.

network. This data transfer is realized in a simple peer-to-peer manner. Complete communication coverage among the nodes and a potential base station is assumed.

- The observation points are static locations in the monitoring area which must be covered by at least one camera's FOV at sufficient resolution, i.e., pixels on target. The pixels on target are determined by the sensor resolution and the distance between camera and observation point.
- We currently only consider convex 2D space without obstacles restricting the camera's FOV.

3.2 Problem Formulation

We consider a set

$$S = \{s_1, \ldots, s_n\}$$

of n camera sensors.

For each sensor s_i its

- geographical position (x_{s_i}, y_{s_i})
- available resources $r_{s_i} = (c_{s_i}, m_{s_i}, e_{s_i})$ describing processing, memory and energy resources
- l possible data input configurations $D_{s_i} = \{d_{s_i 1}, \ldots, d_{s_i l}\}$ where $d_{s_i j}$ is a tuple $(res_{s_i j}, fps_{s_i j})$ representing a certain resolution and the frame rate of the image sensor

are known.

The set $D = \{D_{s_1}, \ldots, D_{s_n}\}$ represents the input configurations for all cameras.

As illustrated in figure 3.2, a camera's field of view is represented by the orientation θ_{s_i}, the covering angle δ_{s_i} and the covering distance ω_{s_i}.

$$T = \{t_1, \ldots, t_m\}$$

is a set of m observation points. For each $t_i \in T$ we know

- its geographical position (x_{t_i}, y_{t_i})
- the required monitoring activity $a_{t_i} \in A$ (where A is the set of all monitoring activities that the sensors are capable of)
- the required QoS expressed as pixels on target pot_{t_i} and frame rate fps_{t_i}

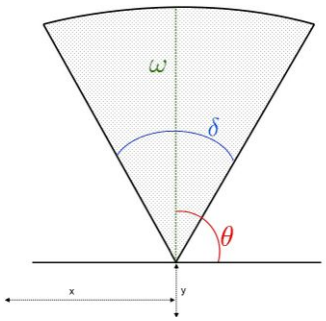


Figure 3.2: The 2D model of the camera's field of view which is defined by the covering angle δ (blue) and the covering distance ω (green) and the orientation θ (red). The location of the sensor are depicted by x and y. The field of view is indicated as gray area.

A high-level monitoring task which must be achieved at an observation point t is represented by activity a_t. Image compression and streaming, change detection, object detection, object tracking and person counting are examples for high-level activities. An activity is realized by executing some image processing procedures at a camera. In general, there may exist several different combinations of single image processing procedures which realize a certain activity. For example, object tracking can be achieved by combining a background subtraction procedure (e.g., mixture of Gaussian or frame differencing), an object detection procedure (e.g., connected components) with a tracking algorithm (e.g., CamShift or KLT tracking). Different combinations of these procedures achieve the desired activity, but naturally impose different resource requirements.

For each $a \in A$ we define the set $P_a = \{p_{a_1}, \ldots, p_{a_p}\}$ representing alternative procedures for achieving a. Thus, each $p_{a_i} \in P$ is a set of procedures $p_{a_{i_1}}, \ldots, p_{a_{i_b}}$. The execution of all these procedures is necessary to achieve a.

A camera is able to execute multiple activities at once. For each activity a_i assigned to that camera an appropriate p_k from P_{a_i} must be selected. The set of sets $\tilde{P}_{s_j}$ contains all p_{a_i} assigned to s_j. If $\tilde{P}_{s_i} = \emptyset$ no image procedure is assigned to s_i, and this camera can be switched off to save resources.

3.2.1 Dynamic Environments

In some environments, surveillance requirements may change at runtime. Observation points may be removed or added, their requirements or location may change or moving objects may pass through the area which have to be tracked continuously.

The properties of some tasks in this network change slowly. The task of performing motion detection may change its quality requirements or location

at some time e.g. based on operator input or activity maps [67]. However, those changes are assumed to occur infrequently.

Other tasks like object tracking require highly dynamic action by the cameras. As the object moves, a camera must determine when to hand over the object to a neighboring camera (assuming a shared field of view) before losing it.

In addition to observation points, we also define *objects to track*. They share the quality requirements of observation points but represent agile objects in the observed area. They have high dynamics, thus the sensor network must react quickly in order to not lose track of objects. If an object approaches the borders of the FOV of a camera, it must be handed over to a neighboring camera. In this handover procedure, the cameras must decide which of them is best suitable for tracking the object, i.e. which is able to deliver the required quality at low resource cost (computational capacity, memory or energy reserves). The set of all p objects to track is defined as $O = \{o_1 \dots o_p\}$.

Figure 3.3 depicts our dynamic reconfiguration problem. A set S of n camera sensors is placed on a 2D space such that they have a partially shared FOV; the coverage area of each camera is represented by a segment. Within the fields of view of the cameras' five observation points (dots) and one moving object (square) are placed.

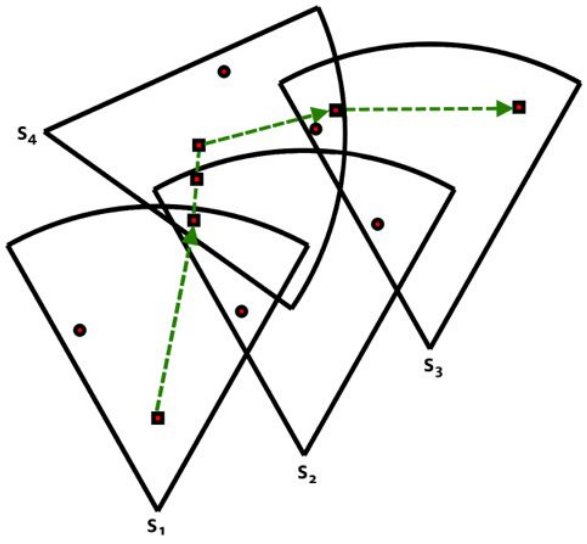


Figure 3.3: A sample scenario with cameras $s_1 - s_4$. FOVs are depicted as segments, five observation points represented as dots and one object with its trajectory.

In this combined *coverage and handover* problem we want to minimize the resource costs inflicted by static tasks (such as background subtraction)

as wells as tasks with dynamic portions (such as tracking). However, resource consumption is not the only optimization criterion. We also include the surveillance quality expressed as minimum pixels on target and framerate for all objects and targets. Hence, we solve a multi-criterion optimization problem in two dimensions where we find the tradeoff between surveillance quality and resource consumption. To be able to perform the reconfiguration at runtime, a distributed approach is required.

3.3 Reconfiguration

To optimize resource consumption, an estimation on resources must be made before deciding on the configuration. A function $\tilde{r}(\tilde{P}_{s_i}, d_{s_i}) \rightarrow (\tilde{c}_{s_i}, \tilde{m}_{s_i}, \tilde{e}_{s_i})$ calculates the required resources. It specifies processing, memory and energy resources for the individual procedures. It depends on a specific data input configuration according to the models described in section 4.1. The required resources are specified on a single frame basis.

Pixels on target for a certain t_i are computed using the function $f(D \times T)$. It is based on our simple 2D geographical model presented in section 4.1.1.

Feasible Configuration

We search for *feasible* configurations of the complete network. This means that all resource requirements, QoS requirements and activity requirements must be satisfied. Thus, for each sensor s_i, the required memory and processing resources of all assigned procedures $\tilde{P}_{s_i}$ must not exceed the available resources. The required resources for the given input data configuration can be computed by $\tilde{r}(\tilde{P}_{s_i}, d_{s_i})$. Thus, the following condition must hold:

$$\forall s \in S : \tilde{c}_s \leq c_s \wedge \tilde{m}_s \leq m_s \tag{3.1}$$

In order to satisfy the QoS requirements, every observation point must be covered by at least one sensor. This point must be within the field of view of the camera. The sensor must be configured to guarantee a certain number of pixels on target:

$$\forall t \in T \exists s_i \wedge \exists d_{s_i} : f_i(d_{s_i}, t) \geq pot_t \wedge fps_d \geq fps_t \tag{3.2}$$

where l represents the number of sensor configurations.

Finally, to satisfy the activity constraints, every observation point must be covered by at least one sensor which must execute the set of image processing procedures that achieve the desired activity for that observation point:

$$\forall t \in T \exists s_j \wedge \exists p \in \tilde{P}_{s_j} : p \in P_{a_t} \tag{3.3}$$

3.3.1 Optimization Criteria

In general, there are multiple feasible configurations possible for a given network configuration problem. Thus, we are interested in configurations which optimize some criteria. This optimization can be performed in multiple optimization criteria and with different optimization objectives. In this thesis, we are focusing on three different criteria: (i) quality, expressed as pixels on target, frame rate and surveillance activity; (ii) energy demand; and (iii) processed data volume. Naturally, different criteria can be defined as well.

Since $\tilde{r}$ calculates the resource usage for processing a single frame, we define the remaining lifetime of a node using the required ($\tilde{e}_{s_i}$) and available energy (e_{s_i}) as well as the frame rate:

$$L_{s_i} = \frac{e_{s_i}}{\tilde{e}_{s_i} \cdot fps_{s_i}}$$

In terms of energy usage, the optimization can follow different criteria. Examples criteria include:

- Minimum global energy usage:

$$\min \left(\sum_{i=1}^{n} \tilde{e}_{s_i} \cdot fps_{s_i} \right) \tag{3.4}$$

- Maximum lifetime for a specific node s_i:

$$\max \left(L_{s_i} \right) \tag{3.5}$$

- Maximum overall network lifetime:

$$\max \left(\min \left(L_{s_i} \right) \right) \tag{3.6}$$

Considering the data volume processed on a node we can minimize the data volume with respect to resolution and frame rate.

$$min \left(\sum_{i=1}^{n} res_{s_i} \cdot fps_{s_i} \right) \tag{3.7}$$

To express surveillance quality at the task level, we assign a quality rating to the processing procedures using the function $q(\tilde{P})$. This function then maps a set of image processing procedures to a quality ranking. By accumulating all quality values, we achieve a global quality measure for our surveillance tasks.

$$max \left(\sum_{i=1}^{n} q(\tilde{P}_{s_i}) \right) \tag{3.8}$$

Building on top of the problem and goal formulations presented above, we can explore the design space defined by the formulations. This includes optimizing in one or more dimensions like resource demand and quality and in multiple levels of environmental dynamics where observation points change and objects must be handed over. In the next chapter, we look at the approach taken to perform this design space exploration.

Chapter 4

Resource-Aware Reconfiguration

VSNs for different application purposes and in different environments show different dynamics in reconfiguration. In a VSN with no dynamics, the tasks to be fulfilled do not or rarely change at all. Thus, it is typically sufficient to calculate a configuration a priori and start the network operation on predefined parameters. Whenever changes occur more frequently at runtime, a distributed algorithm is a smart solution. Since a central algorithm needs all resource and task information in one place, all nodes would need to regularily transmit the corresponding data to the central node. This is an expensive operation. Further, the central node is a single point of failure and thus, no reconfiguration can be performed once it has failed. In a distributed algorithm, all nodes participate in calculating a solution. In our design we try to minimize the communication effort needed to give nodes sufficient information to calculate the solution. This removes the need for a central node calculating a new configuration, thus removing communication effort and increasing the algorithm efficiency.

In this thesis we explore resource-aware reconfiguration in environments with different dynamics. For environments with no changes at runtime we present an Evolutionary algorithm (EA) which is a centralized solution used for a-priori configuration calculation. For applications where observation points change at runtime, a distributed reconfiguration algorithm is described. In VSNs where moving objects are handed over between cameras and where thus a high dynamic can be observed, a hybrid observation point-based reconfiguration and object handover algorithm is developed.

All algorithms use the same models to calculate coverage and estimate resource consumption. Those predictions are used as basis to decide on task allocation and algorithm parameters.

The remainder of this chapter is organized as follows. In section 4.1 we first see how models for coverage and resources are constructed. Section 4.2 describes the centralized EA, section 4.3 describes the distributed algorithm for low dynamic environments. Finally, section 4.4 shows the hybrid reconfiguration and handover algorithm.

4.1 Resource- and Coverage-Models

To perform an optimization of coverage and task assignment, a target goal has to be defined. In case of resource-aware task assignment, the resources must be an optimization goal along with the desired surveillance quality. To assess the quality of a configuration, a prediction on the surveillance quality and resource consumption of this configuration has to be made. This requires models of both of these optimization dimensions.

4.1.1 Coverage Model

The coverage model is used to predict how good an object is covered by a certain camera. It incorporates the Field of view (FOV) of the camera as well as the distance to the object, the resolution of the camera and the surveillance task it is performing. Besides the basic visibility of the object (i.e. if it is in the FOV of the camera), quality is expressed by the pixels on target, i.e. the number of pixels in the camera image which show the objects, and the activity performed on the camera.

To compute the pixels on target for a certain target t_i we use the function $f(D \times T)$ which is based on our simple 2D geographical model. The pixels on target can be calculated by using the angular size of a unit sized object at the given distance $\text{dist}_{i,j} = \sqrt{(x_{s_i} - x_{t_j})^2 + (y_{s_i} - y_{t_j})^2}$ between camera s_i and observation point t_j.

By taking into account the camera's resolution res_i and the camera's covering angle δ_i, we can estimate the 1D resolution at the target, i.e., the pixels on target of a unit sized object of $1m$. This simple estimation is based on the ratio between the angular size of the target within the camera's FOV (which can be approximated by $\frac{360}{2\pi \cdot dist_{i,j}}$) and the covering angle δ_i.

We define
o as the objects real 1D size
r as the objects angular size within the cameras FOV and
d as the distance $dist_{i,j}$.

$$\frac{d}{o} = \frac{360^\circ}{2\pi r} \tag{4.1}$$

If we assume that the size of an object o is 1, this leads to Equation 4.2.

$$d = \frac{360^\circ}{2\pi r} \Rightarrow r = \frac{\frac{360}{2\pi}}{d} \tag{4.2}$$

Assuming that the image in a camera corresponds to a segment of a circle (which's size is δ, the covering angle), we can say that the (horizontal) resolution of the image res corresponds to full δ. To get the amount of pixels for an object of a certain angular size we multiply the resolution by the fraction of $\frac{r}{\delta}$ (see Equation 4.3).

$$\text{pot} = \text{res} \cdot \frac{r}{\delta} \tag{4.3}$$

By substitution of the angular size using Equation 4.2, we can define the function f as shown in Equation 4.4 independent of size measures. This is a very simple model for calculating the pixels on target but it is sufficient to develop an algorithm. If needed, this model can easily be exchanged with a more complex one.

$$f(d_i, t_j) = \begin{cases} res_i \cdot \frac{360}{2\pi \cdot dist_{i,j}} \cdot \frac{1}{\delta_i} & \text{if } s_i \text{ is covering } t_j. \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

4.1.2 Resource Model

The resource model predicts the load generated by a certain task in terms of required CPU and memory resources. In addition, we calculate the energy demand of a platform under a certain load. To model the resources used by a certain task, we take into account the action which is performed, e.g. which computer-vision algorithm is used to process image data. Combined with parameters for image size and platform information we estimate the load generated on the target device. Using a lookup table, we can assess the energy consumed by the device during operation. This lookup table is constructed by measuring load and energy demand when running the real algorithms on the device. Another option to construct this model is using a mathematical description of algorithm complexity, platform performance and energy consumption information. This however is typically more complex to construct and is computationally more expensive at runtime and thus not a good option for using on an embedded system.

4.2 Central Algorithm for Static Environments

In this section, which is adapted from [19], we present a centralized solution for the problem presented in section 3 for environments with no dynamics. The presented algorithm is implemented based on the nEMO software framework for multi-objective optimization (see section 5.1). The search space for the combined coverage and resource allocation problem is typically very large. The number of potential solutions for this problem is very high. Thus, a combinatorial search strategy becomes infeasible. Since this search problem is also multi-objective (there exist multiple optimization dimensions), the solution is no single point in the search space but a set of Pareto-optimal solutions. A popular approach to tackle multi-dimensional optimization problems is the use of evolutionary algorithms (EA) [16] which are inspired by biological processes and apply the "survival of the fittest"

principle in an iterative way [108]. Only the best solutions found in one iteration of the algorithm are kept as basis to generate new, improved solutions.

In an EA, one permutation of the problem space variables is called a chromosome or individual. Many chromosomes form a population (the working set of the algorithm). In every iteration (also referred to as epoch) a certain number of chromosomes is altered by the genetic operators mutation and/or crossover. The mutation generates a copy of a single chromosome and alters some variables of the copied chromosome. The crossover recombines parts of two chromosomes to generate a new one.

During the breed of a new generation by mutation and crossover, the population may grow. To keep the population size constant, the fittest chromosomes must be selected at the end of each epoch. The selection strategy is an elementary part of the evolutionary algorithm and consists of two phases: (i) assigning a fitness value to each individual, and (ii) selecting a subset of the population based on the fitness values and the applied selection strategy.

The fitness of a chromosome is used to measure how it compares to other chromosome. It consists of one or multiple numerical values calculated by a fitness function. The fitness function is problem-specific and calculates how good a chromosome reaches the optimization goal. In multi-objective optimization, the fitness function typically calculates one numerical value per dimension. These values are then called a decision vector.

In the selection, the best chromosomes of the current population are chosen, the remaining are discarded. For single-dimensional problems, the population is sorted according to the chromosomes' fitness values. The chromosome with the lowest fitness are removed until the population size is reached.

In multi-dimensional problem spaces, this approach is not suitable since each chromosome has more than one fitness value. Typical selection strategies for multi-dimensional problems take only non-dominated individuals into consideration. An individual i is said to dominate another individual j if no element in the decision vector of i is smaller than the corresponding element in the decision vector of j while at least one element is larger.

If the selected population is smaller than the targeted population size, dominated individuals may be added to the population again to ensure a maximum diversity.

The execution of an evolutionary algorithm can be influenced by changing the targeted population size (i.e., the number of individuals present after selection), the mutation rate and the crossover rate (i.e., the number of mutation and crossover operations in one epoch). Evolutionary algorithms make heavy use of random variables (e.g., to select a chromosome to mutate).

4.2.1 Evolutionary Modeling and Approximation

Algorithm 1 Hierarchical evolutionary algorithm for approximating the camera coverage and task assignment problem.

```
proc coverage_and_assignment(S, T, A) ≡ //S,T,A as defined in section 3.2
  //OUTPUT: active sensors S' with assignments for D' and P̃
  //ENCODING: for every sensor s_i its status and input config. d_i ∈ D_i
  pop := set_initial_population()
  for e := 1 to number of epochs do
      mutate(pop)            //mutate sensor status and input configuration
      evaluate(pop)                          //assign fitness using Equ. 3.2)
      pop := select(pop)
      c := filter_covering_solutions(pop)    //take only "covering" solutions
      task_allocation(c)
      update(pop,c) //update the population with the task allocation results
      elitist_selection()
  od

  proc task_allocation(c): ≡
    //INPUT: sensor selections satisfying "coverage"
    //OUTPUT: feasible solutions with ranking
    //ENCODING: for every sensor s_i ∈ S' its assigned procedures p̃_i
    pop := set_initial_population
    for e := 1 to number of epochs do
        mutate(pop)                                  //procedure assignment
        evaluate(pop)                                  //(Equ. 3.1 and 3.3)
        pop := select(pop)
        elitist_selection(pop)
    od
```

We approximate the combined camera coverage and task assignment problem in a hierarchical, evolutionary approach (cp. Algorithm 1). As illustrated in algorithm 1, the algorithm takes the sets of sensors S, observation points T and activities A as inputs and returns a set of selected sensors $S' \subseteq S$ with assigned sensor configuration D' and procedures $\tilde{P}$.

In the first step, we focus on the coverage problem and search only for sensor selections and input configurations satisfying the coverage requirements (Equ. 3.2). At the end of each epoch, these "covering" solutions are passed over to a second evolutionary algorithm searching for feasible task assignments. This second step focuses on the resource and activity constraints (Equ. 3.1 and 3.3). Thus, the joint output of both steps satisfies

all conditions for *feasible* solutions which are ranked according to the specified fitness functions. Although our problem formulation considers scenarios with uncovered observation points (i.e., points which are outside the FOV of all cameras) as infeasible, our algorithm implementation is able to eliminate these points in a preprocessing step and still present solutions for all covered points.

Note, that the camera coverage and task assignment problem can also be approximated by a standard, non-hierarchically evolutionary algorithm. However, our hierarchical approach helps to significantly reduce the number of calls to the fitness functions which are computationally expensive and dominate the overall runtime of the evolutionary algorithm. In the following, we describe both steps of our approach in more detail. Note, that in both algorithm steps, we generate the initial population by randomly select initial resolutions, frame rates and activities from a given set. To generate an initial set of tasks for a certain activity (task assignment), we randomly select initial procedures that fulfill the given activity.

Camera Coverage and Input Data Configuration

In the first step we try to select the necessary sensors and set the resolution and frame rate such that every observation point is covered appropriately. The optimization goal is to minimize the data volume processed on all camera nodes.

For the genetic encoding, a chromosome is represented by the status (on/off) and the input data configuration d_{s_i} of each sensor s_i. The available input data configurations are represented in the set D. We only apply mutation as genetic operator which simply corresponds to randomly changing the sensors' status and input data configuration. The fitness function is given by Equ. 3.2. The decision vector generated by the fitness function is defined by two parameters. The first parameter is equal to the number of observation points "covered" by the chromosome, i.e., the number of observation points which have a properly configured camera covering them. The second parameter corresponds to a cost metric referring to the data volume processed at all sensors. It is calculated by the maximum possible data volume of all nodes normalized by the total data volume processed at all nodes.

$$costratio = \frac{\sum_{i=1}^{n} (res_{max_{s_i}} \cdot fps_{max_{s_i}})}{\sum_{i=1}^{n} (res_{s_i} \cdot fps_{s_i})} \tag{4.5}$$

Task Assignment

The "covering" solutions at each epoch of the first step serve as input to the second step. Here we try to find a task assignment for every camera

such that each activity of the covered observation points can be achieved and the resource requirements of the cameras are fulfilled. The optimization goal is the energy consumption and lifetime as specified in the optimization objectives (Equ. 3.4, 3.5, 3.6 and 3.8, respectively).

A chromosome is represented by the set of assigned procedures $\tilde{p}_i$ to all activated cameras $s_i \in S'$. The potential assignments of procedures are specified in P. In this step we also perform only mutation as genetic operator. Thus, the assignments of procedures to cameras are changed randomly. The fitness function is defined by Equ. 3.1 and Equ. 3.3. For each solution we can calculate the processing, memory and energy requirements, i.e., for each feasible assignment $\tilde{P}$ the required resources $\tilde{c}_{s'_i}, \tilde{m}_{s'_i}, \tilde{e}_{s'_i}$ are computed for all cameras $s'_i \in S'$ by applying the function $\tilde{r}$.

The function $\tilde{r}$ can be realized either by a mathematical model of the resource consumptions or by empirical measurements of the resource consumptions on the target hardware. For our algorithm we adhere to the second approach and measured the required resources for all algorithms and input data configurations on the available camera platforms. These resource values are stored in a table. Thus, the resource function $\tilde{r}$ can then be realized as a simple table lookup.

The fitness function for this algorithm checks if the resource requirements are met on all cameras. The first parameter in the decision vector is the total globally achieved quality calculated according to Equ. 3.8. The second parameter is computed according to the resource-related optimization goal. For criterion 3.4, we calculate the global energy usage e_{global} and use $\frac{1}{e_{global}}$ as fitness value. For criterion 3.5, we calculate the lifetime for node s_i and this value it as fitness value. For criterion 3.6, the minimum lifetime is taken as fitness value.

Elitist Selection

For both steps we use an *elitist selection* [108] method which stores the best found chromosome independently of the main population in order to avoid the loss of already found good chromosomes. In every epoch we add chromosomes to the elite which are not dominated by any other element in this elite. Note that chromosomes may remain in the elite. Thus, if the same chromosome is still in the elite in later epochs, we (re-)use the stored task assignment for that chromosome and can avoid the expensive execution of the second step, i.e., a call of the algorithm "task_allocation()".

If a feasible task assignment is found, the chromosome remains in the elite, otherwise it will be removed. By restricting the use of the task allocation algorithm to only the members in the elite (and not to all members in the population) we need to test only a small subset of the whole population. In fact, this approach guarantees that only the chromosomes with the best performance will be tested for resource and activity requirements.

4.2.2 PTZ Optimization

Our approach for sensor selection and resource allocation considers only static cameras. However, some camera networks might consist also of cameras with pan, tilt and zoom (PTZ) abilities. Whenever PTZ parameters are changed, the configuration needs to be updated. In this case, the EA needs to be run again using the new network layout.

To be able to use the evolutionary algorithm it in PTZ scenarios, we combine our approach with the expectation-maximization algorithm described in [67]. This approach uses activity maps. An activity map shows the distribution of activity in the area of interest over time. The PTZ parameters are reconfigured in a way, that the area is covered optimally, i.e. that hotspots in the activity map are better covered than areas with low or no activity. A detailed description can be found in [19].

After the PTZ optimization has been performed, it is necessary to re-do the task assignment. The output of the PTZ optimization algorithm is the location, orientation and zoom factor of every camera in the network.

This output is the input for the task assignment evolutionary algorithm. Observation points can be generated from activity maps i.e., an observation points are placed in areas of high activity, but can also manually placed by users. Afterwards the task assignment algorithm can find a new task allocation for the reconfigured PTZ network.

4.3 Distributed Solution for Low Dynamics

In this section, which is adapted from [21], we describe our distributed algorithm to solve the coverage and task assignment problem in scenarios with low dynamics. The basic idea is that camera nodes autonomously act in a greedy manner to cover observation points (also called targets) in their FOV. They then exchange messages (so called descriptors) describing the required resources to cover an observation point to inform other nodes of their local solution. Improved solutions are identified by comparing the exchanged descriptors. By performing periodic re-evaluation of the assignments (the targets covered by that camera), the solution can be improved iteratively.

We assume that the information on a new observation point is disseminated by a single node. The node might for example decide based on activity maps [67] where a new observation point must be placed. In this case, this node will be the first to transmit information about the new observation point. If an external operator defines new targets, this information will enter the network at a single sensor node, thus this node will further disseminate this information. We assume no knowledge on camera neighborhood, the cameras exchange descriptors with broadcast messages. If we know the neighbors however, we can multicast the descriptors to these nodes and reduce the communication in the network. Neighboring cameras are connected

via two edges in the coverage graph.

Each camera stores the best descriptor for a certain target, be it a local solution or the solution of a remote node. This stored descriptor is broadcast whenever the camera receives a worse descriptor. We support multi-hop dissemination of descriptors using this mechanism. This mechanism also improves the robustness against message loss.

Algorithm 2 shows a pseudocode description of the distributed algorithm. Nodes react to events such as the occurrence of new targets or the reception of new descriptors. Due to this simple protocol, we can quickly react to changed environmental circumstances and detected events in the monitored area.

Figure 4.1 shows a sample execution with three nodes. After node A broadcasts an initial descriptor for the new observation point, node C is the first to answer but the best solution is found by node B. In steps 3 and 4, the nodes A and C confirm the best solution of B by broadcasting the corresponding descriptor once again. This is also done to enable multi-hop communication and to compensate for message loss.

	Step				
Node	**0**	**1**	**2**	**3**	**4**
A	d_A			d_B	
B			d_B		
C		d_C			d_B

Figure 4.1: A sample execution pattern of the algorithm in a slotted representation. Node A defines the observation point, Node B has the best solution for it. d_X denotes a message which carries the solution found by node X.

4.3.1 Information Exchange with Descriptors

Nodes in our approach use descriptors to inform other nodes that an observation point has been covered with a certain amount of resources. A descriptor is a small packet containing only the identifier of the sender node, information about the observation point and the resources needed to cover it. Depending on the optimization goal, the resource field can contain e.g., the energy used to cover a point (this corresponds to the optimization goal of global min energy usage, Equation 3.4) or the remaining runtime that the node has in the current configuration before its energy reserves are depleted (this corresponds to the optimization goal of maximum network lifetime, Equation 3.6).

Algorithm 2 Event-based pseudo code of the distributed algorithm.

```
Algorithm distributed_coverage_and_assignment()
begin
  proc define(t) ≡                          //On define new observation point
    if in_FOV(t)
      then
          calc_settings(t, out res, out fps, out activity)
          d := calc_res(res, fps, activity)                  //d is a descriptor
          broadcast(d)
    fi.

  proc receive(d, t) ≡                      //On receive descriptor d for target t
    if ¬in_FOV(t) then exit fi
    if is_best_solution(d) then exit fi
    if ¬queue_exists_for(t)
      then
          q_t := create_queue(t)
          q_t.add(d)
          set_timer(q_t, τ + ε)
       else
          q_t.add(d)
          restart_timer()
    fi.

  proc timer_elapsed(q_t) ≡      //Timer of queue q_t for target t has elapsed
    d := best_descriptor(q_t)
    d_l := get_local_descriptor(t)
    d_s := get_stored_solution(t)
    d_b := best(d_l, d_s, d)
    if d_b = d
      then
          store(d) fi              //store d as remote best descriptor for t
    broadcast(d_b).

  do periodically ≡
     foreach uncovered target t in FOV do
       if t can be covered
         then
             calc_settings(t, out res, out fps, out activity)
             d := calc_res(res, fps, activity)          //d is a descriptor
             broadcast(d)
       fi od
  od

  do periodically ≡
     t := select_target()      //select one of the targets covered by this node
     d := get_local_descriptor(t)
     broadcast(d)
  od
end
```

4.3.2 Processing of New Targets

During the operation of the VSN, nodes can detect hotspots that need additional surveillance activities. For those spots, they are able to define new observation points with the respective surveillance requirements. There is also the possibility, that new observation points are introduced by external operators. This is done in the procedure *define*.

If a node defines a new observation point, it calculates the necessary resolution, framerate and activity needed to cover this points. Using these quality factors, it calculates which additional resource usage results from those settings. There are cases where a target can be covered "for free" if a node already covers another target that requires the same (or higher) settings. In all other cases, the additional resource demand is used as basis for a descriptor (the difference between the current resource usage and the predicted total usage if the target is covered). From these calculations, nodes construct a descriptor for this target which is sent out to the other nodes.

4.3.3 Processing a Descriptor

Nodes do not reply to each new descriptor at once. Rather, they wait for a configurable interval $\tau + \epsilon$ for other descriptors for the same target to arrive and they process only the best descriptor. Replying to new descriptors immediately would require a higher amount of messages to reach an agreement. ϵ is a small random factor for cases where the communication medium provides no medium access control (MAC). It prevents nodes from sending their results at the same time and to achieve a sequential communication pattern. On a MAC-enabled medium, ϵ is 0.

Upon receiving a descriptor (procedure *receive*), the node first checks if it covers the respective observation point. If so, it either adds it to an existing queue or creates a new one for this target.

4.3.4 Evaluation of Buffered Descriptors

After the buffering interval τ has elapsed, the node calculates a local descriptor (the costs for it to cover it locally) and also checks if it has a descriptor stored that was sent out by another node. This is done in procedure *timer_elapsed*. From those two the node selects the better descriptor and compares it to the best of the buffered ones. The best descriptor is stored locally as accepted solution and is broadcast to the other nodes as a confirmation. If the node's local descriptor is the best, it will cover the given target. A node that has no observation points assigned can shut down its sensory layer to save energy.

4.3.5 Periodic Activities

Each node periodically offers one of the targets it covers (typically the most "expensive" one, but other selection criteria can be chosen as well) in order to find nodes that may possibly cover that node with lower resource usage. This incrementally increases the quality of the solution. We call this mechanism the runtime re-evaluation of observation points.

Note, that the periodic offering of already covered targets is also used as a keep-alive signal to detect faulty nodes. This increases the robustness of the system and reduces the number of messages exchanged at runtime.

4.3.6 Initialization

The algorithm starts as soon as a node defines an observation point as new target. This may be done by an external operator who wants to re-focus the sensor attention to a certain area. Occurring events in the monitored area may require the network to focus on a certain area. Using activity maps [67], new observation points may be generated at activity hotspots. There are several other possibilities to generate new observation points that depend on the application of the sensor network.

In our approach, the nodes act greedily to cover observation points. Only upon receiving a descriptor with a lower resource consumption, it will hand off that observation point. A descriptor is broadcast by a node as soon as it considers the target covered or if it wants to re-evaluate the assignment of the node. This is also used to offer an already assigned target: the node responsible for that target just broadcasts that target's descriptor and if another node has a better descriptor, it is reassigned to that node.

4.3.7 Termination

Without runtime re-evaluation, the algorithm terminates after the nodes have covered all observation points or as soon as the nodes can have no more observation points assigned. As long as there are uncovered observation points, a node that is able to cover that point will try to do that. By enabling the runtime re-evaluation of assignments, the nodes constantly keep exchanging messages. The interval in which they try to hand off targets can be configured and enables faster improvement of the solution at short intervals or less message traffic at longer intervals.

4.4 Distributed Solution for High Dynamics

In this section (adapted from [20]) we look at resource-aware reconfiguration in very dynamic environments. Here, it is necessary that a reconfiguration algorithm is able to handle frequent updates and react also to short-lived

tasks. As a use-case, we look at how to integrate object handover into the distributed algorithm. The goal is to be able to hand over moving objects from one camera to another as they move through the surveilled area.

We tightly integrate the the socio-economic object handover algorithm described in [28, 29] with the distributed coverage and task assignment algorithm presented in Section 4.3 in order to deal with not only observation point reconfiguration but also object handover in a resource-aware manner. The network performs a long-term reconfiguration for tasks with low dynamics (represented as observation points). For highly dynamic objects that are tracked by the cameras, we additionally perform object handover between cameras as necessary. However, situations might occur, where the resource allocation at a certain camera does not permit to track an additional object. In this case, the camera will try to perform a reconfiguration in order to free resources before entering the auctioning process.

Algorithm 3 shows a pseudo-code description of our algorithm. As described in section 4.3, the distributed coverage and task assignment algorithm uses descriptors to exchange information between nodes. For a new observation point, the defining node calculates and broadcasts an initial descriptor[1]. As before, descriptors are buffered and evaluated at the nodes. The best descriptors are stored as accepted solution. Periodic re-evaluation allows for later improvement of the solution.

Further, this algorithm is now combined with the object handover algorithm [28, 29]. This is done to enhance long-term reconfiguration with the capability for handling moving objects in tracking algorithms. A node tracking an object can decide to perform a handover. If the handover requires a node to free resources, it will start a reconfiguration for its observation points.

4.4.1 Handover via Auctions

In the handover algorithm we use the passive approach as described in [28] where an auction is initiated whenever the tracked object is about to leave the FOV of the camera responsible for tracking it. In this case, the camera will send a message including an object description to initiate the auction. After waiting the auction timeout interval, the auctioneering camera compares all received bids and hands the object over to the winning camera.

[1]Note, that we describe the coverage and task assignment algorithm to be using broadcast communication. However, if the handover algorithm has built up a suitable vision graph, we can use this vision graph to switch to multicast operation in order to reduce the number of messages required.

Algorithm 3 Event-based pseudo code of the distributed algorithm. Procedures *define and* receive defined in Algorithm 2 remain unchanged and are omitted here for better readability.

```
Algorithm hybrid_coverage_and_handover()
begin
  proc timer_elapsed(q_t) ≡      //Timer of queue q_t for target t has elapsed
    d := best_descriptor(q_t)
    d_l := get_local_descriptor(t)
    d_s := get_stored_solution(t)
    d_b := best(d_l, d_s, d)
    if d_b = d
      then
           store(d)                    //store d as remote best descriptor for t
           broadcast(d_b)
    fi
    foreach pending auction for o_i do
      calc_settings(o, out res, out fps, out activity)
      b := calc_res(res, fps, activity)                          //b is a bid
      if required resources are within limits
        then
             send_bid_for(o_i,b)
      fi
    od.

  proc init_handover(o_i) ≡              On handover necessary for object o_i
    init_auction(o_i).

  proc process_auction_init(o_i)      On receiving an auction initialization ≡
    if is_visible(o_i)
      then
           calc_settings(o, out res, out fps, out activity)
           b := calc_res(res, fps, activity)                     //b is a bid
           if required resources are within limits
             then
                  send_bid_for(o_i,b)
              else
                  get_most_expensive_target(out t_j)
                  init_reconfiguration(t_j)
           fi
    fi.
end
```

4.4.2 Calculating the Bid

On receiving an auction initialization message, a camera determines if it is able to track this object o. If so, it will calculate the required sensor settings and resource requirements for performing this task. This is done using models of the hardware platform and of the surveillance activity (as described in section 4.1). In case the object can be covered within the resource constraints, the camera calculates the utility $u(o)$ for the object to express its valuation according to Equation (4.6). Thereafter, the camera will transmit a bid containing the utility as the offered amount of *money*. As usual in marked-based control systems, currency is an artificial construct and only used for management purposes; no real money is exchanged.

$$u(o) = e_n(o) \cdot pot_n(o) \cdot fps_n(o) \tag{4.6a}$$
$$e_n(o) = (e_{cur}(o)/(e_{max} - e_{min}) \tag{4.6b}$$
$$pot_n(o) = (pot_{cur}(o)/pot(o)) \tag{4.6c}$$
$$fps_n(o) = (fps_{cur}(o)/fps(o)) \tag{4.6d}$$

In Equ. (4.6a) we use normalized values for energy consumption $e_n(o)$, framerate $fps_n(o)$ and pixels on target $pot_n(o)$ (indicated by a subscript $_n$) to calculate a bid according to the resources required for tracking and the quality a camera can guarantee. The energy consumption in Equ. 4.6b is normalized between 0 and 1 depicting the minimum e_{min} and maximum e_{max} energy consumption on this node. e_{cur} describes the required energy consumption for covering the targets assigned to that node plus the energy required for additionally tracking o.

The normalized quality indicators pot_n and fps_n in Equ. (4.6c) are calculated by using the required framerate and pixels on target of o ($fps(o)$ and $pot(o)$) compared to the currently delivered quality ($pot_{cur}(o)$ and $fps_{cur}(o)$) that consider additionally its assigned tasks.

From Equation (4.6b) it can be seen that the value for $e_n(o)$ ranges from 0 to 1. Contrary, Equations (4.6c) and (4.6d) show that $pot_n(o)$ and $fps_n(o)$ can have values larger than 1. The values are relative to the quality required by o (where a value of 1 means that the requirements are exactly met). However, a sensor may deliver more or less than the required pixels on target.

4.4.3 Reconfiguration on Handover

If a camera receives an auction initialization but tracking the object would consume more resources than available, it initializes a reconfiguration for one of its assigned observation points. If it can free resources after the reconfiguration, it will participate in the auction.

Chapter 5

Tool support for VSN Reconfiguration

This chapter covers the software tools developed in course of this thesis. Each tool supports a part of the reconfiguration process. The nEMO framework aids in the development of EAs and is used to implement the central algorithm presented in section 4.2. nIPO and the Ella middleware are used in developing distributed, modularized applications. nIPO helps in realizing a processing activity a as described in section 3.2 by connecting individual algorithmic implementations (procedures p_i) to build a dataflow. The Ella middleware cares about message and data transport and aids in building fault tolerant, flexible distributed applications. As indicated in the respective sections, each software tool has been made available to the general public as open source software.

5.1 nEMO - A Framework for Evolutionary Multiobjective Optimization

In this section (adapted from [19]) we show a software framework for developing evolutionary algorithms. It is called nEMO[1] (.net Evolutionary Multi-objective Optimization).

To accelerate the development of algorithms and to minimize change effort, we have implemented a generic framework for evolutionary single- and multi-objective optimization problems. It facilitates the reuse of core evolutionary algorithms, encoding of the chromosomes, and the specification of fitness functions for decision vectors of arbitrary lengths (≥ 1). Consequently, our framework is able to perform multi-dimensional approximations.

nEMO is implemented in C#.Net and is compatible to Mono[2] and can thus be used on various platforms including Windows, Linux and MacOS.

In order to solve a new optimization problem using nEMO, the following tasks need to be performed:

1. Define the model and implement a corresponding chromosome along with suitable mutation and crossover operations

[1] http://nemo.codeplex.com
[2] http://www.go-mono.org

2. Define the fitness function

Figure 5.1 gives an overview on the architecture of nEMO. It defines basic interfaces for fitness functions, chromosomes and selections. To implement a new EMO based on nEMO, a class implementing *IChromosome* must be created. In it, the model and its manipulations can be implemented. A class implementing *IFitnessFunction* provides the corresponding evaluation facilities. Chromosome and fitness functions are problem-specific and have to be created for each new algorithm. However, selection methods are already provided with nEMO. It contains elitist and non-elitist selection method. If necessary, user-defined selection methods may be used by implementing a subclass of the *SelectionBase* class.

The framework will then run the evolutionary optimization autonomously. nEMO comes with predefined single- and multi-objective selection strategies. However, the selection strategy can be modified if necessary as well.

The *Optimizer* class is used to control the algorithm itself. It requires a fitness function, an ancestor, the population size, mutation and crossover rates as parameters. Each epoch of the algorithm is then triggered by a dedicated method. The core algorithm of the framework performs mutation, crossover and selection in each epoch. Optionally, it can enforce the population size in cases where the number of non-dominated solutions is smaller than the population size. In this case, it will randomly select as many chromosomes as necessary from the dominated solutions. Large parts of the algorithm can automatically be parallelized to achieve higher performance on multi-processor systems. This is done in mutation, crossover and fitness assignment steps. The selection is done in parallel as well.

5.2 Dataflow Processing in VSNs

In this section (adapted from [22]), we look at the way data is processed in typical VSN applications and how software can support this.

In many VSN applications, a dataflow towards as sink (e.g. an operator computer) can be seen. Certain steps of processing have to be performed for every source data stream (i.e. video from camera sensor) before the data is finally processed at the sink (e.g. displayed or stored). As an example, an activity a as defined in section 3.2 like simple object tracking can be realized in multiple steps. First, data is captured from a sensor. Then a background subtraction is performed. Third, objects are detected in the frame and finally association to objects in the previous frames is performed. This sequential dataflow from the source through all processing steps is very common to VSN applications.

A VSN application needs to be flexible in order to be reconfigured at runtime. Processing parameters (like framerate or resolution) may be changed or parts of the application may be exchanged or moved to other nodes.

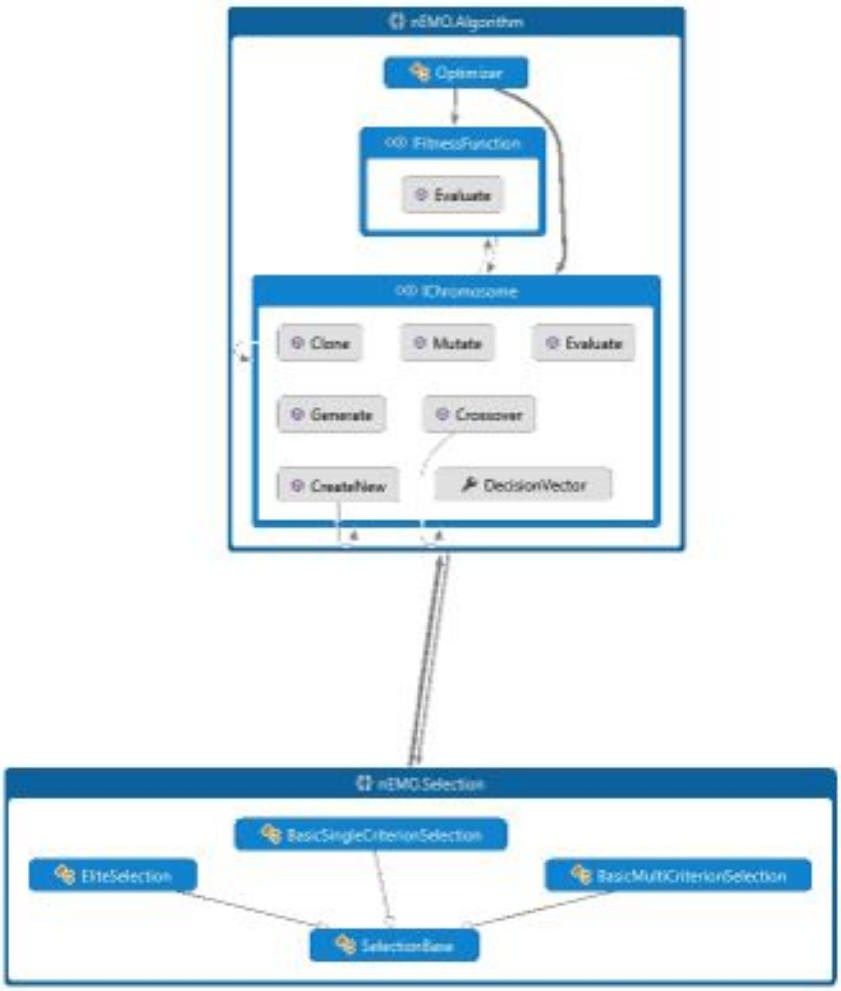


Figure 5.1: An overview of the nEMO architecture

The software framework presented in this section can be used to quickly reconfigure such an application by exchaning plug–ins at runtime.

Performing all processing steps at either the smart camera or the sink may not always be the best solution. Instead, some parts of processing can be done at intermediate nodes or split between source and sink nodes. Assuming a network of heterogeneous devices and heterogeneous capabilities, the search space for this optimization problem increases drastically. In section 5.2.2, we present an evolutionary algorithm which is designed to find suitable solutions for this problem. First, we show a software framework for dataflow-based applications.

5.2.1 The nIPO framework for Dataflow Processing

To facilitate implementation of (distributed) dataflow applications, we present the nIPO[3] (.net Input Processing Output) framework. It features a flexible structure for exchange and combination of single modules using a dedicated plug–in system. It can be used to realize dataflows in distributed and non-distributed applications where each step of the dataflow may easily be exchanged if necessary.

In this section we first explain the requirements that we identified for this software. We then present our software design which fulfills those requirements and explain further details on the framework.

Requirements

For our software we identified some key requirements that have to be fulfilled in order to provide a platform that can be used in large–scale clustered networks.

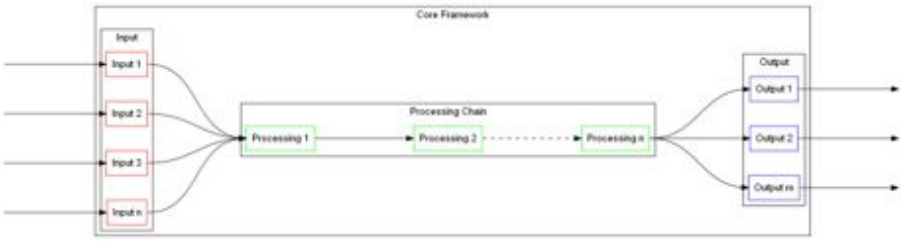


Figure 5.2: Data flow within the framework. One or more input plug–ins continuously pass data to a chain of processing plug–ins. Resulting data is transferred to multiple output plug–ins.

Scalability:
The software has to be able to accept a variable number of inputs and deliver a variable number of outputs. This is essential to evaluate clustering

[3] `http://nipo.codeplex.com`

by varying the number of connected cameras. Additionally, the processing work performed in an instance must be variable and easy to change. To allow different types of work to be performed in the cluster node, a mechanism to easily exchange the processing part must be provided.

Composability and Flexibility:
We want applications which are built on top of our software to be composed from exchangeable modules. This increases the level of code reuse and thus speeds up application development.

To support a variety of application scenarios, the software should not restrict the data flow by means of data type or data content.

Deployment Support and Monitoring:
The software is intended to provide facilities to easily get a VSN node up and running. This includes selecting and composing the processing part, collecting the necessary configuration values and deploying the software modules.

To collect performance indicators in a large network, a VSN node must be able to monitor itself and collect relevant data. In some cases it is not sufficient to just monitor process specific parameters like CPU usage or memory consumptions. A mechanism to monitor the data produced by each step in the dataflow is required.

Further Requirements:
We want our software suitable for many application scenarios, thus it must also be easy to use for developers and users. Additionally the software needs to be platform independent and thus be applicable in many environments. To provide a powerful but resource–saving platform, we try to minimize the memory and performance overhead caused by the software itself.

Framework Design

The framework design reflects a sequential dataflow in an application but defines also further types of modules to provide more flexibility in designing applications.

nIPO is implemented in C#.Net 2.0. It runs on any platform where a .NET or Mono[4] runtime is available, i.e. at least under Windows, most Linux distributions, Solaris, BSD and MacOS.
Plug–In Structure:
Many applications can extend their functionality using plug–ins. Contrary, in our framework the plug–ins actually determine the functionality of the application. In its core, nIPO defines a classical IPO (input-processing-output) chain of plug–ins. However, this concept is not strictly enforced.

[4]http://www.mono-project.com/

There are also plug–ins which are not part of the processing chain itself and nIPO allows for loopbacks in the data flow.

nIPO plug–ins communicate via events. There are predefined events for the IPO chain plug–ins to pass data from one stage to the other. For each data packet that an input plug–in produces it raises an event. This event is then handled by the processing plug–in. Each plug–in can define its own events and use an event-registrar to make it accessible to other plug–ins. Using this you can define your own communication flow within nIPO (like loopbacks and branches).

nIPO defines the following plug–in types:

Input: Input plug–ins capture or generate data from any given source like files, network or sensors. Input plug–ins run in dedicated threads and continuously pass data to the processing chain.

Processing: Processing plug–ins perform the actual processing work. This might be e.g., image processing or any other data processing task. The processing part can be composed from multiple processing plug–ins which form a pipeline (further called processing chain).

Output: Output plug–ins are the data sink of the IPO pipeline. They may store data, display it or send it via the network.

Service: Service plug–ins provide certain services to plug–ins (like shared data or utility functions, e.g., camera calibration). nIPO defines a generic service discovery mechanism that can be used to search for a certain service. If the service type is known in advance, a plug–in can also search for a service by its (.net-) type.

Observer: Observer plug–ins have no pre–defined functionality. They act as pure event handlers (listeners or observers) and are invisible to other plug–ins. Using the generic event discovery mechanism, they can catch events in nIPO. This enables to extend the core nIPO functionality. As an example, think of a plug–in that collects statistics on the data flow (how many bytes of data are processed, what is the system load, ...). Such a plug–in would be implemented as an observer and just consume internal events. There are various other application scenarios for observer plug–ins like providing a user interface (this may also be done by an output plug–in) or providing instance dependent services to an external application.

Figure 5.2 gives an overview on core structure and the IPO dataflow within nIPO.

Generic Data Transfer:
For information interchange between plug–ins an asynchronous, event-based mechanism that is independent of data type and content is implemented. Thus, arbitrary data may be processed and passed within the framework as long as the receiving plug–in is able to interpret the incoming data (i.e.,process the type of data delivered to it).

A computer vision application that works on low level image data will most likely pass images between plug–ins while a fusion algorithm for high–level data may pass only lists of detected objects. nIPO performs a compatibility check to ensure that all plug–ins receive only supported data.

This enables a wide spectrum of applications and the exchange of single plug–ins without the need to exchange or recompile other parts. In the context of a camera network we can receive the input data for a multi-view fusion algorithm either from synchronized video files or from live data over a network connection (or any other data source). The processing plug–in containing the fusion algorithm does not need to be exchanged when the input source changes. Also for a certain set of input video files we can compare different fusion algorithms by simply exchanging the processing plug–in.
Monitoring:
To achieve a generic mechanism for performance monitoring we provide a monitoring interface which, again, is plug–in based. Every monitor plug–in may listen to events in the system in order to extract performance information. This may be done with or without active support by each plug–in, i.e. plug–ins may define and dispatch certain events that are performance related (e.g. the amount of bytes transferred over a network connection).

In cases when just processing events is not sufficient (e.g., whenever the data flowing between plug–ins must be inspected) we insert dedicated proxy plug–ins into the dataflow which dumps or analyzes the incoming data and then forward it to the next plug–in. This procedure is visualized in figure 5.3.

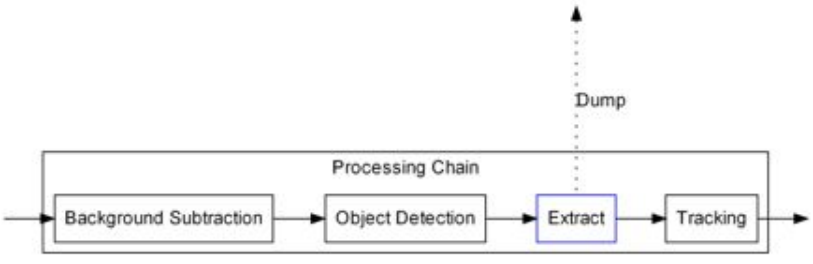


Figure 5.3: The result of the object detection algorithm is dumped transparently for all other plug–ins.

5.2.2 Finding an Optimal Dataflow in VSNs

Based on nIPO we now present an algorithm which can optimize the dataflow in a VSN. It models sensor nodes of heterogeneous architecture and the dataflow between them in the chromosomes. The fitness function uses the models of hardware and algorithms presented in section 4.1 to compute the required resources and thus assign fitness values to chromosome instances.

Chromosome

The chromosome models the network as a list of nodes where every node has a hardware platform (e.g. an ARM- or Intel Atom-board) and a set of available algorithms. Each node in the network has a target node where the result of the local processing is transmitted to. Before transferring data over the network, a compression step is automatically inserted. Just like the algorithm presented in section 4.2, this algorithm can also choose from a number of resolution and frame rate settings to find the optimal balance between quality and processing load.

Since nodes can delegate tasks to other nodes, each node can contain multiple processing pipelines, one for itself and several others from other nodes. This mechanism can be used to move data processing along the data flow between source and sink of each flow. The actions to perform for each source stream are pre-defined (e.g. "tracking must be performed for stream 1"), however the concrete algorithms to achieve this are not (as described as A and P in section 3).

In the mutation step, this chromosome can perform the following actions:

1. Change sensor settings, i.e., resolution or framerate.
2. Move a task to the target node.
3. Move a task back to the preceding node (i.e., the node where it receives its input from).
4. Change the target node of a node, i.e., direct the dataflow over a different node.
5. Change an algorithm for a certain task type, e.g. choose frame differencing for background modeling instead of mixture of Gaussians.

Fitness Function

The fitness function grades each chromosome according to one of the following goals:

1. Global minimum energy consumption

2. Maximum network lifetime, i.e. the time until the first node fails

3. Maximum lifetime of a target node, i.e. the time until the battery of this node is depleted.

The fitness function uses models of the algorithms available at each node to calculate the estimated CPU and memory consumptions. Those values are then used to estimate the energy consumption.

Models

The models used for calculation of resource demands are the same as for the algorithm in section 4.2. Algorithms are tested on the target platforms to build estimation models of CPU and memory demands. The energy consumed under different workloads is measured and from this an estimation model for energy consumption is built.

5.3 Middleware for Distributed Processing in VSNs

The development of distributed operation in a VSN can be very well supported by a middleware system. Typical operations like data transport, coordination or node discovery can be supported by a middleware and do not need to be implemented in every application individually.

This section describes a middleware system to help develop distributed smart camera applications[5].

5.3.1 Requirements

In order to be useful in the context of VSNs a middleware must match several requirements. First, it must provide meaningful services to the application (e.g., communication). Second, it must be flexible enough to allow for application restructuring and quick exchange of functionality. Thus, applications should be constructed from several modules or plug–ins. Third, it should be made easy for developers to port their existing code to the middleware system. Further the middleware should provide mechanisms to decouple individual modules, i.e. make them independent from each other wherever possible.

Further, it is required that the middleware is simple, both in structure and in being simple to understand. This enables it to be used without a big effort to learn its usage concepts. Further, a simple structure makes it faster in execution and easier in maintenance.

In the following sections, we present Ella, a distributed publish/subscribe middleware system.

[5]This section is adapted from [23]

5.3.2 Architecture

In this section, we present the architecture of the Ella middleware. While there are several paradigms for middleware systems, like RPC or tuple spaces, we chose a data-driven approach, namely a publish/subscribe system. This allows for a high degree of flexibility since it provides decoupling in space and time[30]. In contrast to other publish/subscribe implementation (e.g. [96]), Ella is completely distributed without the need for any central coordination.

Publish/Subscribe

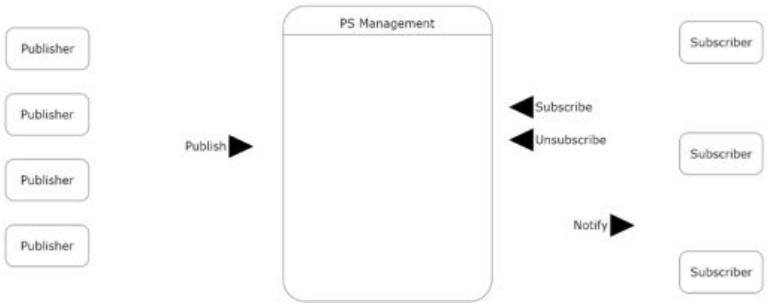
Publish/Subscribe, as illustrated in figure 5.4 is a mechanism which allows for elegant decoupling of functional elements within applications. Publishers are modules which produce data (e.g. capturing of images, fetching data from a specific source) and pass it on to consumers which are called subscribers which process the data. A module may be publisher and subscriber at the same time, i.e. it processes data from another publisher and publishes the results itself. A component for publish/subscribe management takes care of decoupling. Instead of directly connecting the publisher and subscriber modules, a publisher announces its events and a subscriber can announce interest in certain types of events. The publish/subscribe management takes care of matching published events and subscriptions and is also responsible for delivering the published data to all subscribers. A key requirement of publish/subscribe is that neither publishers nor subscribers need to be aware of each other. A publisher does not need to keep track of where its data is sent to and how many subscribers exist for its events, and a subscriber does not need to care about where publishers are located and where their data is coming from (i.e. the local node or a remote node). All this is transparently handled by the publish/subscribe middleware.

As illustrated in figure 5.4 a) this can in a simple case be done at a central component which keeps track of all publishers and subscribers and relays data to the correct subscribers. However, an architecture like this is hardly scalable because the central component may become a bottleneck. Figure 5.4 b) illustrates a distributed publish/subscribe architecture, where a small publish/subscribe management component resides on each node in the network and is responsible for its local modules and for connecting them to remote nodes.

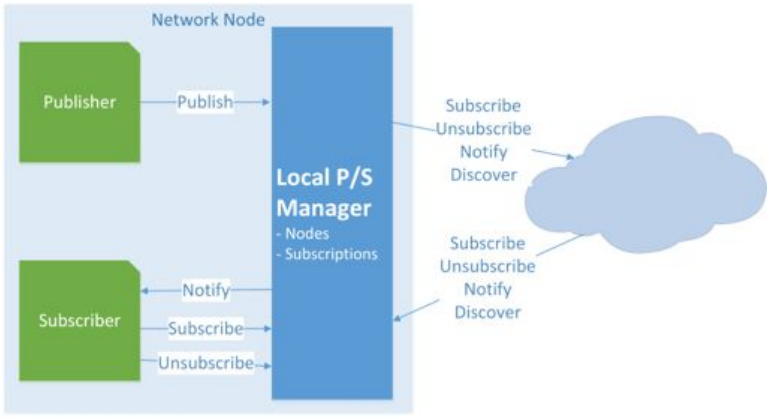
Decoupling

A publish/subscribe system enables decoupling in the following dimensions:

- Space decoupling: Modules do not need to know where they and other modules are located in the network. This means that publishers do not hold any references to subscribers and vice versa.



(a) Centralized publish/subscribe



(b) Distributed publish/subscribe

Figure 5.4: Illustrations of centralized and distributed publish/subscribe

In a VSN, e.g., a publisher of images does not need to care if they are delivered to one or more displays or other modules.

- Time decoupling: Publishers and subscribers do not need to participate in an interaction at the same time. The publisher might for instance publish an event while there is no subscriber connected. Publishers which start after a subscriber can still be matched to an earlier subscription request. In a VSN, cameras may not start up at the same time, still they must form one distributed application.

- Synchronization decoupling: Preparing events does not block the publishers, and subscribers can be notified of an event, even though they are currently executing another activity. As an example, a publisher of images can capture the next image while the current image is still delivered to subscribers.

Subscribing

Ella uses type-based subscription, i.e. a subscriber specifies a certain data type to subscribe to. As an addition, Ella provides the possibility to request a template object from each subscriber. The middleware will then ask each publisher (which is matching in type) to generate such a template object and will hand it to the prospective subscriber. The subscriber can then decide whether this specific publisher is accepted or not. In addition, upon subscribing, the subscriber can decide to exclude remote publishers to obtain only subscriptions from the local node. This can be useful whenever node-specific information is requested like the current resource allocation.

Publishing

Creating a publisher for Ella is a straightforward task. Instead of forcing developers to implement a specific class hierarchy (i.e. subclassing a publisher base class or implementing a publisher interface) we use code annotations to declare a certain class to be a publisher. These code annotations can be reflected by Ella at runtime to detect publishers in code libraries. This approach has several advantages. First, it makes it very easy to adapt existing code to run on top of Ella. This requires mainly annotating the existing code and adapting the way of data passing to the mechanism provided by Ella. Second, it makes the development of modules more flexible because developers are not bound to any inheritance hierarchy and thus it it easier to integrate Ella into any software architecture. Third, Ella-based code is easily readable and maintainable because the annotations directly inform about what the specific module does.

Listing 5.1 shows an example publisher which publishes the current time. It provides this in two ways, first as a formatted string and second as a DateTime object. The published types are declared in brackets on top of the class definition. Each published event type must have a unique identification number. This enables one module to publish an arbitrary number of event types. Further attributes identify methods used to create new instances and start or stop an instance. The *TemplateData* attributes are used to identify methods which provide template objects. However, this is optional and needs not to be implemented by every publisher.

Listing 5.1: A sample publisher

```
1
2 [Publishes(typeof(String), 1)]
3 [Publishes(typeof(DateTime), 2)]
4 public class TestPublisher
5 {
6   private bool _run = false;
7
8   [Factory]
```

```
 9   public TestPublisher() { }
10
11   [Start]
12   public void Run()
13   {
14     _run=true;
15     while(_run)
16     {
17       Publish.Event(DateTime.Now.ToLongDateString(), this, 1);
18       Publish.Event(DateTime.Now, this, 1);
19     }
20   }
21
22   [Stop]
23   public void Stop() {
24     _run = false;
25   }
26
27   [TemplateData(1)]
28   public string GetTemplateObjectString(int id)
29   {
30     return id == 1 ? DateTime.Now.ToLongDateString() : string.
           Empty;
31   }
32
33   [TemplateData(2)]
34   public DateTime GetTemplateObjectDateTime()
35   {
36     return DateTime.Now;
37   }
38
39 }
```

Network Management and Remote Operation

To support a convenient way of developing and deploying software modules for Ella, the middleware provides a transparent node discovery mechanism which is used to detect any running Ella instances on other nodes in the network. This relieves the developer of the need for managing other nodes in the network. As soon as an Ella instance is detected, it is registered as a known host and it will also be checked for suitable publishers of events requested by local subscribers. This way, it is much easier to scale an existing application without having to modify existing code. As soon as Ella detects other instances, it will include them in its operation.

On startup, Ella tries to first discover other nodes in the network. By default this is done with a UDP broadcast. This broadcast also contains connection information necessary to address this node in the network. However, this may be exchanged with any other suitable discovery provider (e.g. for non-IP compatible media like ZigBee). Upon reception of a broadcast message, a node will send a unicast answer to the broadcasting node with

its own connection information. Thus, each node keeps a local directory of known remote hosts. This directory is used when searching for matching publishers on other nodes.

Whenever a subscriber requests a new subscription, all remote hosts will be inquired about matching publishers. If some are found, proxy objects at the remote node and stubs at the subscriber node will be created which act as transparent transport points for published event data. A proxy acts as subscriber at the remote node, serializes the event data and sends it to the stub. The stub deserializes it and publishes it as a local publisher for the original subscriber to receive.

The requested subscription types by each subscriber module are cached by the local Ella instance. Whenever a new node is discovered in the network which runs suitable publishers, they will start to deliver their events to the local node.

Communication

Ella instances on remote nodes use an efficient message structure to exchange data. A binary protocol is used to encode message types and to transport any necessary data. For any given message payload, only 9 bytes of overhead are added, one byte for the message type, and four bytes each for the sender node ID and the message ID. For small networks this can be reduced by only using single bytes for the sender node ID.

Besides data communication, Ella provides also a control channel which can be used to exchange application-specific messages between publishers and subscribers. Instead of transporting data, control commands can be sent between publishers and subscribers. As an example, an image processing module could instruct the image capturing module to adapt its framerate.

Implementation Details

Ella has been developed in C#.Net. It is capable of running in the open source Mono[6] runtime and can thus be deployed on all major operating systems and many other platforms. Since it is only performing high-level tasks like I/O and management of subscriptions, its overhead compared to a native implementation is very low. In addition, it is easily possible to integrate native code components into any .Net application. Thus, performance critical applications parts can be written in e.g. C++ and be integrated into Ella with low effort. Of course, pre-existing native code can be integrated as well.

Static Facade:
Ella can be accessed from an application using a facade pattern with static

[6] http://www.go-mono.org

classes and methods. The key goal is to provide a fluent and easy-to-understand interface to the middleware. The most important classes and methods of the facade are shown in figure 5.5.

Listing 5.2: A sample main method for Ella. Calls to facade classes are shown in red.

```
1  public static void Main(string[] args)
2  {
3    Start.Networking();
4    TestPublisher p = new TestPublisher();
5    if(Is.Publisher(p))
6    {
7      Start.Publisher(p);
8    }
9    TestSubscriber s = new TestSubscriber();
10   Subscribe.To<DateTime>(s, s.Notify);
11   Console.ReadLine();
12   Stop.Publisher(p);
13   Stop.Networking();
14 }
```

The application programmer as well as a module developer do not need to take care about any instance handling of the middleware since this is handled internally by Ella. Instead of instantiating a manager class and adding publishers to that instance, a call to *Start.Publisher* handles all of it. The code sample in figure 5.2 shows an example of how to initialize Ella with new modules.

Subscription Handling:
On each node, Ella keeps a list of all subscriptions relevant to this node, i.e. all subscriptions where modules of this node are publishers and/or subscribers (this is also true for subscriptions only on the local nodes). Whenever a publisher publishes a new event, all subscribers are found in this list. In the simplest case, this data is delivered to a local subscriber (which is on the same node). For remote subscribers, this list contains the proxy at the publisher node. A proxy serializes the event data and sends it to the receiver node. There, a stub reconstructs the data and publishes it locally for the intended subscriber to receive it. In cases, where unreliable transport can be used to deliver data (i.e., where loss of data can be tolerated), a UDP multicast mechanism can be used in order to save communication costs.

Event Correlation:
In some cases, a user might want to indicate that two events are somehow correlated. For example the image of a camera and the result of a tracking algorithm might correspond to each other. For this case, Ella provides a simple mechanisms where a publisher can indicate such a correspondence. This is then delivered to all modules which subscribe to both events.

Figure 5.5: The static facade used to access Ella functions

5.3.3 Requirements Conformity

Ella provides several useful services to an application. It handles discovery and communication, provides a control channel and helps in modularizing an application. It enables the flexible reconfiguring of an application with its module-based architecture. The use of code annotations to declare Ella-specific code regions makes it very easy for developers to port their existing code. The transparent subscription mechanism of publish/subscribe enables decoupling in space, time and synchronization of modules.

5.3.4 Use Case: EPiCS

In the EPiCS project[7], the Ella middleware was used to build a single-object multi-camera person tracking system which uses a market-based algorithm to handover the tracking responsibility from one camera to another.

In the EPiCS case-study application, we perform distributed person tracking on multiple cameras where at any point in time, only a single camera is responsible for tracking the person. In the event that the person leaves the field of view (FOV) of the currently tracking camera or is no longer detected, the camera initiates a handover where the other cameras

[7] `http://epics-project.eu`

try to find the person. This is either done on all cameras, or on a subset of neighboring cameras. The neighborhood of a camera is the set of other cameras which are possible targets to hand an object over to. The handover is performed by means of an auction as described in [28]. In any case, each camera typically streams its currently captured video stream to an operator PC running a graphical user interface.

Looking at the global dataflow of this application, this means that the tracking module is not always active on every camera. Only after winning the handover auction, the tracker is activated.

As depicted in figure 5.6, three major modules are concerned in the realization of this system. The *CV* component is responsible for performing the computer vision tasks of acquiring images from the sensor and performing tracking as soon as the tracking responsibility is assigned to the respective camera node. The *Handover* component takes care of agreeing with other cameras on the tracking responsibility. In case a tracked object is about to leave the field of view of the camera, the *Handover* component will initialize an auction. All cameras which detect the object can send a bid for it. The camera with the highest bid will win the auction and will then be responsible for further tracking the object. The *UI* module is a visualization component for the security operator who will initially select the object to be tracked.

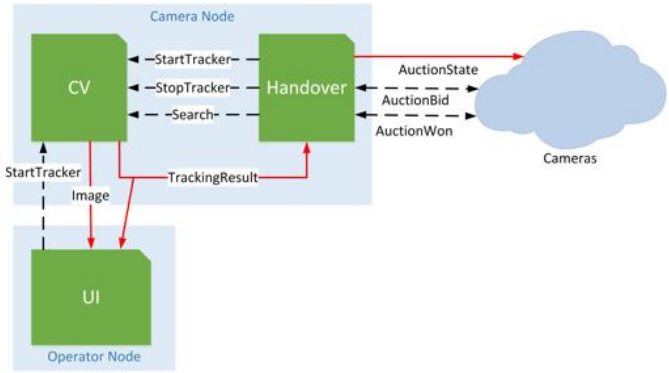


Figure 5.6: The realization of the EPiCS use case on top of Ella.

Using the publish/subscribe mechanism and the control channel in Ella, this application can be realized with an elegant decoupling of components. First, *CV* is publishing images (to be displayed in *UI*) and tracking results (containing the location in the image and the confidence of the tracked object) in case it is responsible for tracking. Both, *UI* and *Handover* are subscribed to this tracking information. *UI* uses it to show the bounding box of the tracked object as overlay to the camera image stream. *Handover* is informed of the current tracking status with this information and can react to e.g. a lost object.

Handover is publishing an auction state event which initializes and ends

an auction. If a camera wants to contribute to an auction with its own bid, it will use the control channel to directly address the auction initiator. *Handover* uses tracker control messages on the control channel to start and stop the tracker and to pass the model of the object to be tracked to the tracker. *UI* does the same once the operator has selected an object to track in one of the cameras' views.

Chapter 6

Evaluation

In this section (adapted from [19, 20, 21]), we evaluate the algorithms presented in this thesis. We first look at the central algorithm and then compare the distributed solutions to its benchmark results. Before presenting detailed evaluations of the algorithms, we present the hardware platforms used to build the resource models used by the algorithms.

Hardware platforms

The hardware platform on which a VSN application is executed plays a major role for the minimization of energy consumption. Minimizing the resource consumption at the software level makes only sense if the underlying hardware is already energy efficient.

In course of this thesis, two types of smartcam platforms were employed. First, we use Intel Atom-based smart cameras built by SLR Engineering[1]. As a second platform, we use custom-built PandaBoard[2]-based camera systems (as shown in figure 6.1).

The SLR cameras are equipped with an Intel Atom processor running at 1.6 GHz and an 100 MBit Ethernet interface. Furthermore, the SLR camera has a CCD image sensor with a native resolution of 1360×1024 pixels.

In contrast, the PandaBoard platforms are equipped with Logitech HD Pro c920 webcams. The PandaBoard platforms are based on Texas Instruments OMAP 4430 system-on-chip which features a dual core ARM Cortex-A9 MPCore CPU running at 1.2 GHz and uses a 802.11 b/g/n wireless connection to the network. The connected Logitech c920 webcam operates with a native resolution of 3MP. The PandaCam is the current result of our efforts to build an energy-efficient smart camera with high computing power.

6.1 Evolutionary Approximation

To evaluate the central EA, we perform systematic tests of our algorithm using a simple, a medium and a complex scenario. We evaluate the impact of parameters such as population size, mutation rate and number of epochs on

[1] www.slr-engineering.at
[2] http://www.pandaboard.org

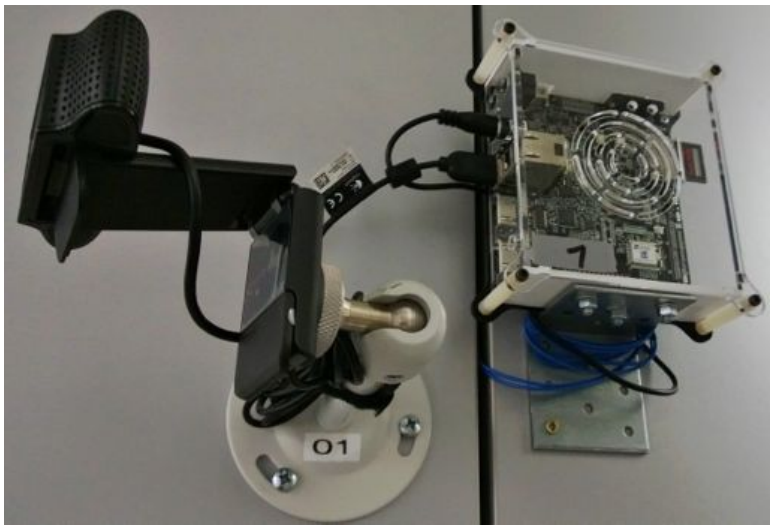


Figure 6.1: The Pandaboard-based camera using an USB sensor.

Point	*pot*	fps	*activity*
t_1	10	8	change detection
t_2	10	4	change detection
t_3	20	18	object tracking
t_4	15	8	object detection

Table 6.1: The quality requirements of observation points 1-4 expressed as pixels on target, frames per second and activity.

the performance of the evolutionary algorithm. We further study the runtime of our algorithm, explore the tradeoff between surveillance quality and resource utilization and compare the predicted resource of the assigned tasks with the measured resource consumption on the target platform. Finally, we evaluate the integration of the PTZ optimization.

6.1.1 Scenarios

Simple scenario

Our first scenario is the example setup from section 3 (see figure 3.1(a)). This simple scenario consists of five cameras and four observation points on a 10000 m^2 area. The requirements of the observation points can be seen in table 6.1. The algorithms used in the task assignment are shown in table 6.3 (these apply to all scenarios).

For this simple scenario, a single optimal solution for sensor selection and sensor configuration exists if the following preconditions are fulfilled: i) all processing tasks have an equal quality assigned ii) no node has resources

Sensor	*res*	*fps*	*activity*
s_1	SQCIF	8	change detection
s_2	QCIF	4	change detection
s_3	QVGA	18	object tracking
s_4	off	off	off
s_5	VGA	8	object detection

Table 6.2: The optimal solution for the simple scenario assuming no previously allocated resources on the nodes.

allocated to other tasks initially and iii) at most one activity per sensor is assigned. Then, s_4 is switched off and object tracking is assigned to s_3. This is because s_3 is closer to t_3 and can thus provide *pot* coverage at a lower sensor resolution. The optimal configuration for this scenario is shown in table 6.2. The total global data volume is at about 5.6% of the maximum global data volume.

Assuming that the amount of free resources of s_3 is initially reduced to 10% of the available resource (i.e. it already performs other tasks), the task of object tracking can only be assigned to s_4 and s_3 should be switched off.

In our evaluations, we show the success rate for the simple scenario. We show how often the algorithm finds the optimal result for this scenario. For all scenarios, we show how often it finds a feasible result at all.

Complex scenario

We tested our approach in a more complex scenario of 100 sensors and 20 observation points in an area of 62500 m^2 (see figure 6.2). The observation points require between 10 and 30 pixels on target.

PTZ scenario

We demonstrate the combination of our approach with the PTZ coverage optimization in a medium sized scenario with eight cameras and five observation points on an area of 8000 m^2. We first perform the expectation-maximization algorithm to find the optimal PTZ-parameters of each sensor. Then, we run our evolutionary algorithm to find the optimal sensor configuration and task allocation.

Impact of increasing number of solutions

We further evaluate the behavior of our algorithm in scenarios that are within the same level of complexity but which have a different number of possible solutions. In a basic scenario of seven observation points we vary the number of covering sensors. Every additional covering sensor increases

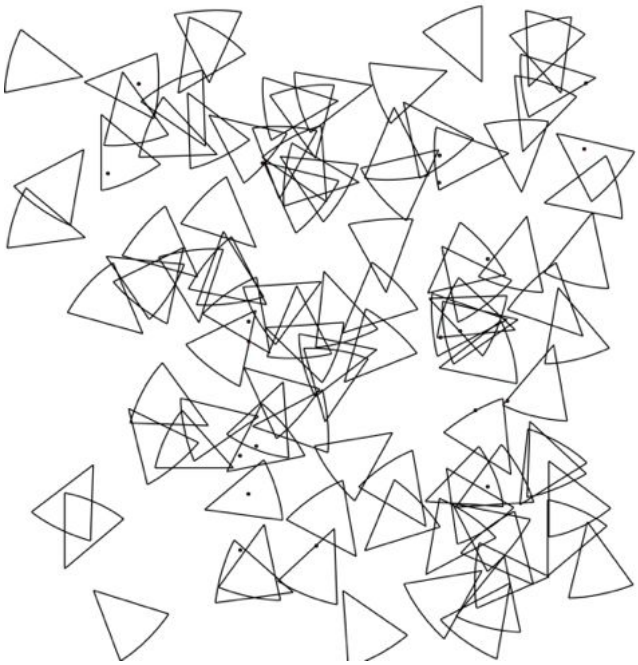


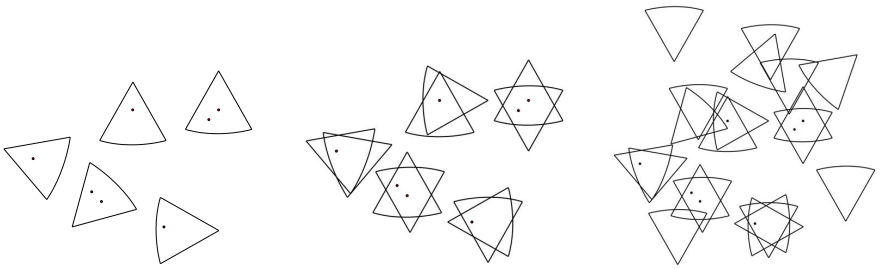
Figure 6.2: The randomly generated complex scenario consisting of 100 camera sensors and 20 observation points.

the number of solutions. This impacts the runtime and the final result. Our basic scenario has five cameras placed such that every point is covered by exactly one camera. We then add cameras to achieve degrees of overlap of two, three and five. Additionally, we constructed two scenarios that have additional non-covering cameras. The scenarios are shown in figure 6.3.

For these scenarios we show multi-objective approximation including the quality ratings of algorithms. We manually assign a quality rating to each algorithm for our second algorithm stage (see table 6.3).

Algorithm	Quality
Simple Frame Differencing	0.5
Double Frame Differencing	0.8
Mixture of Gaussians	1
Blobfinder	1
Kalman Tracker	2
CCCR Tracker (openCV)	0.6

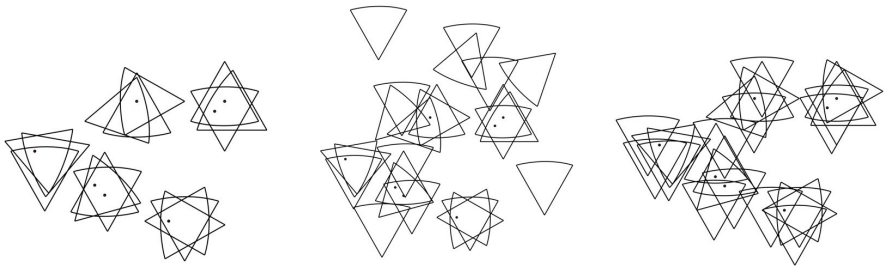
Table 6.3: Quality rating for algorithms.



(a) One camera per observation point.

(b) Two cameras per observation point.

(c) Two overlapping cameras per observation point with additional non-covering cameras.



(d) Three overlapping cameras per observation point.

(e) Three overlapping cameras per observation point with additional non-covering cameras.

(f) Five overlapping cameras per observation point.

Figure 6.3: Scenarios with different degrees of overlap and complexity.

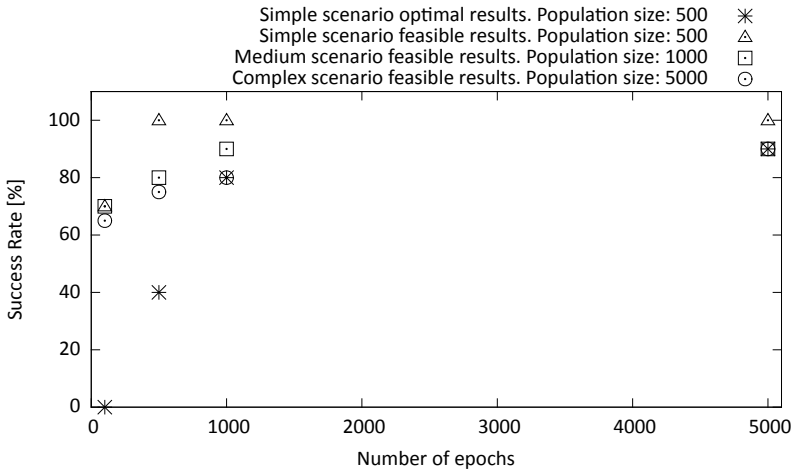


Figure 6.4: The relation between number of epochs and the rate of finding optimal and feasible results (success rate), respectively.

6.1.2 Number of Epochs

Typically, in evolutionary algorithms, the results improve with increasing number of epochs. It can easily be seen that an increased population size will also require increasing the number of epochs to achieve the same results at the same mutation rate.

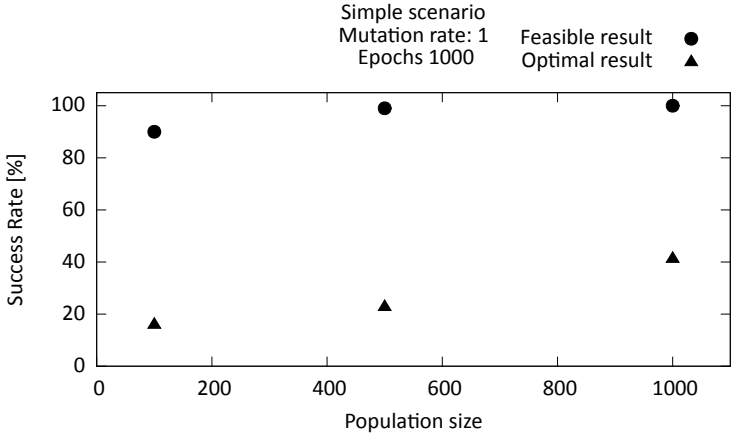
In figure 6.4 we show the change in number of feasible and optimal results with increased number of epochs. By choosing a suitable population size, a predictable rate of feasible results can be achieved. Depending on the complexity of the task, a larger number of epochs may be required.

6.1.3 Population Size

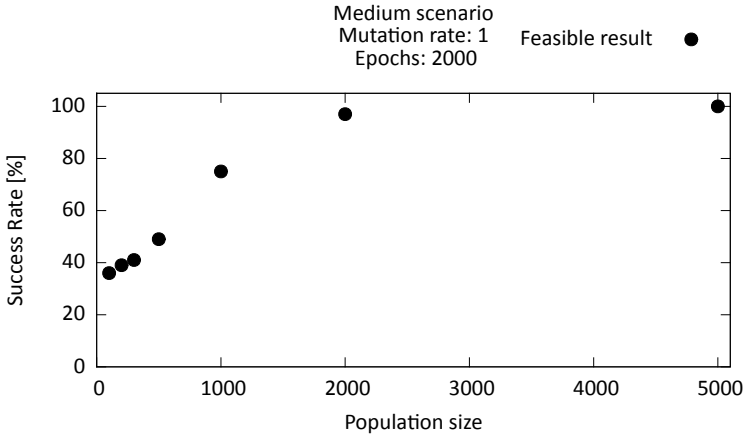
The population size represents the number of different permutations present at a certain point in time. A larger population size increases the probability that the population contains good individuals.

For our algorithm, it is necessary to increase the population size with increasing complexity of the scenario. As our results show, a population size of 5000 individuals is sufficient to achieve good results even for the complex scenario. For less complex scenarios, population sizes between 100 and 1000 are sufficient.

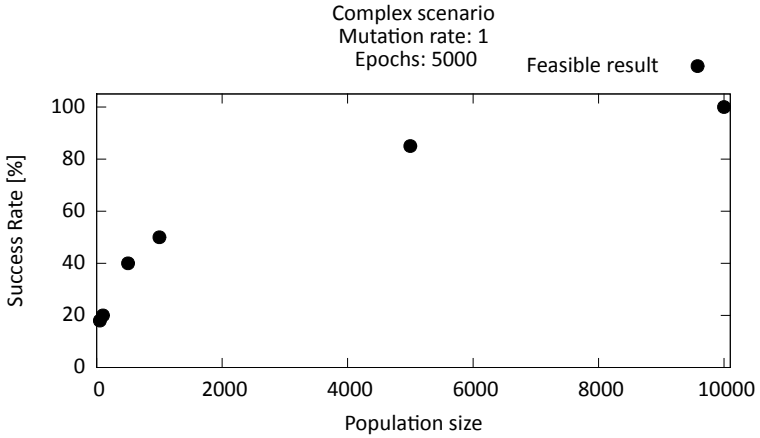
Figure 6.5 shows the impact of larger population sizes on the resulting rate of finding feasible solutions. It can be seen that the solution quality of complex scenarios can be improved by increasing the population size.



(a) Simple scenario.



(b) Medium scenario.



(c) Complex scenario.

Figure 6.5: Relation between population size and the rate of finding feasible results (success rate) for a) Simple b) Medium and c) Complex scenario.

6.1.4 Mutation Rate

Since the mutation rate determines how many chromosomes are altered per epoch, it greatly influences the number of epochs needed to find good results.

Figure 6.6 shows the influence of mutation rate on the achieved rate of feasible solutions. It can be seen that larger mutation rates not necessarily yield higher success rates. Thus, the mutation rate must be chosen carefully. Selecting the right mutation rate depends on population size and chromosome complexity. With a too high mutation rate, the algorithm might miss good solutions, if it is too low, the algorithm will require more epocs to achieve feasible results.

6.1.5 Algorithm Runtime

Evolutionary algorithms typically have a very large search space which causes long runtimes. We show that our algorithm has a linear runtime w.r.t. population size (Figure 6.7a), number of epochs (figure 6.7b) and mutation rate (figure 6.7c). Note that the runtime values shown are independent of the scenario complexity, i.e., to run 1000 epochs at 0.5 mutation rate and population size 1000 takes the same amount of time for the complex and the simple scenario, respectively.

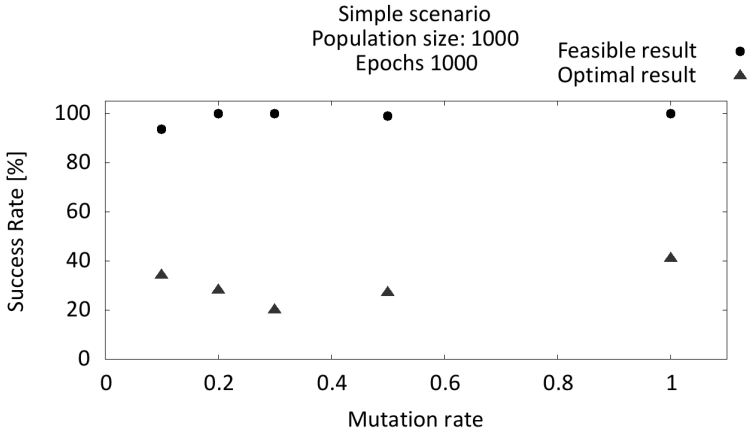
We performed the tests on a standard PC equipped with an Intel Core2Duo processor with 2.5 GHz. For each scenario we ran at least 1500 test runs at different combinations of mutation rate and population size and took dumps of the algorithm state at certain epochs.

Runtimes for scenarios with increasing degree of FOV overlap are shown in figure 6.8. It can be seen that an increasing number of solutions has a small impact on the runtime (whereas this also means that feasible solutions might be found earlier). Increasing the scenario complexity by adding non-covering cameras however, has almost no impact on the runtime.

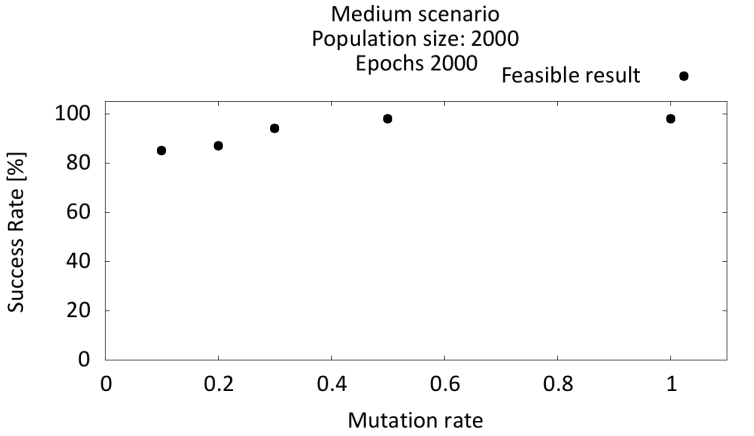
6.1.6 Surveillance Quality

By assigning quality ratings to algorithms, we can explore the tradeoff between surveillance quality and resource utilization. Figure 6.9a shows the Pareto front for the scenario of medium complexity for an elite size of 20 (i.e., we choose 20 non-dominated solutions from the Pareto front). Energyscore is calculated by $1/e_n$ where e_n is total energy consumption in the network. Since the optimization algorithm always maximizes values in the decision vector but lower energy values are better, we perform this invert operation. Thus, a higher value in energyscore means a lower energy consumption.

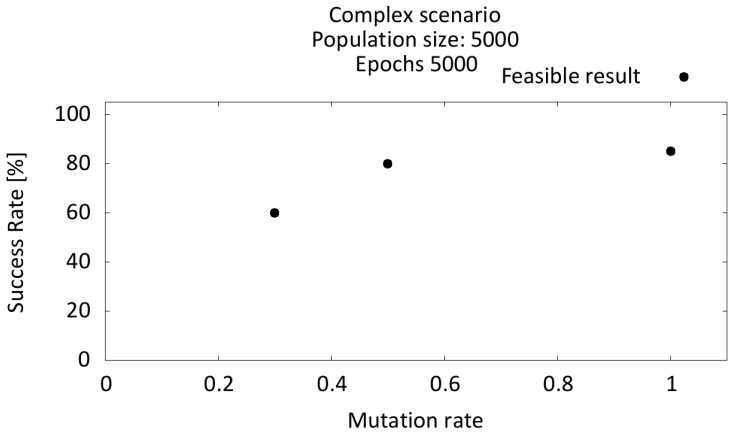
Figure 6.9b shows an example Pareto front for the scenario with five overlapping cameras per observation point. We used an elite size of 100 for this experiment. This shows that there is a large number of possible solutions that our algorithm is able to find. All those solutions must be regarded as



(a) Simple scenario.

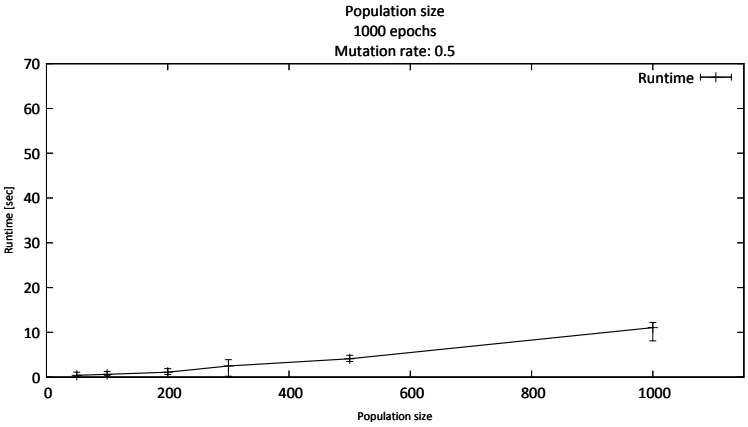


(b) Medium scenario.

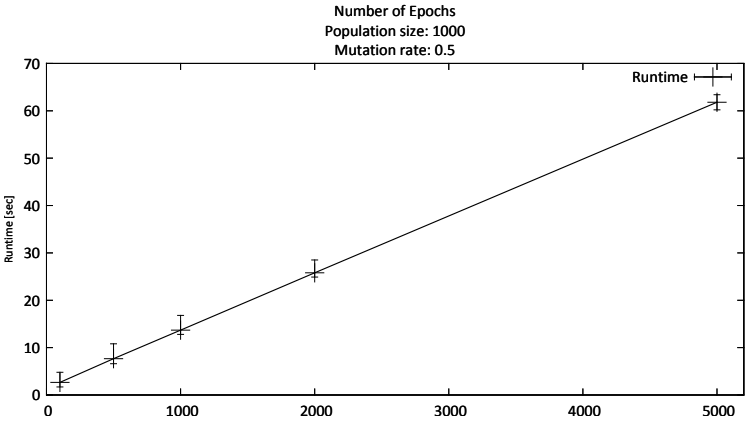


(c) Complex scenario.

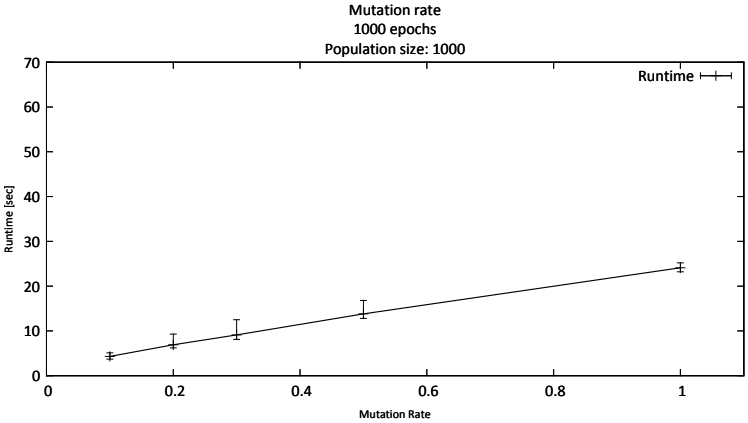
Figure 6.6: The relation between mutation rate and the rate of finding feasible results (success rate) for a) Simple b) Medium and c) Complex scenario.



(a) Population size. 1000 epochs. Mutation rate: 0.5.



(b) Number of Epochs. Population size: 1000. Mutation rate: 0.5.



(c) Mutation Rate. 1000 epochs. Population size: 1000

Figure 6.7: Runtime with respect to population size, number of epochs and mutation rate.

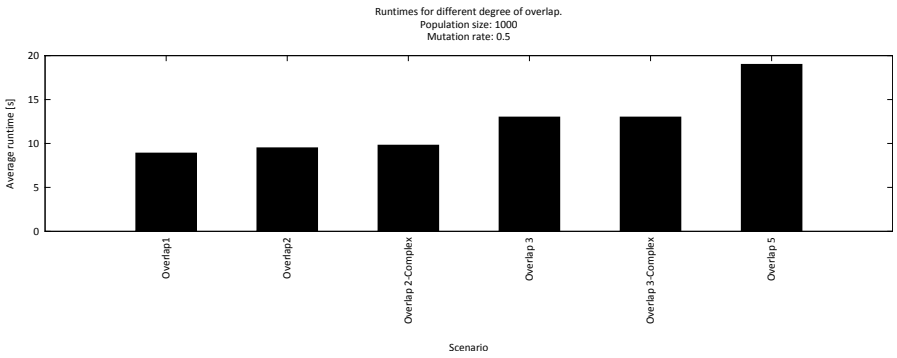


Figure 6.8: The runtimes for our overlap scenarios. "Overlap1" e.g., refers to the scenario where each observation point is covered by one camera aso.

Node	Tasks
1	SingleGaussian
2	FrameDoublediff
3	FrameDoublediff, Blobfinder, Kalman
4	off
5	FrameDoublediff, Blobfinder

Table 6.4: The result of the task allocation for the simple scenario.

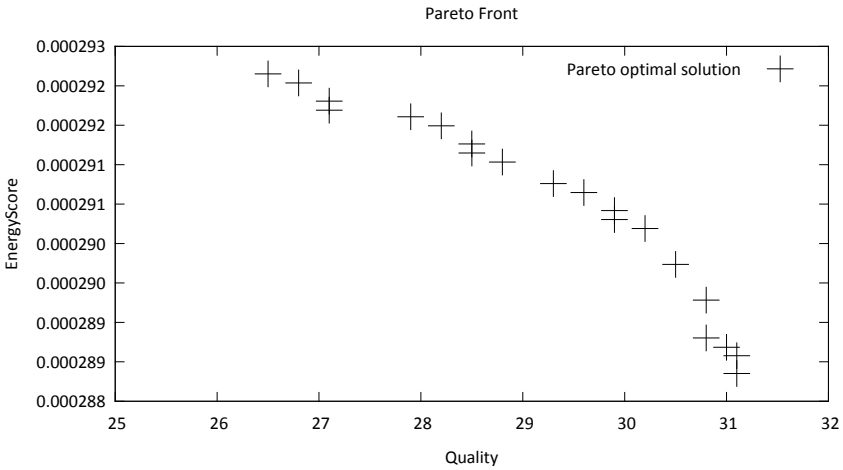
equally good tradeoffs between quality and resource usage. From those feasible solution, one has to be selected. This may be done according to a predefined weighting of quality versus resource usage or by any other metric or selection function.

6.1.7 Measurements of Resource Consumption

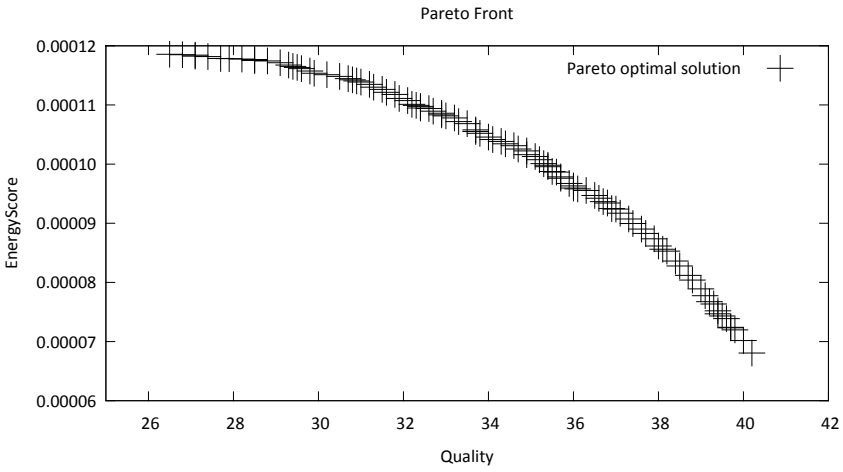
The result of our evolutionary algorithm is a feasible camera configuration and a task allocation along with a prediction of the resource demand. To evaluate the accuracy of the resource prediction, we experimentally test the resource consumption of the assigned tasks on a real platform. Based on the SLR hardware platform we use in our tests (see section 6), we first construct the mapping $\tilde{r}$ from measuring algorithm performance on this hardware. We run the algorithms with videos of different resolutions as inputs while measuring resource and energy usage.

We can then use $\tilde{r}$ as a lookup table in the algorithm to predict the resource demand of a certain configuration. We implement the application according to the task assignment in table 6.4 and measure the CPU load and power usage. For these tests, the application reads images from a video file of the calculated resolution and executes the assigned tasks.

We use Atom-based embedded boards as target platforms. These are



(a) Medium complexity scenario. Elite size: 20.



(b) Scenario with five cameras per observation point. Elite size: 100.

Figure 6.9: Pareto fronts for a) medium (PTZ) scenario and b) scenario with 5 overlapping cameras. Global minimum energy usage is used as resource-related optimization goal.

Node	CPU_p[%]	$Power_p$[W]	CPU_m[%]	$Power_m$[W]
1	4.32	0.09	3.6	0.1
2	0.34	0.01	0.21	0.01
3	17.55	0.35	16.2	0.3
5	12.28	0.25	11.6	0.2

Table 6.5: The predicted and measured resource usage for nodes in the simple scenario. CPU_p and $Power_p$ are predicted values, CPU_m and $Power_m$ are measured values.

also used in the SLR cameras. We test all algorithms on pITX-SP 1.6 plus board manufactured by Kontron[3].

As it can be seen from table 6.5, our predictions match with the measured results. The small difference between predicted and measured result is caused by the operating system which cannot be completely cut out in tests.

6.1.8 Integration of PTZ Reconfiguration

In PTZ scenarios we want *i)* to find feasible configurations and task allocations and *ii)* to select the subset of sensors which is required to cover the observation points. The sensors which are not required, can then be commanded to basic coverage while the other sensors can focus on detection and tracking at the observation points.

PTZ optimization

To test the automatic configuration of the pan, tilt and zoom parameters of the proposed PTZ network a map representing a certain area has been selected. On such a map eight different cameras have been deployed to cover the entire environment. In figure 6.10 a representation of the testbed area together with the deployment configuration of the PTZ network is presented.

Initially, a camera configuration has been achieved by running the PTZ reconfiguration on a homogeneous activity map (e.g., each cell of the map has the same activity density). Then, ten different trajectory clusters have been defined. These activity maps are later used to define observation points.

The PTZ cameras are reconfigured as described in [19]. Based on the generated activity maps, the cameras are oriented in a way, that the coverage on the area is maximized. This is done by employing the expectation-maximization approach described in [67].

[3]http://www.kontron.com

Figure 6.10: Deployment of the cameras (black circles) on the monitored environment. Each camera is placed at height 14m.

Sensor selection and resource allocation

Taking the result of the PTZ optimization as input, we run the sensor selection and resource allocation optimization. In the areas of high activity, we place observation points requiring object detection or object tracking. The resulting input for algorithm is shown in figure 6.11.

Table 6.6 shows the results for the sensor selection and sensor configuration. Table 6.7 shows a resulting task allocation. The resulting Pareto front for the task allocation is shown in figure 6.9a).

Sensor	*res*	*fps*	*activitiy*
1	SQCIF	4	object detection
2	VGA	18	object tracking
4	QVGA	4	object detection
7	SQCIF	12.5	object tracking
8	QCIF	2	object detection

Table 6.6: The result for sensor selection and sensor configuration. Sensors $3, 5, 6$ are switched off.

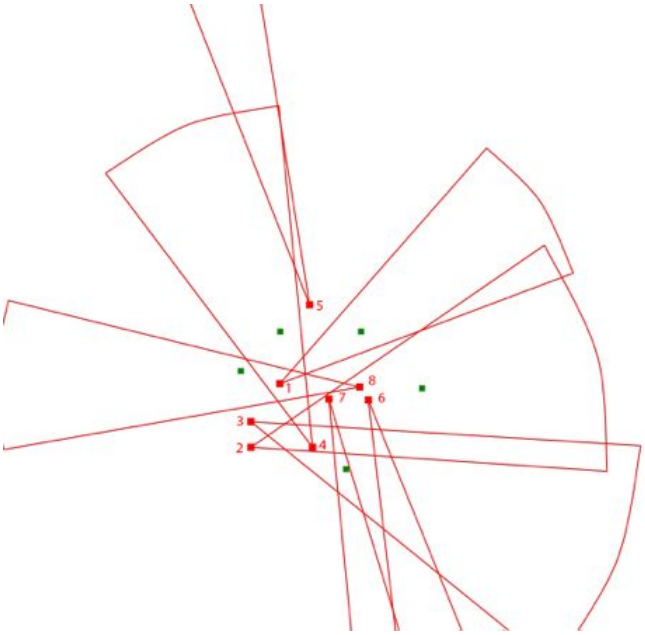


Figure 6.11: Input for the sensor selection and resource allocation optimizer.

Sensor	Tasks	CPU [%]	Mem. [MB]	Power [W]
1	FDD, BF	0.24	0.77	0.005
2	FDD, BF, K	70.22	18	1.4
4	FDD, BF	1.65	1.49	0.03
7	FDD, BF, K	2.03	0.73	0.04
8	FDD, BF	1.01	1.49	0.02

Table 6.7: The resulting task allocation. FDD: Frame Double Diff. BF: Blobfinder. K: Kalman. Sensors $3, 5, 6$ are off.

6.2 Distributed Algorithm for Low Dynamic Environments

In the following sections (adapted from [21]) we evaluate the distributed algorithm for reconfiguration in environments with low dynamics. We use simulation as well as real environments.

To evaluate the distributed reconfiguration algorithm, we first regard the theoretical boundaries of its communication complexity. After that, we show results from practical evaluations.

6.2.1 Communication Complexity

In this section, we show a theoretical evaluation of the communication complexity of the presented algorithm. To assess the communication complexity of our algorithm in terms of best and worst cases, we model the communication in a slotted fashion. We assume that only one node can transmit its message at a given time and that all other nodes overhear this message and that the nodes send messages in random order.

For this evaluation, we define the node neighborhood as the joint observation of a certain target by multiple cameras, i.e., it is the set of all cameras which have this target in their FOV. Using the function

$$c(s,t) = \begin{cases} 1 & \text{if } t \text{ is in the field of view of } s. \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

we define the neighborhood of cameras wrt. an observation point t as

$$N_t = \{s \in S | c(s,t) = 1\} \tag{6.2}$$

Note, that we define best and worst case communication complexity for a single observation point only. To calculate the respective numbers for a scenario consisting of multiple observation points, the best and worst cases for each of those points must be summed up.

Best Case

In the best case, the node which initially introduces a new target also has the best solution for it. In this case, it broadcasts its descriptor and all neighboring nodes reply to confirm. This corresponds to a best case communication complexity of $|N_t|$ messages for a single observation point t where $|N_t|$ is the number of cameras in the neighborhood wrt. this target point. This case is visualized in figure 6.12 with a neighborhood of size four.

	Step			
Node	0	1	2	3
A	d_A			
B				d_A
C			d_A	
D		d_A		

Figure 6.12: The best case communication complexity. Node A has already initially the best solution. It broadcasts the corresponding descriptor and receives a confirmation from each neighbor.

Worst Case

The worst case arises if the node which initially defines a new target has the least optimal solution for it and if the subsequent communication is performed in reverse order of solution optimality (i.e., nodes with the best solutions act last). From the set of nodes $\tilde{N}$ which have not yet transmitted their solution, always the node $n_i \in \tilde{N}$ for which the condition

$$\forall n_j \in \tilde{N}, j \neq i : d_{n_i} < d_{n_j}$$

holds, will transmit its solution (where the expression $d_1 < d_2$ means that d_1 is a worse solution than d_2).

We assume that the first nodes to confirm the slightly better descriptor are only those which have a worse solution. This means, that for the least optimal result, one message is sent (by this node itself), for the next better result, two messages are sent (one by the node that broadcasts this solution and one by the node with the worse solution as confirmation), and so on. In this constellation we reach the worst case communication complexity of

$$\frac{|N_t| \cdot (|N_t| + 1)}{2}$$

A worst case for a four nodes scenario is shown in figure 6.13. Assume that solutions are better according to the alphabetical order of the node IDs. Thus, Node D has the best solution, C has the second best, B the third best and A has the worst solution. However, it is node A which initially defines the observation point and transmits an initial descriptor. Node B replies with the next best solution which is then confirmed by A before C transmits its solution. It can easily be seen that any other transmission order would result in a lower number of required messages. If, for example, D transmits its solution immediately after the initial descriptor of A, the total number of messages required is 5.

	Step									
	0	1	2	3	4	5	6	7	8	9
A	d_A		d_B			d_C				d_D
B		d_B			d_C				d_D	
C				d_C				d_D		
D							d_D			

Figure 6.13: The worst case communication complexity. A defines the observation point but has the most expensive solution for it. D has the best and C the second-best solution.

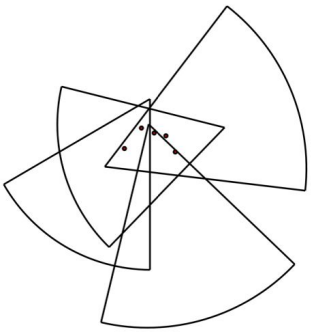


Figure 6.14: The camera and observation node placement in the practical scenario

6.2.2 Practical Evaluation

To give a thorough evaluation of the distributed algorithm for low dynamics reconfiguration, we conduct a series of tests with multiple scenarios. In both a real implementation and in simulations we evaluate the properties of the presented algorithm. First, we use the central algorithm presented in 4.2 as a benchmark for the distributed approach in order to assess in which circumstances the distributed version has deviating results. Second, we also evaluate the number of messages the algorithm needs to achieve its results and we show that this number is within the theoretical borders presented above. Note, that since a descriptor easily fits into a single message in almost every wireless communication technology, the number of messages is equal to the number of descriptors exchanged during operation. Third, we show the behavior of the distributed algorithm in scenarios with message loss.

We show that the distributed algorithm has a short runtime (measured in the number of messages exchanged to achieve a result), is reliable even in scenarios with message loss as high as 15% and scales linearly in terms of messages needed per observation point.

In section 6.1.7 the appropriateness of our resource models has been demonstrated. We have shown that the predictions of our energy and resource usage models closely match the tests on real hardware. To achieve

results for large scenarios, we rely on simulations. However, we first show the results from a real deployment to show that results from simulation and real deployments correspond.

Realworld Scenarios

We first show an evaluation of the algorithm in a real scenario built at the Alpen-Adria Universitt Klagenfurt campus. We use this scenario to validate the models used in the later simulation.

To show the algorithm in a practical application, we define a scenario with four Pandaboard-based[4] embedded cameras equipped with an Cortex A9-based OMAP4 processor. The cameras cover an area of approx. $1500m^2$. They observe five, partially shared observation points. The scenario is shown in figure 6.14. The observation points require either simple background subtraction, object detection or tracking. The corresponding task implementations are realized by using standard OpenCV[5] algorithm implementations (BgStatModel, BlobDetector, TemplateTracking). Cameras start the image processing procedures according to the observation points they are responsible for. We measure the time until a stable solution is reached. We further compare the solution which the algorithm achieves in our real scenario to the results of our simulation with the same scenario parameters. We do this to show that our simulation adequately corresponds to real deployments.

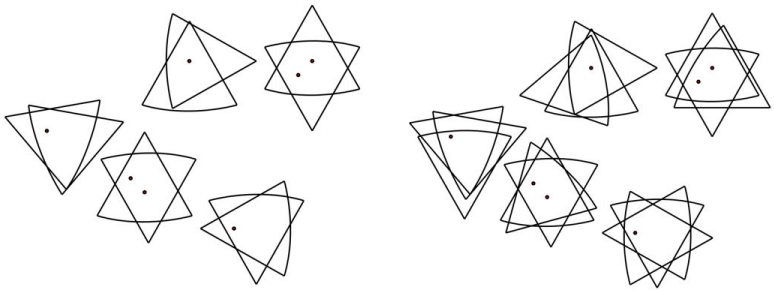
Simulated scenarios

We have defined seven scenarios that pose different challenges to the algorithm. We choose typical deployment scenarios for VSN with i) separated clusters with overlapping FOV that may arise in applications with surveillance of separate rooms in buildings or intersections of streets as well as ii) connected fields of view that are typical when a continuous surveillance should be achieved like pathways or large halls.

The first two scenarios—which have separated clusters—are used to show the behavior of the algorithm in low-complexity settings. Both scenarios have the same observation points but scenario b has more sensors covering those points. In scenario a 10 sensors are deployed while 15 sensors cover the observation points in scenario b. Scenarios a and b are shown in figure 6.15.

The other two scenarios have no clear separation of clusters and are thus more complex to solve. Scenario d contains additional sensors and observation points to scenario c but has additional sensors and points added which leads to more complexity. Every node in scenario d transitively shares its field of view with every other sensor. In total, scenario c contains 9 sensors

[4]http://www.pandaboard.org
[5]http://opencv.willowgarage.com



(a) Two cameras per observation point.

(b) Three overlapping cameras per observation point.

Figure 6.15: The low complexity scenarios a) and b). They share the same observation points, but scenario b) has one additional camera per cluster.

and 9 observation points while scenario d has 15 sensors and observation points respectively. The complex scenarios are shown in figure 6.16.

Scenario	Best Case	Worst Case	Average Overlap
real	14	27	2.8
a	14	21	2
b	21	42	3
c	21	36	2.33
d	38	70	2.53

Table 6.8: Best and worst case communication complexity along with the average number of FOV overlap per observation point for scenarios a - d.

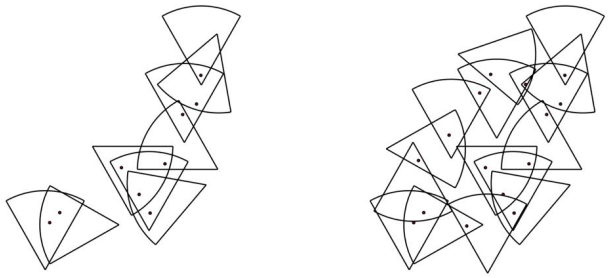
In all scenarios we use the resource usage prediction of the central algorithm as a reference solution for the distributed algorithm. We measure the performance of the algorithm in terms of deviation from the targeted results (the higher the deviation, the more resources will the solution demand during runtime). We also measure the number of messages the algorithm needs to achieve this result.

The experiments are performed in two steps:

1. By disabling the periodic re-evaluation, we can estimate the quality of results achieved when the algorithm terminates after all points have been covered.

2. By enabling the re-evaluation for 100 messages, we can see the improvement made by this mechanism. In all of our test cases we show, that this improves the solution quality (except in cases where the optimal result is already reached initially).

Additionally, we perform the same evaluation again for scenarios c and



(a) Medium complex scenario with interdependencies

(b) High complex scenario with circular cluster interdependencies.

Figure 6.16: Two scenarios with more complexity and interdependencies c) and d).

d with message loss rates of 5% and 15%. Again, we first look at the initial result and then compare them to the result after a re-evaluation phase of 100 messages.

Large-scale Networks

Additional to the four fixed test scenarios we evaluate the behavior of the proposed approach with three very large networks. A randomly generated scenario of 100 sensors and 25 observation points is used. By adding more observation points we derive two further scenarios containing 33 and 50 observation points respectively.

Targets	Best Case	Worst Case	Average Overlap
25	48	77	1.92
33	68	116	2.06
50	102	169	2.04

Table 6.9: Best and worst case communication complexity along with the average number of FOV overlap per observation point for the complex scenarios with 25, 33 and 50 observation points.

6.2.3 Evaluation Results

We first show results of our realworld deployment and then use simulations to test our algorithm in larger and more complex scenarios.

	Measured result	Simulation result
Deviation	2.45%	-
Time	12s	-
Messages *real*	26	21
CPU load node 1	95.3%	90.6%

Table 6.10: Results of the realworld deployment compared to the simulated results.

Realworld Results

In table 6.10 we show the results from our practical test. It can be seen that the simulation achieves results very similar to a real system both in terms of achieved results and required messages. The deviation of 2.45% means that the resources required by the solution of the distributed are 2.45% higher than the result predicted by the simulation. We also show how the actual processor load on node 1 corresponds to the predicted load. The difference here is caused by additional I/O operations performed by the application which are not reflected in the simulation's models.

Simulation Results

We compare the distributed and the central algorithm. This is important to assess the solution quality it will deliver at runtime. The central algorithm needs information about all nodes as input, and thus it is not affected by strongly connected fields of view. Opposed to that, each node running the distributed algorithm needs to find a solution only using its local knowledge and the limited amount of information it receives by the descriptors of other nodes.

In our simulations we perform 500 simulation runs per scenario and show the average, minimum and maximum deviation from the reference result of the central algorithm. We also present the average, minimum and maximum number of messages needed to initially cover all points. We also show, how often the algorithm finds the optimal result initially and after an re-evaluation phase of 100 message. Note, that even if the optimal result is not found, the result is still valid but requires a higher amount of resources. Additionally we show the average, minimum and maximum deviation from the reference result after the re-evaluation.

The results of this test series are shown in table 6.11. It can be seen that the algorithm performs very well in the simple (a and b) and medium complex (c) scenarios. It is able to find the optimal result for scenario *a* already in the initial assignment phase. Also, the number of messages needed is very low making the algorithm very suitable for resource-limited networks. Note that the results for scenario *c* are better after the re-evaluation than in scenario *b*. This is based in the fact, that the average overlap in scenario

c is lower (as shown in table 6.8).

	Scenario			
	a	b	c	d
Deviation initially [%]				
Average	0	1.5	5.1	11.9
Min - Max	0 - 0	0 - 36.8	0 - 15.5	0 - 64.9
Number of messages				
Average	17.5	31.2	29.4	54.1
Min - Max	14 - 21	22 - 43	21 - 43	44 - 78
Optimal results in initial run [%]				
	100	45.1	16.2	4.5
Optimal results after +100 messages [%]				
	100	45.5	54.8	41.8
Deviation after +100 messages [%]				
Average	0	1.3	3.7	6.7
Min - Max	0 - 0	0 - 12.2	0 - 7.8	0 - 55.5

Table 6.11: Results of the distributed algorithm compared to the central EA with no message loss.

Note, that for the scenarios b, c, d in some cases more messages than in the worst case are needed. This is because the theoretical worst case is calculated by slotting the communication of the algorithm for simplicity reasons. This gives a good approximation for the most cases. In the real implementation however, communication is not slotted but we perform unsynchronized buffering (i.e. there are no common time slots for all nodes). There are rare cases, where the nodes' descriptor buffering times are suboptimal, slightly more messages than the worst case can be produced.

We can see that the algorithm finds the optimal result for scenario d initially in 4.5% of our test cases but is able to increase this rate after the re-evaluation to nearly 42%. Thus, it can be seen that the algorithm finds results fast and that the re-evaluation allows the improvement of the solution already after a small number of additional messages.

Impact of Message Loss

A distributed algorithm that may be used in networks with non-reliable message transport must be able to deal with message loss. We performed simulations with 5% and 15% message loss to evaluate our approach. We assume a broadcast medium where a message loss means that the current message containing one descriptor is lost.

The results for 5% message loss are shown in table 6.12. It can be seen that the results are still very close to the results with no loss. It can be

	Scenario	
	c	d
Deviation initially [%]		
Average	21.96	25
Min - Max	0 - 229.2	0 - 218.3
Number of messages		
Average	27.96	51.2
Min - Max	21 - 38	38 - 64
Optimal results in initial run [%]		
	16.8	2.2
Optimal results after +100 messages [%]		
	57.8	45.2
Deviation after +100 messages [%]		
Average	3.3	7.1
Min - Max	0 - 12.6	0 - 55.5

Table 6.12: Results of the distributed algorithm compared to the central EA with 5% message loss.

seen that in the average deviation from the optimal result increases and reaches a rather high deviation in some rare cases. Here, the added value of performing a continuous improvement after the initial assignments can be seen very clearly since the improvements made after additional 100 messages is very large.

As table 6.13 shows, the algorithm still works well in scenarios with message loss of 15%. It shows the same behavior also for this rate of message loss. Initial results show a larger deviation from the optimal result which is then greatly reduced by the runtime re-evaluation of assignments.

Large-scale and Complex Networks

To show the behavior of our approach in large-scale and very complex networks we have randomly generated a network of 100 sensors and populated it with 25, 33 and 50 observation points.

We show in table 6.14 that the algorithm yields results very close to the optimum and that the average number of messages needed per observation point remains stable.

	Scenario	
	c	d
Deviation initially [%]		
Average	43.7	56.7
Min - Max	0 - 275.4	0 - 285.0
Number of messages		
Average	25.8	50.0
Min - Max	21 - 37	38 - 67
Optimal results in initial run [%]		
	5.4	0.4
Optimal results after +100 messages [%]		
	35.7	36
Deviation after +100 messages [%]		
Average	3.8	9.2
Min - Max	0 - 12.6	0 - 55.5

Table 6.13: Results of the distributed algorithm compared to the central EA with 15% message loss.

# Observation points	Deviation [%]	# Msgs.	Msgs. / OP
25	0.36 - 0 - 1.5	63.5 - 57 - 70	2.54
33	0.3 - 0 - 1	93.5 - 81 - 110	2.83
50	0.73 - 0 - 2.6	138.9 - 117 - 152	2.73

Table 6.14: The results for 25, 33 and 50 observation points (first column). We show the deviation from the central algorithm's result (column two), the number of messages required (column three) and the number of messages per observation point (column four).

6.3 Hybrid Distributed Algorithm for High Dynamic Environments

In this section (adapted from [20]) we evaluate our approach by first showing the advantage in terms of energy demand of using our approach compared to always tracking with the highest possible quality. Second, we show that the reconfiguration can be used to free resources on cameras that would otherwise be unable to track an object. Finally, we compare our approach to a centralized approach in a larger scenario.

6.3.1 Scenarios

To show the basic mechanics in our approach, we take the scenario shown in figure 3.1(a) with four cameras with partially overlapping FOV. In the first scenario, we remove all observation points. The results shown below were obtained by means of simulation.

The moving objects in all scenarios require tracking at eight frames per second and with 14 pixels on target. These quality requirements must be met in order to achieve a feasible solution. Tracking at higher settings will result in improved quality but also higher resource demands. We take six snapshots (steps) of the scenario, calculate the benchmark results and compare them to the results of our approach (in-between steps we assume continuous tracking). Whenever an objects comes close to the edge of the FOV of a camera, the node will try to perform a handover.

We calculate the utility for the tracked objects according to Equation 4.6a and the predicted energy consumption in the total network (according to our platform and algorithm models) in each step.

6.3.2 Evaluation Results

Figure 6.17 shows that the reduction of consumed energy for our first scenario is apparent. While the total utility is lower in our approach, the object is still tracked at its required quality and the energy consumption is 45% lower.

In the second scenario, camera S_3 is covering two observation points with high quality requirements. If no dynamic reconfiguration is applied, the camera is unable to additionally track the object when it enters its field of view. Thus, the system loses track of the object which results in a zero utility. Figure 6.18 shows how the system adapts the task allocation from before and after step 5 in order to free resources on S_3.

The third scenario is more complex than the first two and defines ten sensors and eight fixed observation points. Over 10 steps, up to six objects move concurrently in the area of interest. The sensors are arranged to cover

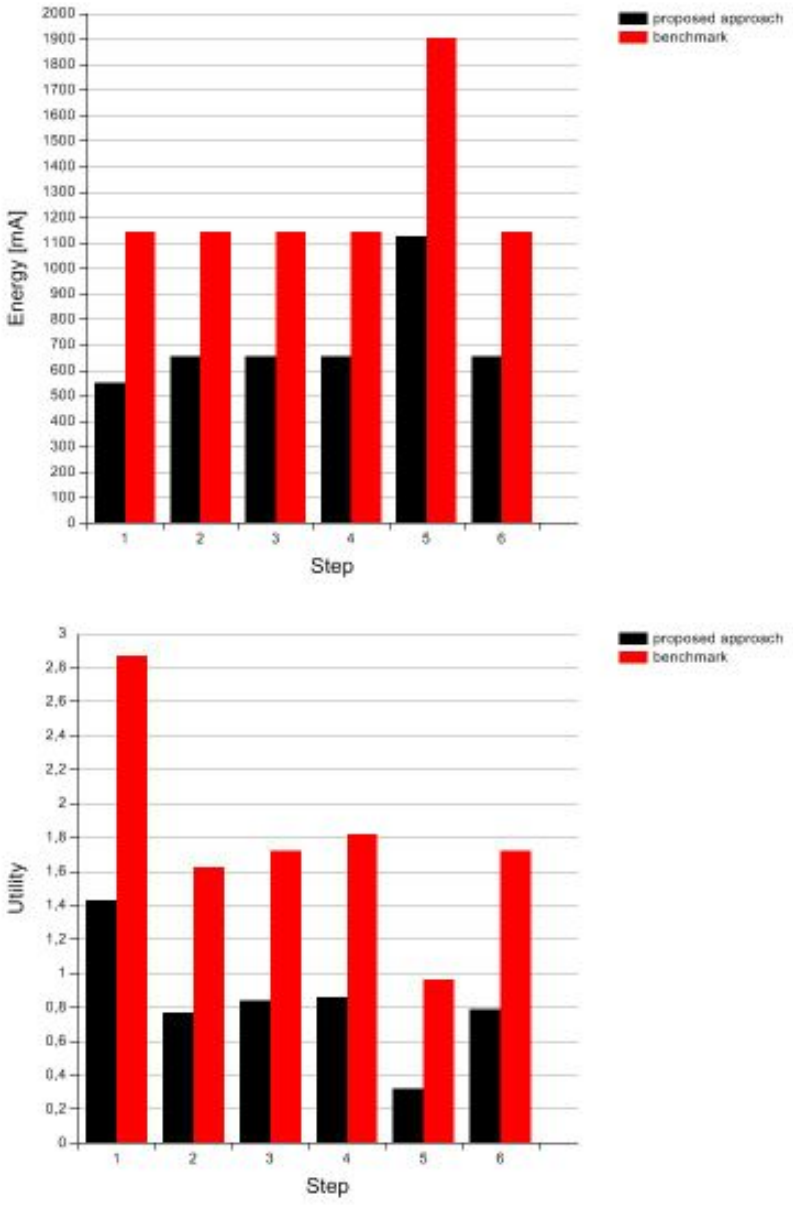
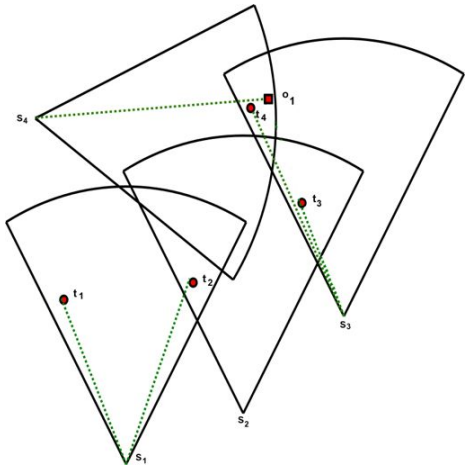
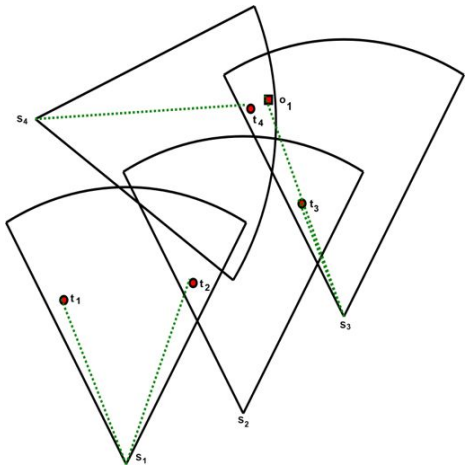


Figure 6.17: The energy consumption and utility in scenario one compared to the benchmark.



(a) State before handover.



(b) State after handover and reconfiguration.

Figure 6.18: The task allocation before and after the reconfiguration in step 5 in scenario 2. It can be seen that node s_3 has changed its configuration in order to track object o_1. To free resources it has handed over observation point $t4$ to node s_4.

an L-shaped area that could be e.g. a large corridor in a building or a pathway between buildings. The setup is shown in figure 6.19.

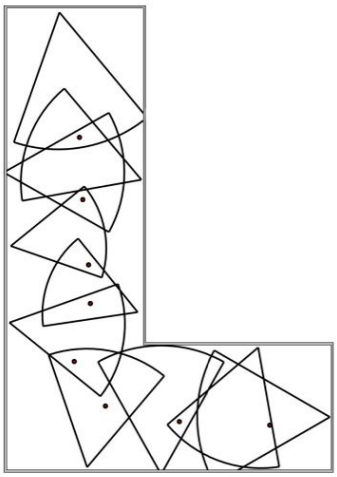
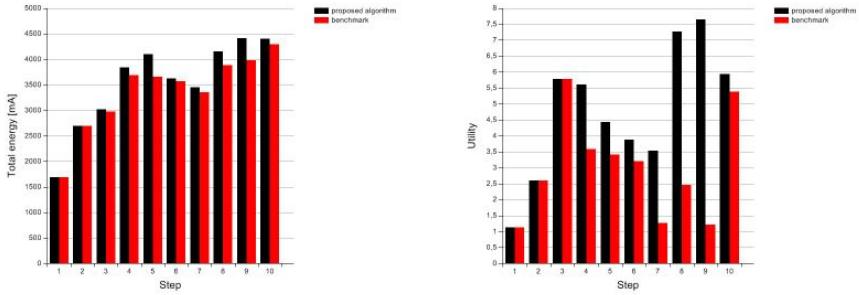


Figure 6.19: Scenario 3 defines ten sensors and eight fixed observation points in an L-shaped area of interest.

Again, the centralized algorithm presented in section 4.2 is used as benchmark. The optimal assignment calculated by the centralized algorithm for each step is compared to the results of our approach to that. Since the central algorithm cannot deal with moving objects (its input is $O \cup T$), we need to provide a static input for it. For each of the defined ten steps, we can use the central algorithm to calculate an optimal benchmark solution based on the static object placement in the respective step. Thus, the moving objects appear to the algorithm as observation points. While this does not give implications for where to hand over an object, the optimal solution for each step is a benchmark for the minimum energy consumption possible. The results of the proposed approach differ in several cases because an object has been handed over based on its movement direction.

In each step, we compare the total utility (the sum of the delivered utilities of all objects) and the total energy consumption in the network to the results of the central algorithm. Further, we show how many times the network performs a handover or reconfiguration operation.

Scenario three results are shown in figure 6.19. The utility is always higher or equal to the benchmark but also the energy demand is in some cases higher due to the required handover. Our results show that the average total utility for all tracked objects in the network was 202% higher than in the benchmark solution but that the average energy demand only increased by 4.2%. Thus, with the hybrid algorithm we have combined the ideas of reconfiguration with low dynamics with a handover mechanism for agile objects and still operate with nearly the same energy demand.



(a) The overall energy demand in the network in the proposed solution compared to the benchmark

(b) The total utility in the proposed approached compared to the benchmark solution.

Figure 6.20: Comparison between the proposed solution and the benchmark for scenario 3.

Table 6.15 shows the actions that have been taken in the individual steps by our algorithm. We show in which step it hands over an object and in which steps it needs to reconfigure observation points prior to handing over a moving object. It can be seen that several handover operations have been performed and in some cases also reconfiguration due to high object densities were necessary.

Step	**# Handovers**	**# Reconf.**
1	0	0
2	0	0
3	1	0
4	2	0
5	3	1
6	3	1
7	2	0
8	3	1
9	3	0
10	3	0

Table 6.15: The number of handovers and reconfigurations in each step.

6.4 Evaluation Results Summary

In this chapter we have performed an evaluation of the algorithms presented in section 4. We have shown for the central algorithm that it is able to approximate the Pareto front for the multi-objective optimization problem of

surveillance quality versus energy consumption. We have shown that the algorithm can flexibly be adapted to different scenarios in terms of population size and mutation rate.

In simple and complex scenarios, we have shown that the algorithm has a high reliability in finding feasible solutions. We have also shown that the predictions of the resource model closely correspond to real measurements. Further, we have shown how a PTZ reconfiguration can be integrated into the algorithm in order to deal with varying area coverage.

For environments with low dynamics we have evaluated the distributed algorithm which performs observation point reconfiguration. We can show that it achieves results which closely match the benchmark results of the central algorithm. We further show that in a real scenario, the algorithm achieves a result which corresponds to the simulation predictions and which is found fast enough to react to slow changes in the environment.

We have shown for our distributed algorithm for high dynamic environments that it achieves significant resource savings by performing reconfiguration of sensor settings. We have further shown that it prevents objects from being lost due to insufficient resources on the handover target node. Finally, we have shown that we can achieve large increases in surveillance quality at very low additional resources costs.

Chapter 7

Server Infrastructure Reconfiguration

To show that the evolutionary algorithm for resource-aware reconfiguration presented in this thesis is not only applicable to VSNs, we port this method into the field of large server infrastructures. In recent years, we see that web-based and mobile applications drastically increase in terms of the services offered and the number of users. Typically, such applications need a large server infrastructure as a backend. Such a backend consists of multiple load-balanced servers which process thousands of client requests per second.

Applications for mobile devices have become a fast growing market segment. The increasing number of smartphones which are GPS-enabled and have high-bandwidth internet connection is a driving factor for the development of location-based multi-user applications. Already today, massive multiplayer online games (MMOG) are very successful. Combined with the trend towards more mobility we expect that *mobile* massive multiplayer online games (MMMOG) will be an important part of the mobile application market in the coming years. Although the individual application scenario may be different, many of these applications will share some common properties. They are typically *client-server based*, *location-based* and enable *user interaction*.

Typically, server load balancing is done by replicating the processing instances to multiple servers and adding a load balancing layer to redirect requests to servers with free capacities. This however, requires the use of data synchronization between server instances. In the GeoBashing [18] server infrastructure described below, we have taken advantage of the geo-based nature of a mobile multiplayer game by assigning servers geographical responsibility areas and by load balancing based on those responsibilities. This reduces the data to be shared between servers and enables a more efficient load balancing. To find the optimal assignment of regions to servers, we adapt the evolutionary algorithm to perform reconfiguration on the server load balancing.

In the following sections (adapted from [18]) we describe the server architecture for geo-based load balancing and a use case in form of a mobile massively multiplayer online game. Finally, we look at an EA to optimize the load balancing based on resource-aware reconfiguration.

7.1 The Geobashing MMMOG Architecture

The server infrastructure basically consists of a web service layer and an application server (AS) layer. The web service layer is only a thin entry point to the application server layer. The application servers perform all game-specific tasks. Figure 7.1 shows the Geobashing architecture.

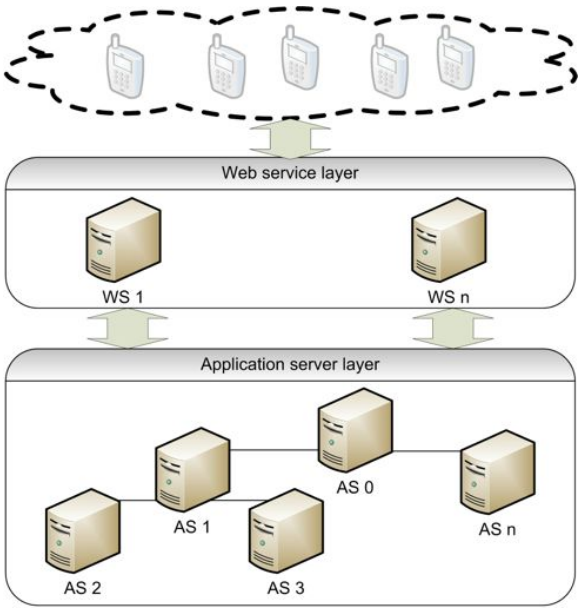


Figure 7.1: From a high level view the Geobashing architecture consists of a web service and an application server layer.

Web Service Layer

Since web services, or—more particularly—the web server they are running on, represent a major bottleneck in many web applications, their functionality is reduced to a minimum in our system. The main tasks of this component are user authorization and forwarding of requests to the application server. Due to this reduced functionality, a transparent load balancing of the web services can be realized easily, using any well–known load balancing mechanism (e.g. DNS-based load balancing). In order to support the application server load balancing described below, web services hold a mapping between users and responsible application server instances. This mapping is the only shared state that all web services have to access and is used to directly address the currently responsible application server for a user. If a mapping between a user and an application server cannot be found, the request is relayed to the root application server. To make these data available, a distributed hash table (DHT) is used.

Application Server Layer

The application nodes implement the game logic and manage the game state. We assume the game state to be a set of static and dynamic game objects, which have a certain position on a map. Since heavy load on these nodes has to be expected, a load balancing technique must be implemented. In our approach the load on the AS instances is balanced based on the user's current position which is part of every user request. The server topology consists of multiple application servers where every node is responsible for a certain geographical area (which we call the server's bounding box).

Since responsibility areas may be nested, we organize the application servers in a tree where the bounding boxes of all sub servers are contained in the root server's bounding box. To ensure a distinct mapping of user positions to servers, the bounding boxes of sub servers must not intersect (however, they may be fully contained in each other). Every server knows its sub server but not its superordinate node. The advantage of this server infrastructure is that a server only maintains the state of game objects that are within its responsibility area and that are not contained in a sub server's bounding box. The server hierarchy provides the possibility to deploy a server infrastructure that addresses areas with a higher player density. If such an area is identified, the load can be split by adding more servers to the particular sub tree.

Like in every other application there are also requests that simply query data. In the context of mobile games these requests often enable the client application to provide parts of the game in offline mode (e.g. transfer challenges to the client). In the presented architecture, requests which are not geo-related (e.g., a query for player character details),can be handled by any application node.

Game State

To support our architecture, we assume the game to have a lightweight game state that consists of only two object types: static and dynamic objects. Static objects are bound to a non-changing position on the map, whereas dynamic objects move on the map. Only static objects are stored in the persistent storage while dynamic objects are added on the fly.

Data Storage

The persistent storage is only accessed by the application nodes. The storage is used whenever static game objects need to be stored, modified or queried. Its main task is to provide the possibility to query static game objects based on a provided search area, e.g. whenever a new application server node is initialized.

Request Handling

Every application server processes a request in two phases (see figure 7.2). In the *first phase* a sub server that is responsible for the request (i.e. the user position is contained in the sub server's bounding box) is searched. In *phase two* the request is either forwarded (if a suitable sub server was found; the sub server then performs the same query again) or handled by the current server. After handling the request, the response is returned through the server tree. This server selection process ensures that a request is always handled by the server with the smallest bounding box containing the user position. Since this lookup is an expensive operation, the response contains the ID of the server that actually handled the request. Thus, on the next request, the server can be queried directly.

The mapping between a user and the responsible application server is stored in the previously mentioned DHT in the web service layer. Since users move while being logged in, it is possible that the mapping between a user and a server becomes invalid. In this case the application server responds with a redirect message (see figure 7.2), indicating that it is no longer responsible for the particular user. The calling web service then redirects the request to the root application node, a new responsible application node can be selected and the user-to-server mapping is updated.

A special type of request is the so called in-range request which queries game objects within a certain search area. In the application server infrastructure there is the possibility that this search area intersects with the responsibility areas of more than one server. In this case, the request is performed by the server that completely contains the search area. This server forwards the requests to all responsible sub servers and aggregates the data.

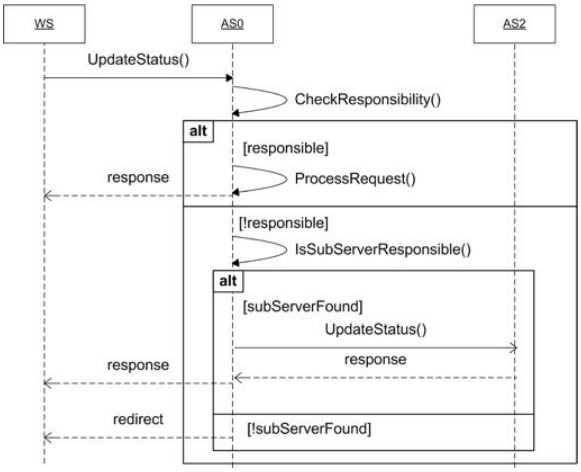


Figure 7.2: Selection of the responsible application server (simplified). WS is the calling web service.

Latency Reduction

The presented load balancing mechanism is applied to distribute the load to multiple application servers within a single data center. By extending the web service layer, the same strategy can be used to achieve a distribution to different data centers. This addresses the need to reduce the latency on the client. In such a scenario the server infrastructure is present at multiple sites where every site has a different responsibility area. For this to work, the web service layer has to have knowledge of the global distribution and bounding boxes of the different sites. The lightweight game state supports such a multi-site infrastructure by partitioning the static objects according to their positions, as all objects are contained in a well-defined bounding box.

Fault Tolerance

The combination of the responsibility areas of servers, the possibility to redirect a request and the lightweight game state have an additional advantage. Due to this, a fault tolerant application server infrastructure can easily be realized. The system supports the transfer of the game state from one server to another without the need for a complex synchronization mechanism. If an application node fails, the web service layer will no longer be able to connect to this particular node. In this case the request will be forwarded to the root application node. If—at any point—a server cannot be reached, this particular server will be deleted from the parent's sub server list. Thus, the whole server sub tree will no longer be reachable. A user who was previously managed by a server in this particular sub tree will be assigned to a new server. Thus, all dynamic objects of the game state are migrated on the fly. The static objects can be easily added to the game state of the server now responsible by simply querying the data storage providing the bounding box of the failed sub tree.

Implementation Details

To support fast development of client applications, the implementation of the web service layer solely relies on Restful Web services (based on HTTP). The web service and the application server are implemented in .NET. For the communication between the web service and the application servers or between different application servers .NET Remoting is used.

To provide an easy way to exchange and extend the functionality of the application server, it is realized using the plug-in framework presented in section 5.2.1. This enables game developers to use the Geobashing MMMOG architecture and build different game concepts on top of it. To support faster querying of position-related data at runtime, an R-tree [36] is used

to store the game objects at the application servers. We use the R-tree implementation of Sharpmap [84]. For the DHT we use Memcached [54].

The default communication pattern in our architecture is a client-side pull communication. Nevertheless, in some cases the server may want to explicitly notify the client of some event (to improve the game flow). For those cases we use a push communication channel based on SMS messages.

7.2 The Geobashing Game

As a use-case of our architecture we present the Geobashing game. Geobashing combines player interaction with mobility aspects, sports and role play elements. Players in this game can challenge other players with different tasks. By completing challenges, players earn experience points and virtual money. This part of Geobashing is called the *active part* (players have to actively choose to create or participate in a challenge).

In contrast to other games, Geobashing is meant to be played all day. Players can leave the application running in the background while carrying their phones. The application periodically sends the current GPS position to a server (this is called the *passive part*). The update frequency is controlled by the server which yields the possibility for server-controlled energy management based on e.g. player density. The server also evaluates if other players are nearby. Like in other role plays, a player can attack if nearby players are encountered.

7.2.1 Game World

The world of Geobashing is a virtual overlay to the real world. Every game element has a geographical position attached. A Geobashing player only needs to know the part of the virtual world that is immediately surrounding her current position.

7.2.2 Game Elements

A *character* is the virtual representation of the player and has the attributes *attack*, *defense*, *stealth*, *health*, a *level* and *experience points*, a purse for *money* and can carry *items*.

Items are static objects, i.e., they are bound to a certain position and do not move. Players can carry a certain number of items with them. They consume storage slots and can alter character attributes (e.g., the mighty sword of destruction, or the red baby-trike). They can be dropped or picked up by a player which enables *trading* between players.

Traps are similar to items, but additionally, they can be dropped and activated. If a player approaches a trap, a corresponding action is triggered (e.g., the player loses a specific amount of health points).

Money is needed for several tasks and is modeled as an item with a count.

Every player can define a certain position as *home base* for her character (e.g., her home). Within a fixed radius around the home base the player is not attackable. If a player is in this zone, she automatically restores health points. Additionally, she can exchange money and items with the home base storage or access the global Geobashing *item shop*. The home base can also be equipped with items which provide services to its owner or other players (e.g., a hospital: allows other players to restore health points for money).

7.2.3 Challenges

Players can participate in *Challenges*. The first time a player participates in a challenge a starting fee may be paid as a bet. A challenge consists of at least one position, has a goal and a reward. We have defined the following types of challenges so far.

A *Race Challenge* follows the "be the fastest" principle. A track of waypoints must be passed and the player gets ranked according to her best time. On expiration, the sum of all starting fees is split among the best participants. Participants may receive medals, items and/or experience points as a reward. The creator may place a bet, which gets added to the rewards and she receives experience points for every participating player beating the creator's reference time. If the participant does not pay the fee, she receives no monetary reward.

Another type is the *Paper Chase Challenge*. The idea is to give the character hints or let her solve riddles in order to get the next waypoint. Neither is the time relevant nor is a participation fee needed, but one can only participate once. A reward is optional. The creator is rewarded with experience points for every successful participant. A ranking is not needed for this type.

The *Shape Challenge* adopts the idea of GPS drawing[1]. A player who draws a given shape with the longest GPS track wins this challenge. Rewards and rankings are handled in the same way as in the race challenge.

For the most possible freedom players can create an *Open Challenge*. The creator only needs to declare a condition and a reward (e.g., "Pick me up at noon, drive me to the next bar, I might have a cool item to give away.").

The *Group Challenge* addresses groups of players. A group challenges is a composition of multiple single player challenges. Players form a group and find an approach to solve all challenges as fast as possible. All challenges must be successfully mastered by the group, whereas every player must participate in at least one challenge. The majority of the group must vote

[1] e.g. http://www.gpsdrawing.com

for the start of the group challenge (this starts the timer), it ends when all challenges have been attended at least once. The experience rewards are multiplied by a group factor and are distributed to the members, all other rewards are discarded. The best ranked players are awarded gold, silver and bronze *medals*.

7.2.4 Fight

Whenever two players are within a given range, they both are notified of the other (see screenshot in figure 7.3). This range depends on the stealth attribute of each player character. The player can then decide whether or not to attack.

Figure 7.3: A screenshot of Geobashing showing the players in range. The three icons at the top right indicate (left to right) that GPS is online, players are in range and that the player is logged in.

In a fight, the attacked player has the chance to escape. If, within a certain time frame, she can flee from the attacker (by overcomming a distance), the fight is aborted. Otherwise, if the attacker chases the attacked player and the time frame expires, a fight is started. The fight is turn-based and the result is based on the individual attack and defense attributes of the characters. The fight result is calculated at the server without player interaction. However, the attacker can cancel the fight at any time. This also means that the attacked player may try to persuade the attacker to abort the fight.

7.2.5 Player Interaction

Geobashing itself defines no communication or interaction channels between players. This means that there is e.g. no messaging system. We assume that in a game, where players play in the real world, they also should interact like in the real world. To trade, players have to bargain for good prices for

their items. To persuade an attacker to abort a fight, a player will have to be creative. We consider this a very essential part of the Geobashing game.

7.2.6 Interfaces

Geobashing provides a *mobile client* and a *web application*. The mobile client enables the player to participate in challenges. It also periodically reports the current position to the server and notifies the player of items or other players in range. Using the web application, a player can manage her home base, download challenges to the mobile client, view her statistics and challenge details.

7.3 Optimization of Server Task Assignment

While the server architecture described above can be used to realize a position-based load balancing, it is still unclear how the assignment of geographical responsibility areas to servers should be determined in order to achieve an optimal assignment of regions to servers. In this sections, a task assignment algorithm for servers is presented which works very similar to the central reconfiguration algorithm for VSNs.

7.3.1 Chromosome

The chromosome is modeled as a list of servers with assigned responsibility areas. As described above, those areas must not overlap but may be contained in each other. The chromosome does not know the player distribution in the area of interest.

Mutation

There are 7 actions possible in the mutation step.

- Add a region
- Remove a region
- Add a server with at least one region
- Enlarge a region
- Shrink a region
- Move a region
- Assign a region to a different server

In every action, a certain amount of randomness is applied. This concerns the amount of change in size or location and the selection of a server or a region to modify. For all these actions no concern is taken regarding the load on each server. This enables the coverage of the complete search space. The load is only calculated in the fitness assignment step. If a server contains no more regions, it is deleted from the server list.

Crossover

The crossover step takes the lists of servers and regions and merges them. Afterward resulting intersections of regions are removed.

7.3.2 Fitness Function

The fitness function evaluates a chromosome with respect to various parameters. First, basic metrics like expected server load and costs per hour are considered. Further, more elaborate measures like expected latency, number of servers not under full load and the minimum number of players on a server can be included.

For each server, the base load (i.e. the load generated by the application if no clients are connected) as well as the average expected load per 1000 players is used to predict the load on the system at a certain number of players.

7.3.3 Models

The models used to assess the quality of an assignment are derived from practical load tests done in our infrastructure[2]. A mix of real an simulated clients are used to test the server architecture under load.

Practical Evaluation for Model Building

To evaluate our architecture, we performed several tests with the Geobashing game. The Geobashing game is not very sensitive to high latencies, but to preserve a good user experience, latencies below five seconds have proven to be acceptable.

In a first experiment we test how latencies change if a greater load is posed on the server. To achieve this we simulate varying numbers of players that continuously perform the status update request. In this request the client application submits the current location, the server searches for surrounding players and game objects and returns this list to the client. The tests are performed using an Intel Core i7-920 based server with 8 GB of main memory running Debian with Mono 2.4.4. The web service (running on Mono XSP2) as well as the single AS run on the same machine. A

[2]This section is adapted from [18]

#Simulated Users	*Avg. client latency[ms]*	*Avg. Processing Time[ms]*
0	653	15.8
1,500	704	21
7,500	570	37

Table 7.1: The latency at the client and the processing time per request wrt. various user numbers.

Geobashing client prototype using the .NET Compact Framework is used as game client.

The time required to execute the request on cell phone clients (Nokia E71 running Redfivelabs Net60 runtime for .NET Compact Framework in the cellular network of A1 Austria) is measured. Additionally, we record the request duration at the server to be able to assess the latency of the mobile network. The results of this experiment are summarized in table 7.1.

The difference between the time measured at the clients and the actual processing time at the server shows that the latency at the client is mostly caused by the mobile network. In this experiment our server was located in Germany, the clients were running in Klagenfurt, Austria. Thus, by bringing the web service closer to the client, a reduction of latency can be achieved. This experiment also showed that the processing time per request at the server increased with the number of users. Since we did not simulate the full range of requests a typical user would perform, we expect a higher load with the same number of real clients. This in return, indicates the possible need for load balancing the AS at a number of concurrent users below 10,000.

In a second experiment we evaluate the maximum number of users that a single AS can handle. In a lab environment we split application server and web service to separate machines. The AS runs on an Intel Core2Duo 1.86 GHz with 2 GB main memory. All servers are connected using 100MBit/s Ethernet and run the Mono 2.6.1 LiveCD (based on OpenSuse) with Mono XSP2 as web server. Again, clients are simulated to generate the necessary load.

The experiments show that the AS can handle more than 500 requests per second without significant increase of latency. This roughly corresponds to 5,000 to 8,000 (simulated) users. Additionally, this experiment shows that the web service layer has a higher risk of being a bottleneck.

Finally, we evaluate the presented load balancing mechanism. We focus on the costs of redirects and forwards. In both cases the request cannot be posed directly to the AS but—starting from the root AS—the responsible server must be found in the application server tree. In case of a forward, the responsible server cannot not be found in the distributed hash table (DHT). In case of a redirect, the server is found but it has denied service because the user had left the server's responsibility area. Thus, a redirect is preceded by

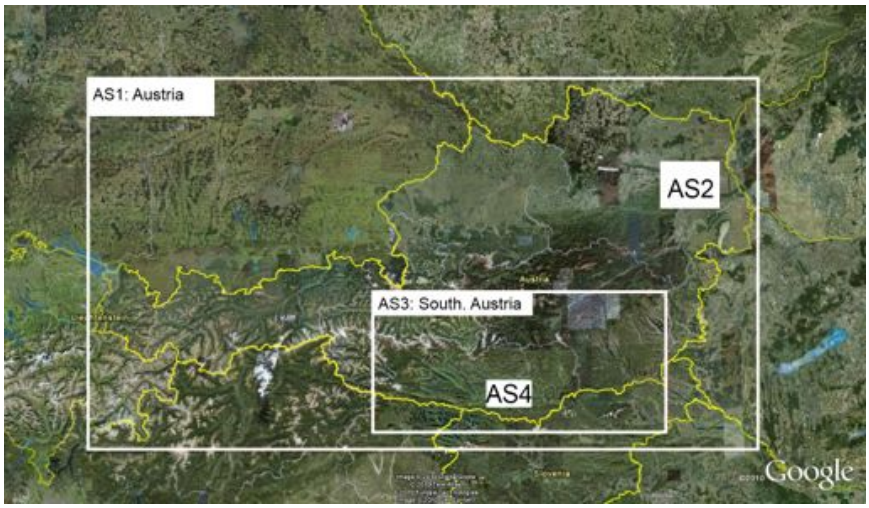


Figure 7.4: A map showing the geographical responsibility areas of AS1-4

one unsuccessful request. If this cost is very high, players migrating from one AS to a neighboring could generate a high overhead. In addition, we test the performance of in-range requests that cannot be handled by a single server (as described in section 7.1).

Using the same lab setup as in our second experiment, we look at the following setup of application servers: AS1 is the root AS responsible for the Austria region. AS2 is responsible for the players in Vienna. AS3 covers southern Austria (Styria and Carinthia) and AS4 covers Klagenfurt and the Wörthersee region. Thus, AS2 and AS3 are contained in AS1 and AS4 is contained in AS3. Figure 7.4 shows a map of the responsibilities.

# Requests	*Avg[ms]*	*Max[ms]*	*Min[ms]*
0	3.05	2189	1.3
1	6.13	4124.7	2.7
2	7.14	2351.2	4.1

Table 7.2: The duration to find the responsible AS in case of forwarding.

Table 7.2 shows the time necessary to find the responsible AS in the application server tree in case the player was not present in the DHT. The maximum number of redirects within the application server tree is two in our setup (for AS4; redirect from AS1 to AS3 to AS4). Zero requests means that the root application server (AS1) was responsible for the player. Table 7.3 shows the search time in the application server tree in case of a redirect. Zero requests means that after the initially failed request to a sub server (AS2 or AS3), the application server was responsible for the player.

The results show that forward and redirect differ mainly in the time for

# Requests	*Avg[ms]*	*Max[ms]*	*Min[ms]*
0	5.90	1851.5	3.55
1	8.12	1015.25	4.98
2	10.92	509.45	6.31

Table 7.3: The duration to find the responsible AS in case of redirecting.

the single initial request of the redirect. A direct query to an AS takes about two to three milliseconds in this setup and the search for a suitable server takes linear time. By keeping the Player-to-AS mapping in the DHT, in most cases the responsible AS will be queried directly.

To test the performance of the multi-server in-range requests we simulate users in the border region of AS4. To find all neighboring players, AS3 had to merge its own results with the results of AS4.

The average single-server request at AS3 took 0.26ms while the average time to perform the multi-server request was 3.5ms. The additional request to AS4 again took approximately 3ms. To save time, a multi-server request with a higher number of servers involved is performed in parallel. In the worst case, four servers must be queried to perform this request. This shows that the relative cost of a multi-server request is high but judging from the absolute value. However, this will not influence the game flow. Furthermore, the radius in which Geobashing searches for neighboring players is only 200m, i.e. this case is unlikely to occur often in the Geobashing game.

Cost Model

A second dimension in the task assignment optimization is the financial costs caused by employing a certain number of servers. Of course, this number is desired to be low. In a modern infrastructure it is common to rent the required infrastructure at a cloud hosting provider. Therefore we use the prices to rent the required number of servers in the cloud as basis for our model calculation. We take the prices of the Microsoft Azure Service as basis. From this we derive the costs using the expected number of servers and expected load on the servers to determine the total running costs for a certain solution.

7.4 Evaluation of Server Task Assignment

In order to test and evaluate the algorithm a small, a medium and a large scenario are used. The scenarios represent a certain player density in a geographic region. In certain parts of this region, the density is higher to simulate cities. In the remaining parts, the density is lower.

In a small scenario, we process an area of 250×250 kilometers with approx. 22700 users. The region of interest with the player densities are shown in figure 7.5.

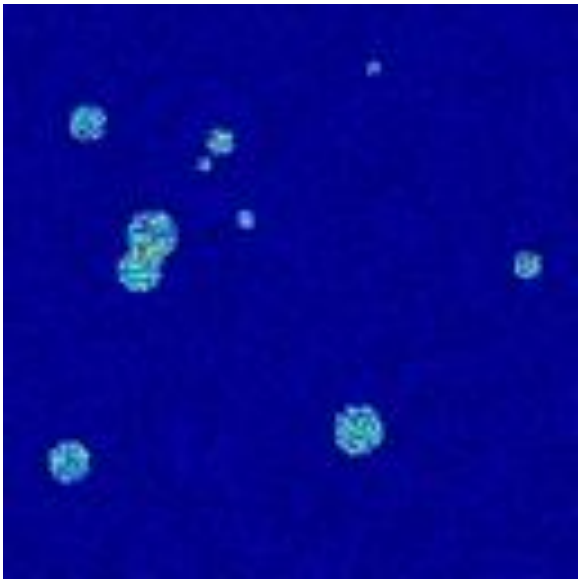


Figure 7.5: The region of interest of the small scenario. The image represents a map of the area, the colors indicate the player density. Brighter colors mean more players in the region. The bright spots indicate e.g. cities with more players.

The results for this scenario are shown in figure 7.6. The algorithm proposes several solutions with different load/cost ratings. The results are based on the estimated maximum load on the employed servers (all other servers have a lower load and thus lower latency) as well as the estimated cost for running this setup in the Microsoft Azure[3] cloud (in $ cents per hour). We calculate the cost based on the prices for small virtual machine instances (0.09$ per computing hour as of 2013-24-09).

The medium scenario contains approx. 91000 users and has a size of 500×500 kilometers. This increases the search space for the algorithm already drastically. Still, as it can be seen in figure 7.7, it finds suitable results. One result contains contains a load of over 100%. The fitness function is configured to tolerate this in a certain boundary to not restrict the diversity of the results.

The third scenario has a size of 1000×750 kilometers. Over 240000 players are in the region of interest.

In figure 7.8 we can see that the algorithm proposes several solutions which range from a maximum load of approx. 70% and close to 100%. A

[3] http://www.windowsazure.com

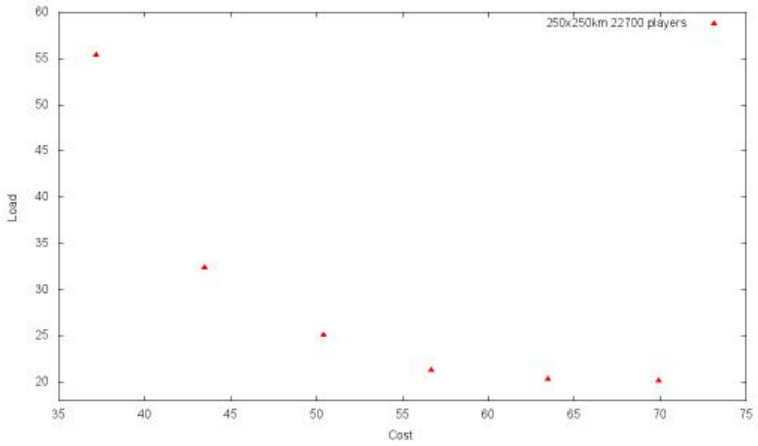


Figure 7.6: The resulting pareto front for the small scenario. The graph shows the solutions in cost ($ cent per hour) versus maximum CPU load on a server.

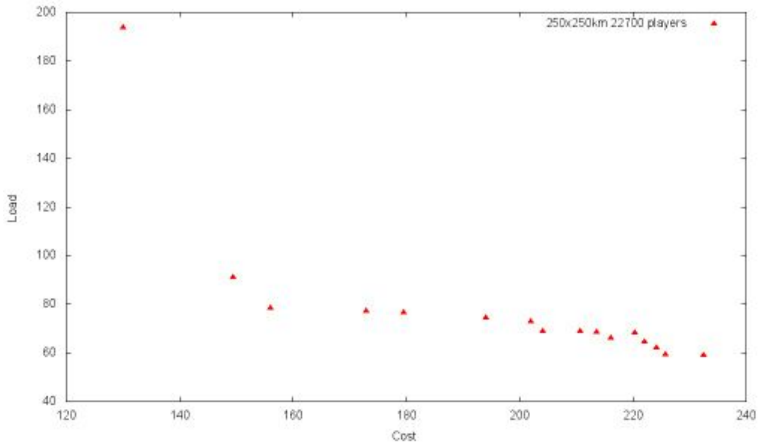


Figure 7.7: Resulting pareto front for the medium scenario.

user could choose to employ 7-13 server instances to cover the region of interest.

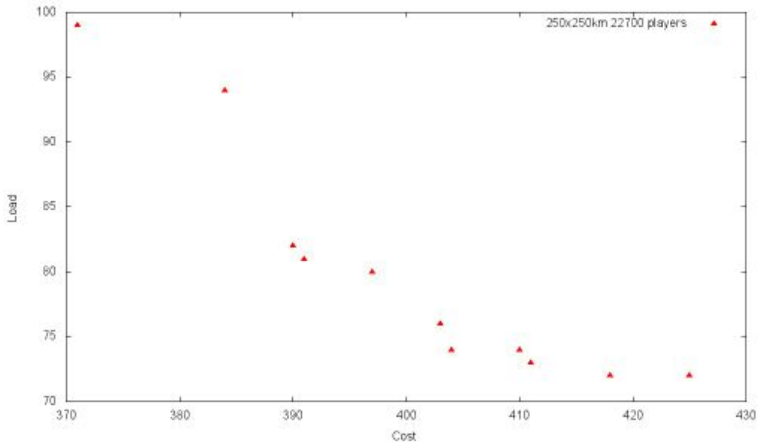


Figure 7.8: Results for the large scenario

Chapter 8

Conclusion

This thesis is concluded by a summary of its contributions and an outlook to future research directions.

8.1 Summary

Resource constraints in VSNs will be one of the major research focus in the coming years. It is not only important in order to save energy on battery-powered devices but it is also necessary to keep cameras at optimal working loads.

This thesis presents several contributions to the field of resource-aware reconfiguration in visual sensor networks.

First, a formalization of the problem has been proposed. Resource constraints are modeled as observation points and objects to track. Covering cameras need to provide sufficient surveillance power to cover them. Contrary to this, they also need to stick to the available resources and minimize the consumption of CPU power, memory and energy. This formal description of the problem is an important foundation to develop algorithms which reconfigure a visual sensor network.

Based on the problem formulation, models for coverage and resources have been developed. The models reflect the coverage of a visual sensor networks including the quality of coverage well as the resources needed for this. In several experimental settings, the models have proven to accurately reflect real measured values.

In exploring the algorithmic design space of this problem several algorithms for different environmental dynamics have been developed and evaluated. A *central* algorithm is able to calculate solutions for environments, where observation points do not dynamically change their properties at runtime. A *distributed* algorithm handles situations, where observation points change slowly and may be added and removed at runtime. A *hybrid* distributed reconfiguration and handover algorithm deals additionally with moving objects.

A thorough evaluation of the developed algorithms has been presented. We have seen that the resource consumption of a visual sensor network can significantly be reduced using the approaches presented in this thesis. In real as well as simulated scenarios, the algorithms were able to improve surveillance quality and lower the resource demand of nodes.

The general usefulness of the presented approach has been shown in a task-assignment problem for cloud infrastructures. Here, the responsibilities of servers has been optimized using cost and load as optimization dimensions.

8.2 Future Research Directions

The work presented in this thesis shows several directions for future research work.

Self-organizing Methods to Improve Reconfiguration

New methods for reconfiguration based on self-organization can be developed. Methods of self-organization are already included in the hybrid algorithm. This can be extended to make the whole optimization process purely self-organizing. There are several areas where the approaches presented in this thesis can be extended with self-organization.

First, models for resource consumption and coverage may be learned in a self-organizing distributed process to reduce the effort of initially deploying a visual sensor network. An approach in this direction needs self-aware VSN nodes which are able to monitor several parameters of their operation. This includes measurement of resource consumption using load monitoring for CPU and memory as well as the measurement of energy consumption using e.g. smart batteries or other sensors. In addition, nodes need to build coverage models dynamically, e.g., by estimating FOV parameters. Further, a distributed mechanism to exchange this information is required to improve the model quality as well as to decrease the effort needed for model construction.

Second, the (distributed) reconfiguration algorithms can be improved by self-organizing methods. The inclusion of e.g. a pheromone system to model the task assignment could be an interesting approach. Here, a reconfiguration can be triggered by the change of pheromone potential which can be modeled to include coverage and resources.

New Applications of Reconfiguration

New applications for resource-aware reconfiguration can be found. As an example, in mobile environments, distributed interactive smartphone applications can be reconfigured. Optimization dimensions may be optimal resource consumption and enhanced user experience. These dimensions are again—as quality and resources are in this thesis—divergent since a higher user experience can be expected to also demand more resources. As an example, showing a video costs more resources than displaying a textual

description of the video's content. However, the video is typically a better experience for the user than the text is.

Another reconfiguration in this area may include certain application intentions. Here, users may be motivated to take certain actions (e.g., take a picture and upload it). In a group of app users, each user may be motivated to perform a different task in order to reach a certain goal. This can be supported by a reconfiguration which decides which content is shown to which user. Reconfiguration actions may also include reconfiguration of the network topology, i.e., to use a local ad-hoc network to connect individual phones instead of using a central server which requires expensive UMTS connection active on each phone.

Integration of Resource Reconfiguration in Application Design

A deep integration of resource reconfiguration into the design flow of distributed software is an interesting direction of research. Regarding (self-organizing) resource-optimization and reconfiguration as part of a to-be-developed software from the beginning opens up new possibilities for reduced energy consumption and enhanced performance of distributed applications. General design guidelines for resource awareness in distributed and non-distributed applications can be developed. Further, the modeling of reconfiguration and reconfigurable application states and properties can be included in a software design process.

Tools and frameworks to support and automate reconfiguration and resource-awareness can be developed. The development of a general framework for reconfigurable applications which provides basic mechanisms to optimize the resource consumption is a very interesting line of research.

Bibliography

[1] Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdhury. A survey on wireless multimedia sensor networks. *Computer Networks*, 51:921–960, 2007.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[3] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[4] F. Al Machot, C. Tasso, B. Dieber, K. Kyamakya, C. Piciarelli, C. Micheloni, S. Londero, M. Valotto, P. Omero, and B. Rinner. Smart resource-aware multimedia sensor network for automatic detection of complex events. In *Proceedings of the 2011 8th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 402–407, 2011.

[5] Fadi Al Machot, Bernhard Dieber, Petra Hossl, Kyandoghere Kyamakya, Sabrina Londero, Christian Micheloni, Paolo Omero, Claudio Piciarelli, Bernhard Rinner, Carlo Tasso, and Massimiliano Valotto. Avss 2011 demo session: Smart resource-aware multi-sensor network. In *Proceedings of the 2011 8th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 529–530, 2011.

[6] L. Albani, P. Chiesa, D. Covi, G. Pedegani, A. Sartori, and M. Vatteroni. Visoc: A smart camera soc. In *Proceedings of the 28th European Solid-State Circuits Conference, 2002. ESSCIRC 2002*, pages 367 –370, sept. 2002.

[7] C. Arth, H. Bischof, and C. Leistner. Tricam - an embedded platform for remote traffic surveillance. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06.*, page 125, june 2006.

[8] D. Bauer, A. N. Belbachir, N. Donath, G. Gritsch, B. Kohn, M. Litzenberger, C. Posch, P. Schön, and S. Schraml. Embedded vehicle speed estimation system using an asynchronous temporal contrast vision sensor. *EURASIP Journal on Embedded Systems*, 2007(1):34–34, January 2007.

[9] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. In *Mobile Data Management*, Lecture Notes in Computer Science, pages 3–14. Springer Berlin / Heidelberg, 2001.

[10] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-time video analysis on an embedded smart camera for traffic surveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004. RTAS 2004*, pages 174 – 181, may 2004.

[11] M. Casares and S. Velipasalar. Adaptive Methodologies for Energy-Efficient Object Detection and Tracking With Battery-Powered Embedded Smart Cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1438–1452, oct. 2011.

[12] Krishnendu Chakrabarty, S. Sitharama Iyengar, Hairong Qi, and Eungchun Cho. Grid Coverage for Surveillance and Target Location in Distributed Sensor Networks. *IEEE Transactions on Computer*, 51(12):1448–1453, 2002.

[13] Chung-Hao Chen, Yi Yao, David Page, Besma Abidi, Andreas Koschan, and Mongi Abidi. Camera Handoff with Adaptive Resource Management for Multi-Camera Multi-Target Surveillance. In *Proceedings of the Fifth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 79 – 86, 2008.

[14] P. Chen, P. Ahammad, C. Boyer, Shih-I Huang, Leon Lin, E. Lobaton, M. Meingast, Songhwai Oh, S. Wang, Posu Yan, A.Y. Yang, Chuohao Yeo, Lung-Chung Chang, J.D. Tygar, and S.S. Sastry. Citric: A low-bandwidth wireless camera network platform. In *Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008. ICDSC 2008.*, pages 1 –10, sept. 2008.

[15] I-Tsun Chiang, Jong-Chang Tsai, and Shang-Ti Chen. Using xbox 360 kinect games on enhancing visual performance skills on institutionalized older adults with wheelchairs. In *Proceedings of the 2012 IEEE Fourth International Conference on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL)*, pages 263 –267, march 2012.

[16] C.A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28 – 36, feb 2006.

[17] Henry Detmold, Anton van den Hengel, Anthony Dick, Katrina Falkner, David S. Munro, and Ron Morrison. Middleware for distributed video surveillance. *IEEE Distributed Systems Online*, 9(2):1, feb. 2008.

[18] B. Dieber, T. Grassauer, J. Mayring, and B. Rinner. The geobashing architecture for location-based mobile massive multiplayer online games. In *Proceedings of the 2010 Fourth International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST)*, pages 13–18, 2010.

[19] B. Dieber, C. Micheloni, and B. Rinner. Resource-aware coverage and task assignment in visual sensor networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1424 –1437, oct. 2011.

[20] Bernhard Dieber, Lukas Esterle, and Bernhard Rinner. Distributed resource-aware task assignment for complex monitoring scenarios in Visual Sensor Networks. In *Proceedings of the 2012 Sixth International Conference Distributed Smart Cameras (ICDSC)*, pages 1–6, 2012.

[21] Bernhard Dieber and Bernhard Rinner. Distributed Online Visual Sensor Network Reconfiguration for Resource-aware Coverage and Task Assignment. In *Globecom 2013 - Ad Hoc and Sensor Networking Symposium (GC13 AHSN)*, Atlanta, USA, December 2013.

[22] Bernhard Dieber, Bernhard Rinner, and Nikolaus Viertl. Flexible Clustering in Networks of Smart Cameras. In *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 834–839. IEEE, October 2009.

[23] Bernhard Dieber, Jennifer Simonjan, Lukas Esterle, Georg Nebehay, Roman Pflugfelder, Gustavo Javier Fernandez, and Bernhard Rinner. Ella: Middleware for Multi-camera Surveillance in Heterogeneous Visual Sensor Networks. In *Proceedings of the Seventh ACM/IEEE International Conference on Distributed Smart Cameras*, Palm Springs, USA, October 2013.

[24] A. Doblander, A. Maier, B. Rinner, and A. Zoufal. An efficient middleware for power-aware service reconfiguration in multi-dsp smart cameras. In *Proceedings of the 2nd Conference on Information and Communication Technologies, 2006. ICTTA '06.*, volume 2, pages 2985 –2990, 0-0 2006.

[25] A. Doblander, B. Rinner, N. Trenkwalder, and A. Zoufal. A lightweight publisher-subscriber middleware for dynamic reconfiguration in networks of embedded smart cameras. In *Proceedings of the 5th World Scientific and Engineering Academy and Society International Conference on Software Engineering, Parallel and Distributed Systems*, 2006.

[26] Y. Durmus, A. Ozgovde, and C. Ersoy. Distributed and Online Fair Resource Management in Video Surveillance Sensor Networks. *Mobile Computing, IEEE Transactions on*, PP(99):1, 2011.

[27] R.A. El-laithy, Jidong Huang, and M. Yeh. Study on the use of microsoft kinect for robotics applications. In *Proceedings of the 2012 IEEE/ION Position Location and Navigation Symposium (PLANS)*, pages 1280 –1288, april 2012.

[28] L. Esterle, P.R. Lewis, M. Bogdanski, B. Rinner, and Xin Yao. A socio-economic approach to online vision graph generation and handover in distributed smart camera networks. In *Proceedings of the 2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, pages 1 –6, aug. 2011.

[29] Lukas Esterle, Peter R. Lewis, Xin Yao, and Bernhard Rinner. Socio-economic vision graph generation and handover in distributed smart camera networks. *ACM Transactions on Sensor Networks*, 10(2), 2014. In press.

[30] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[31] S. Fleck and W. Strasser. Smart camera based monitoring system and its application to assisted living. *Proceedings of the IEEE*, 96(10):1698 –1714, oct. 2008.

[32] Sven Fleck, Florian Busch, and Wolfgang Strasser. Adaptive probabilistic tracking embedded in smart cameras for distributed surveillance in a 3d model. *EURASIP Journal on Embedded Systems*, 2007(1):24–24, January 2007.

[33] Chien-Liang Fok, G. C. Roman, and Chenyang Lu. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In *Proceedings of the IEEE International Conference on Distributed Computing Systems, 2005*, pages 653–662, 2005.

[34] Giordano Fusco and Himanshu Gupta. Selection and Orientation of Directional Sensors for Coverage Maximization. In *Proceedings of the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1 – 9, 2009.

[35] Dimitrios Georgoulas and Keith Blow. Making Motes Intelligent: An Agent-Based Approach to Wireless Sensor Networks. *WSEAS on Communications Journal*, 5(3):525–522, March 2006.

[36] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM.

[37] Hamid Aghajan and Andrea Cavallaro, editor. *Multi-Camera Networks*. Elsevier, 2009.

[38] A. Hampapur. Smart video surveillance for proactive security [in the spotlight]. *IEEE Signal Processing Magazine*, 25(4):136 –134, july 2008.

[39] A. Hampapur, S. Borger, L. Brown, C. Carlson, J. Connell, M. Lu, A. Senior, V. Reddy, C. Shu, and Y. Tian. S3: The ibm smart surveillance system: From transactional systems to observational systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007*, volume 4, pages IV–1385 –IV–1388, april 2007.

[40] Xiaofeng Han, Xiang Cao, Errol L. Lloyd, and Chien-Chung Shen. Deploying Directional Sensor Networks with Guarranteed Connectivity and Coverage. In *Proceedings of the IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 153–160, San Francisco, CA, USA, 2008.

[41] Zhihai He and Dapeng Wu. Resource Allocation and Performance Analysis of Wireless Video Sensors. *IEEE Transactions on Circuits and Systems for Video Technology*, 16:590–599, 2006.

[42] S. Hengstler, D. Prashanth, Sufen Fong, and H. Aghajan. Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks, 2007. IPSN 2007*, pages 360 –369, april 2007.

[43] M. Hoffmann, M. Wittke, Y. Bernard, R. Soleymani, and J. Hahner. Dmctrac: Distributed multi camera tracking. In *Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008. ICDSC 2008.*, pages 1 –10, sept. 2008.

[44] Eva Hörster and Rainer Lienhart. Approximating Optimal Visual Sensor Placement. In *Proceedings of the International Conference on Multimedia and Expo*, pages 1257–1260, 2006.

[45] M. Jovanovic and B. Rinner. Middleware for dynamic reconfiguration in distributed camera systems. In *Proceedings of the 2007 Fifth Workshop on Intelligent Solutions in Embedded Systems*, pages 139 –150, june 2007.

[46] Deepak R. Karuppiah, Roderic A. Grupen, Zhigang Zhu, and Allen R. Hanson. Automatic resource allocation in a distributed camera network. *Machine Vision and Applications*, 21:517–528, 2010.

[47] Kwangsu Kim and G. Medioni. Robust real-time vision for a personal service robot in a home visual sensor network. In *Proceedings of the 16th IEEE International Symposium on Robot and Human interactive Communication, 2007. RO-MAN 2007*, pages 522 –527, aug. 2007.

[48] R. Kleihorst, M. Reuvers, B. Krose, and H. Broers. A smart camera for face recognition. In *Proceedings of the 2004 International Conference on Image Processing, 2004. ICIP '04.*, volume 5, pages 2849 – 2852 Vol. 5, oct. 2004.

[49] Purushottam Kulkarni, Deepak Ganesan, Prashant Shenoy, and Qifeng Lu. Senseye: a multi-tier camera sensor network. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 229–238, New York, NY, USA, 2005. ACM.

[50] Yiming Li and Bir Bhanu. A comparison of techniques for camera selection and handoff in a video network. In *Proceedings of the Third ACM/IEEE International Conference on Distributed Smart Cameras, 2009. ICDSC 2009*, pages 1 –8, 30 2009-sept. 2 2009.

[51] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, March 2005.

[52] Arnold Maier, Bernhard Rinner, Wolfgang Schriebl, and Helmut Schwabach. Online Multi-Criterion Optimization for Dynamic Power-Aware Camera Configuration in Distributed Embedded Surveillance Clusters. In *Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 06)*, pages 307–312, Vienna, Austria, April 2006.

[53] A. Marcus and O. Marques. An eye on visual sensor networks. *Potentials, IEEE*, 31(2):38 –43, april 2012.

[54] Memcached. `http://www.memcached.org`.

[55] Christian Micheloni, Bernhard Rinner, and Gian Luca Foresti. Video Analysis in PTZ Camera Networks - From master-slave to cooperative smart cameras. *IEEE Signal Processing Magazine*, 27(5):78–90, 2010.

[56] L. Middleton, S.C. Wong, M.O. Jewell, J.N. Carter, and M.S. Nixon. A middleware for a large array of cameras. In *Proceedings of the*

2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005), pages 801 – 806, aug. 2005.

[57] Anurag Mittal and Larry S. Davis. A General Method for Sensor Planning in Multi-Sensor Systems: Extension to Random Occlusion. *International Journal of Computer Vision*, 76:31–52, 2008.

[58] Mohammad M. Molla and Sheikh Iqbal Ahamed. A survey of middleware for sensor network and challenges. In *Processdings of the 2006 International Conference on Parallel Processing Workshops, 2006. ICPP 2006 Workshops.*, pages 1 – 6 pp., 2006.

[59] Eduardo Monari and Kristian Kroschel. Task-Oriented Object Tracking in Large Distributed Camera Networks. In *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2010.

[60] T.W.J. Moorhead and T.D. Binnie. Smart cmos camera for machine vision applications. In *Proceedings of the Seventh International Conference on Image Processing and Its Applications, 1999. (Conf. Publ. No. 465)*, volume 2, pages 865 –869 vol.2, 1999.

[61] J. Nayak, L. Gonzalez-Argueta, Bi Song, A. Roy-Chowdhury, and E. Tuncel. Multi-target tracking through opportunistic camera control in a resource constrained multimodal sensor network. In *Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008. ICDSC 2008*, pages 1 –10, sept. 2008.

[62] S. Obdrzalek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, and M. Pavel. Accuracy and robustness of kinect pose estimation in the context of coaching of elderly population. In *Proceedings of the 2012 IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBC)*, pages 1188 –1193, 28 2012-sept. 1 2012.

[63] Yahya Osais, Marc St-Hilaire, and F. Richard Yu. The Minimum Cost Sensor Placement Problem for Directional Wireless Sensor Networks. In *Proceedings of the IEEE Vehicular Technology Conference*, pages 1 – 5, 2008.

[64] Yahya Osais, Marc St-Hilaire, and F. Richard Yu. On Sensor Placement for Directional Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1 – 5, Dresden, Germany, 2009.

[65] Jianping Pan, Lin Cai, Y. Thomas Hou, Yi Shi, and Sherman X. Shen. Optimal Base-Station Locations in Two-Tiered Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 4(5):458–473, 2005.

[66] M. Parajuli, Dat Tran, Wanli Ma, and D. Sharma. Senior health monitoring using kinect. In *Proceedings of the 2012 Fourth International Conference on Communications and Electronics (ICCE)*, pages 309 –312, aug. 2012.

[67] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. Occlusion-aware multiple camera reconfiguration. In *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras*, pages 88–94, 2010.

[68] F. Pletzer, R. Tusch, L. Boszormenyi, and B. Rinner. Robust traffic state estimation on smart cameras. In *Proceedings of the 2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 434 –439, sept. 2012.

[69] M. Quaritsch and B. Rinner. Dscagents: A lightweight middleware for distributed smart cameras. In *Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008. ICDSC 2008*, pages 1 –8, sept. 2008.

[70] M. Quaritsch, B. Rinner, and B. Strobl. Improved agent-oriented middleware for distributed smart cameras. In *Proceedings of the First ACM/IEEE International Conference on Distributed Smart Cameras, 2007. ICDSC '07*, pages 297 –304, sept. 2007.

[71] Faisal Qureshi and Demetri Terzopoulos. Smart Camera Networks in Virtual Reality. In *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1640 – 1656, Vienna, Austria, 2007.

[72] Mohammad Rahimi, Rick Baer, Obimdinachi I. Iroezi, Juan C. Garcia, Jay Warrior, Deborah Estrin, and Mani Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 192–204, New York, NY, USA, 2005. ACM.

[73] Ramesh Rajagopalan, Chilukuri K Mohan, Kishan G Mehrotra, and Pramod K Varshney. Multi-objective evolutionary algorithms for sensor network design. In Lam Thu Bui and SameerEditors Alam, editors, *MultiObjective Optimization in Computational Intelligence Theory and Practice*, pages 208–238. Information Science Reference, 2008.

[74] U. Ramachandran, K. Hong, L. Iftode, R. Jain, R. Kumar, K. Rothermel, Junsuk Shin, and R. Sivakumar. Large-scale situation awareness with camera networks and multimodal sensing. *Proceedings of the IEEE*, 100(4):878 –892, april 2012.

[75] B. Rinner, M. Jovanovic, and M. Quaritsch. Embedded middleware on distributed smart cameras. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007.*, volume 4, pages IV–1381 –IV–1384, april 2007.

[76] Bernhard Rinner, Bernhard Dieber, Lukas Esterle, Peter R. Lewis, and Xin Yao. Resource-aware configuration in smart camera networks. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 58–65, 2012.

[77] Bernhard Rinner and Markus Quaritsch. Embedded middleware for smart camera networks. In Hamid Aghajan and Andrea Cavallaro, editors, *Multi-Camera Networks*. Elsevier, 2009.

[78] Bernhard Rinner, Markus Quaritsch, Wolfgang Schriebl, Thomas Winkler, and Wayne Wolf. The Evolution from Single to Pervasive Smart Cameras. In *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC-08)*, pages 1–10, Stanford, USA, 2008.

[79] Bernhard Rinner and Wayne Wolf. Introduction to distributed smart cameras. *Proceedings of the IEEE*, 96(10):1565–1575, October 2008.

[80] Bernhard Rinner and Wayne Wolf. Toward Pervasive Smart Camera Networks. In Hamid Aghajan and Andrea Cavallaro, editors, *Multi-Camera Networks*, pages 483–496. Elsevier, 2009.

[81] Corina Rotar, Mircea RÎsteiu, Ioan Ileana, and C-Tin Hutanu. Optimal sensors network layout using evolutionary algorithms. In *Proceedings of the 10th WSEAS international conference on Automation & Information*, pages 88–93, Stevens Point, Wisconsin, USA, 2009. World Scientific and Engineering Academy and Society (WSEAS).

[82] A. Rowe, A. Goode, D. Goel, and I. Nourbakhsh. Cmucam3: an open programmable embedded vision sensor. In *Proceedings of the International Conferences on Intelligent Robots and Systems*, 2007.

[83] Wolfgang Schriebl and Bernhard Rinner. A resource-aware distributed event space for pervasive smart camera networks. In *Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras*, ICDSC '10, pages 82–87, New York, NY, USA, 2010. ACM.

[84] SharpMap. `http://sharpmap.codeplex.com/`.

[85] Yu Shi, P. Raniga, and I. Mohamed. A smart camera for multimodal human computer interaction. In *Proceedings of the 2006 IEEE Tenth*

International Symposium on Consumer Electronics, 2006. ISCE '06., pages 1 –6, 0-0 2006.

[86] Yu Shi and Timothy Tsui. An fpga-based smart camera for gesture recognition in hci applications. In Yasushi Yagi, SingBing Kang, InSo Kweon, and Hongbin Zha, editors, *Computer Vision ACCV 2007*, volume 4843 of *Lecture Notes in Computer Science*, pages 718–727. Springer Berlin Heidelberg, 2007.

[87] Hsien-Po Shiang and Mihaela van der Schaar. Information-Constrained Resource Allocation in Multicamera Wireless Surveillance Networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(4):505–517, 2010.

[88] J. Shin, R. Kumar, D. Mohapatra, U. Ramachandran, and M. Ammar. Asap: A camera sensor network for situation awareness. *Principles of Distributed Systems*, pages 31–47, 2007.

[89] P. Sikka, P. Corke, L. Overs, P. Valencia, and T. Wark. Fleck - a platform for real-world outdoor sensor networks. In *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007*, pages 709 –714, dec. 2007.

[90] J. Smisek, M. Jancosek, and T. Pajdla. 3d with kinect. In *Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1154 –1160, nov. 2011.

[91] F. Soltani, F. Eskandari, and S. Golestan. Developing a gesture-based game for deaf/mute people using microsoft kinect. In *Proceedings of the 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 491 –495, july 2012.

[92] Stanislava Soro. *Application-Aware Resource Management in Wireless and Visual Sensor Networks*. PhD thesis, School of Engineering and Applied Sciences, University of Rochester, 2007.

[93] Stanislava Soro and Wendi Heinzelman. A Survey of Visual Sensor Networks. *Advances in Multimedia*, pages 1–21, 2009.

[94] Cristian Soto, Bi Song, and Amit K. Roy-Chowdhury. Distributed multi-target tracking in a self-configuring camera network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1486–1493, Miami, USA, 20-25 Jun 2009.

[95] M. Tanaka. Robust parameter estimation of road condition by kinect sensor. In *Proceedings of the 2012 Proceedings on SICE Annual Conference (SICE)*, pages 197 –202, aug. 2012.

[96] Xiaoyu Tong and E.C.-H. Ngai. A ubiquitous publish/subscribe platform for wireless sensor networks with mobile mules. In *Proceedings of the 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 99 –108, may 2012.

[97] M. Valera and S.A. Velastin. Intelligent distributed surveillance systems: a review. *IEEE Proceedings on Vision, Image and Signal Processing*, 152(2):192 – 204, april 2005.

[98] Adam Williams, Deepak Ganesan, and Allen Hanson. Aging in place: fall detection and localization in a distributed smart camera network. In *Proceedings of the 15th international conference on Multimedia*, MULTIMEDIA '07, pages 892–901, New York, NY, USA, 2007. ACM.

[99] T. Winkler and B. Rinner. Trustcam: Security and privacy-protection for an embedded smart camera based on trusted computing. In *Proceedings of the 2010 Seventh IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 593 –600, 29 2010-sept. 1 2010.

[100] Thomas Winkler and Bernhard Rinner. Pervasive Smart Camera Networks exploiting heterogeneous wireless Channels. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 296 – 299, March 2009.

[101] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *Computer*, 35(9):48 – 53, sep 2002.

[102] Mohamed Younis and Kemal Akkaya. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, 6:621 ?– 655, 2008.

[103] Chao Yu and Gaurav Sharma. Camera Scheduling and Energy Allocation for Lifetime Maximization in User-Centric Visual Sensor Networks. *IEEE Transactions on Image Processing*, 19(8):2042–2055, 2010.

[104] Yang Yu, Bhaskar Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, 2004.

[105] Ali Akbar Zarezadeh and Christophe Bobda. Hardware orb middleware for distributed smart camera systems. In Toomas P. Plaks, David Andrews, Ronald F. DeMara, Herman Lam, Jooheung Lee, Christian Plessl, and Greg Stitt, editors, *ERSA*. CSREA Press, 2010.

[106] Ali Akbar Zarezadeh and Christophe Bobda. Hardware middleware for person tracking on embedded distributed smart cameras. *International Journal of Reconfigurable Computing*, vol. 2012:10pp, 2012.

[107] Z. Zhang. Microsoft kinect sensor and its effect. *Multimedia, IEEE*, 19(2):4–10, 2012.

[108] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. *Metaheuristics for Multiobjective Optimisation*, 535(535):21–40, 2004.

[109] Junni Zou, Chong Ta, Ruifeng Zhang, and Hongkai Xiong. Modeling and Optimization of Network Lifetime in Wireless Video Sensor Networks. In *Proceedings of the IEEE International Communications Conference*, pages 1 – 6, 2010.