5-2015

Enabling Runtime Self-Coordination of Reconfigurable Embedded Smart Cameras in Distributed Networks

Franck Ulrich Yonga Yonga
University of Arkansas, Fayetteville

Enabling Runtime Self-Coordination of Reconfigurable Embedded Smart Cameras in Distributed Networks

Enabling Runtime Self-Coordination of Reconfigurable Embedded Smart Cameras in Distributed Networks

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Engineering

by

Franck Ulrich Yonga Yonga
University of Yaounde I
Bachelor of Science in Computer Science, 2005
University of Yaounde I
Master of Science in Computer Science, 2008

May 2015
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Dr. Christophe Bobda
Dissertation Director

Dr. David Andrews
Committee member

Dr. Chaim Goodman-Strauss
Committee member

Dr. John Gauch
Committee member

Abstract

Smart camera networks are real-time distributed embedded systems able to perform computer vision using multiple cameras. This new approach is a confluence of four major disciplines (computer vision, image sensors, embedded computing and sensor networks) and has been subject of intensive work in the past decades. The recent advances in computer vision and network communication, and the rapid growing in the field of high-performance computing, especially using reconfigurable devices, have enabled the design of more robust smart camera systems. Despite these advancements, the effectiveness of current networked vision systems (compared to the operating costs) is still disappointing. The main reason is the poor coordination of the camera entities at runtime which results from the lack of a clear formalism to dynamically and autonomously capture and address the self-organization problem. In this dissertation, we investigate the use of a declarative-based modeling approach for capturing runtime self-collaboration of distributed smart cameras. Combining modeling approaches borrowed from logic programming, computer vision techniques, and high-performance computing, we propose an autonomous and cooperative smart camera system. We also propose a compact modeling approach based on Answer Set Programming for architecture synthesis of a system-on-chip camera that is able to support runtime collaboration with other camera nodes in a distributed network setup. Finally, we propose a declarative approach for enabling runtime camera self-coordination in case of distributed object tracking wherein moving targets are decentrally handed over and successfully recovered after node failure.

Acknowledgements

This work has been partially supported by the National Science Foundation under grant CNS-1302596.

I am heartily thankful to my supervisor, Dr. Christophe Bobda, whose guidance, leadership, and encouragement from the initial phase of this work back in Germany to the final level enabled me to develop an understanding of the addressed topic. I was very fortunate to learn under his guidance the importance of hardworking and perseverance, especially in the midst of doubts and uncertainties. I am also thankful for all support and advices that I have received from him beyond the scope of our academic collaboration.

To Dr. David Andrews, I would like to express my sincere gratitude for the constructive criticism that considerably improved the quality of this dissertation. I also want to give many thanks to the other members of this dissertation committee, Dr. John Gauch and Dr. Chaim Goodman-Strauss, for the time they spent on reviewing this work, and especially for have accepted to be part of this committee. I appreciate all of the suggestions and comments.

Many thanks to my colleagues from the Smart Embedded System laboratory and the Computer Systems Design laboratory for their strong cooperation in different projects. Most especially to Michael Mefenza for his tremendous technical support during this 4 years of research. An immense thank to all my dear friends here in Fayetteville and to the Saint Thomas Aquinas University Parish community, who always provided to me a friendly and spiritual home away from my home.

Last but not least, special thanks to my family and friends. To my dear father, Jean-Marie Yonga, and my beloved mother, Monique Kengou, who constantly encouraged me and provided the financial and love support I needed to reach this level of study. To my *other mom* Sabine Ngandjoui, for the special care she always gave to me. To my siblings, Simplice, Emerencienne, Sandrine, Francky, Cedric, Dominique, Christelle, and Henri Yonga; my siblings-in-law Patrick Ngaleu, Hypolite Pene and Michelle Djeuthieu; my dear friends Roberta, Edwin, and Diane Moukam, please find through this work a humble expression of my profound gratitude.

Dedication

This Work is dedicated to God Almighty, who alone is the Author of all things.

Table of Contents

List of Figures

List of Tables

I Introduction

The recent technological advances in circuit design, computer vision, machine learning, and sensor networks, have led to a proliferation of intelligent vision systems as a response to the increasing insecurity in many countries around the world. Indeed, many of the existing video systems can now support very sophisticated computer vision operations, such as object detection and tracking, behavior recognition, and activity analysis from multiple camera sources. However, despite the decreasing price of equipments and the available technology, the effectiveness of these networked systems are still disappointing and their operating costs, very high [1], as could be illustrated with the following application examples.

I.1 Motivating Applications

I.1.1 In-store Observation for addressing Shoplifting in Retail Stores

According to the National Retail Security Survey Final, retailers experienced shrinkage of 1.51% to sales in 2008, which translated to roughly $36.6 billion in retail lost annually [2, 3]. Despite millions of dollars that retail companies spend every year on asset protection and theft detection, yet they still incur tremendous losses. The efficiency of current theft detection and prevention systems, such as exception reporting, cameras surveillance, and article/customer monitoring, is only marginal, resulting in negligible positve results compared to the overall losses [4]. Current In-Store Observation systems use many surveillance cameras installed in a Closed Circuit Television (CCTV) architecture. While CCTV provide a global view of the in-store activities, operators are needed to check and detect potential theft in real-time. This process is less than trivial. In fact, with the number of cameras installed in store and the amount of customers (particularly in period of high traffic), an army is required to check each customer, identify potentials suspects and follow them across the store. Moreover, recording suspicious activities, such as putting an item into a bag, is not enough and should not always be assumed as theft, since the customer might have dropped the item later without the camera picking up the dropping part.

I.1.2 Autonomous Guided Vehicles for Material Transport in Industry

Autonomous Guided Vehicles (AGV) have the potential to revolutionize operation in areas such as manufacturing, distribution, transportation, and military. In these areas, AGVs can efficiently be used to accomplish mundane and often repetitive tasks such as transporting materials in manufacturing or deploying troops or equipments on battlefields. Despite obvious advantages of AGVs, the dream of having hundreds of such vehicles, collaboratively and autonomously performing tasks has not materialized so far. The biggest impediment to date has been the lack of models and technologies to actively capture the world with semantically labeled objects, actions and events, and to generate goals, priorities, and plans. On board computational limitations coupled with complexity of the environment have resulted in highly specialized, brittle and non-scalable solutions. Efficient, cost-effective, and dynamic localization and collaboration needed for indoor navigation has not been satisfactorily addressed. Radio frequency communications proposed in the literature and used in teleoperation of unmanned ground vehicles still pose huge challenges, particularly in indoor environment, due to interference and noise, potential for jamming, bandwidth, and latency [5]. Approaches that rely on dense scene reconstruction from a variety of sensor data such as LIDAR and video imagery relay are still too expensive [6].
The distributed nature of the aforementioned applications and the complexity of the running environment necessitate the use of a robust and decentralized approach wherein a network of self-collaborative, resource-efficient smart cameras will be used for gapless identification, tracking, and coordination of targets of interest (people or vehicles) across the entire monitoring region. Improving the efficiency and maximizing operations in such dynamic multi-camera vision environments present some challenges that have motivated the research in the present dissertation.

I.2 Challenges

I.2.1 Limitations of Centralized Networked Architectures

The possible integration of hundreds and more nodes into a single camera network has increased the complexity of a remote monitoring. Most of the current surveillance systems are made up of CCTV-Based cameras that simply transmit raw videos to a

central location for processing and analysis. In such a network system, human operators are completely responsible for monitoring ongoing activities on dozen (or more) of TV screens, analyzing the scenes, and coordinating the different cameras. In addition to having a single point of failure, such network setups have also proven to be ineffective because of the burden placed on the operators and the communication costs involved. Monitoring in this case requires to visually analyze hours of video footage and it has been shown in [7] that an operator will often miss up to 95% of all scene activity after approximatively 22 minutes when viewing two or more sequencing monitors.
Furthermore, with recent advances in image resolution, CCTV-Based systems require a high bandwidth communication network and a reliable interconnection mechanism among nodes, usually provided only through fixed and broadband protocols such as Ethernet. This brings the necessity of implementing vision processing directly within the cameras and enabling a strong cooperation among nodes to improve the system productivity.

I.2.2 Lack of Coordination in current Smart Camera Networks

Distributed smart camera (DSC) provide more flexibility in a network by disseminating processing and analysis on the end nodes. A smart camera is generally viewed as a system able to perform on-site processing and extract meaningful information within video frames. However, without a compact formalism that would allow those information to be exchanged among cameras, the effectiveness of the whole network would be seriously mitigated. For example, let's assume that every camera node in a typical surveillance system performs video processing on-situ and generates 5 events in average every second (person running, car parking, pedestrian crossing the street, etc.). For a network of only 20 cameras, this amounts to almost a hundred of abstract information that has to be merged and analyzed every second at the remote station. It is obvious that this task cannot be performed manually by a human, but instead, an auto-collaboration scheme among cameras has to be investigated.
Over the past few years, while tremendous efforts have been done (both in industry and academia) to increase the processing power of embedded camera systems, very few works have focused on improving the self-collaboration of networked nodes. Existing smart cameras are powerful enough to implement very complex computer vision operations such as the automated face recognition and tracking in a crowd [8]. The ongoing progress in

chip technology yields a steadily increasing processing power and it is obvious that in the near future even the weakest sensor node in a network will have enough computational power to share its resources as services with others. Unfortunately, self-coordination of nodes is an important aspect that has been overlooked very often when designing network of smart cameras, although it has been proven [9] that the effectiveness of a such a network could be drastically improved when the different nodes collaborate for efficient problem solving. Enabling self-coordination of distributed cameras presents many advantages, such as: more *flexibility* and *interoperability*, increase in *scalability* and *reliability*, *reduced bandwidth utilization* and *system response time*.

I.2.3 Unsuitable Solutions for Addressing Self-Coordination in DSC

Practical examples demonstrating the use of self-coordination properties in embedded smart camera systems are still lacking. Although many researches have focused on developing heuristics for the self-coordination in computer system [10, 11, 12, 13], none of these works has successfully addressed the complexity of modeling the coordination of distributed cameras, especially considering the network dynamics (nodes failure or nodes overwhelming) and the tight resource and power constraints on the available embedded infrastructures. Additionally, many of the existing solutions, such as the multi-agent systems [14, 15], mostly focus on improving the software functionality of the system and are developed for platforms with minimal or non-existent SWaP constraints (size, weight, and power). In contrary, embedded smart cameras are generally battery-powered with a limited lifetime and hardware capabilities (CPUs, memories, and buses). Therefore, to build such camera systems with tight resource constraints that could support the complex operations required for self-coordination in a dynamic environment, the prerequisites are: 1) a rigorous formalism that captures all facets of the self-coordination problem together with the resource constraints on the computing infrastructure, 2) a computational tractable and accurate strategy to devise a solution from the previous formulation, and 3) a flexible computing infrastructure for each smart camera that could be modified on the fly to adapt to runtime changes in the network.

I.2.4 A Hardware/Software Co-Design Approach for Optimizing Resource Utilization

When addressing self-coordination, software solutions are usually implemented without considering the infrastructure on which they will be executed. As result, the optimal use of the computing platform is hardly achieved because either solutions are not implementable on the available architectures (over-utilization) or resources are not used efficiently (under-utilization). An illustration might be the execution of a convolution function in software while a hardware implementation could have provided more speedup. As a solution, the hardware and software design should be tackled in tandem by using a multi-objective SoC synthesis methodology that would perform a systematic mapping of application blocks unto the available hardware while optimizing resource utilization.

I.3 Contribution of the Dissertation

The main contributions of this dissertation are:

- a self-coordinating methodology based on a declarative modeling approach to autonomously adapt the behavior of cameras to runtime environmental changes, including node failures.
- a holistic approach for the synthesis of a system-on-chip computing infrastructure that will allow hardware restructuration at runtime.

In this dissertation, we propose the use of Answer Set Programming (ASP), a declarative programming paradigm, as a viable alternative for efficiently implementing self-coordination among a set of collaborative nodes in a DSC network. Basically, the self-coordination issue is addressed by first modeling the problem on each node as a logic program that captures the network environment (camera topology), the internal state of the camera (target properties, tracking capacity, etc.), and the system objectives (balance load distribution, minimize object-to-camera distance, etc.). Then, the logic program is solved using an embedded answer set solving engine wherein all possible coordination scenarios (from each camera perspective) are dynamically evaluated and only those that do not violate the objectives stated in the input problem will be generated as stable solutions. Each solution consists in a sequence of actions/commands that individual cameras will execute in order to bring the networked system into the target state. In

existing methodologies [16, 17, 18, 19], solutions to self-coordination consist of step-by-step algorithms that are hardwired among other camera functionalities. This follows the imperative programming approach where the focus is usually on *how* to solve a given problem. With such a paradigm, however, the difficulty of encoding an algorithm to solve a given problem increases proportionally with the system complexity. In contrary, our proposed declarative-based approach focuses on defining *what* is the problem to be solved and then relies on existing powerful solvers to generate efficient solutions to the specified problem. By focusing on *what to self-coordinate* rather than *how to self-coordinate*, the proposed approach relieves the designer from manually programming a solution to camera coordination and reduces the addressed problem to the complexity of modeling runtime self-coordination. This methodology offers, therefore, benefits like interoperability and reusability since the modeling of the coordination problem is explicitly separated from the specification of problem instances. In other words, once the coordination problem is encoded, it can be applied/tested on different network scenarios without modification. Also, programmability could be considered as a benefit since the bulk of work is henceforth shifted from programing complex algorithms for implementing self-coordination to modeling the problem itself.

In order to efficiently address real-time requirements and all changes in the surrounding, low-level architecture on each node in a distributed network should be designed in such a way to maximize operations and support runtime readjustment. Consequently, adaptivity and flexibility must be key factors when designing the smart camera infrastructures. By using Field Programmable Gate Arrays (FPGAs) as the main processing infrastructure on cameras, we intend to provide the required processing power and flexibility at runtime through the combination of hardware and software on a single chip and the possibility of restructuring the hardware at runtime. In such a decomposition, while complex and repetitive computations of a processing chain are implemented directly in hardware, the control part is carried out by an embedded processor. Moreover, due to their parallel nature and reconfigurable capacity, FPGAs are becoming increasingly attractive for image processing as they can easily exploit parallel structures in many computer vision problems and operate in different computation modes, compared to fixed architecture devices such as serial CPUs, ASICs, and DSPs [20, 21]. Nevertheless, the resource limitations on FPGAs and the possibility of swapping tasks between hardware (HW) and software (SW) increase the design complexity on such architectures and require the use of an automatic

design process. In this regard, we propose a multi-objective synthesis methodology of FPGA-Based systems-on-chip (SoC) for the camera nodes in the network. The synthesis approach leverages ASP to i) capture the network environment (topology, resource and power constraints on camera and synthesis goals), ii) perform the design exploration of all possible SoC configurations, and iii) produce only the optimum architectures regarding user-defined objectives.

I.4 Thesis Structure

The presentation in this dissertation follows a bottom-up approach. First, we present in Chapter II, the concept behind Answer Set Programing, a declarative-based modeling approach that is used in this research to both encode the synthesis of systems-on-chip and to model the self-coordination problem. Then, the synthesis methodology to devise a reconfigurable system-on-chip for the smart cameras will be presented in Chapter III. Next, Chapter IV will provide a more detailed presentation of the generated camera SoC; especially how the architecture has been designed so as to support runtime self-coordination operations. Afterward, a runtime object tracking and self-coordinated camera handover system will be presented in Chapter V as an application example to validate the proposed concept. Finally, a summary and conclusion to the dissertation will be provided in Chapter VI.

II Answer Set Programming

II.1 Definition

Answer Set Programming (ASP [22]) is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems and which stems roots in the area of nonmonotonic reasoning and logic programming [23]. It has already been used in various domains including multi-processor system synthesis [24], decision problems [25], reasoning tools in system biology [26, 27], or package configuration in Linux [28] and has proven to yield satisfactory performances compared to other synthesis methodologies such as Integer Linear Programming or evolutionary algorithms [29].
The basic idea behind ASP is that, computational problems are reduced to logic programs whose stable models –sets without conflicting statements like $\{a, not\ a\}$– correspond to solutions to the initial problem. A logic solver is then used to search for stable models and produce the answer sets, which are basically the minimum set among all possible stable solutions. The computation of answer sets relies on instantiation of the input logic program, also known as *grounding*, which produces a so called *variable-free* program. The new generated program is a propositional object made of a set of clauses –finite disjunction of literals like $\{a \vee \neg b \vee c\}$– that will be given to a satisfiability solver for being resolved following a SAT-based solving approach (Boolean Satisfiability [30]). This solving methodology is almost similar to the SLDNF-resolution scheme [31] used in Prolog programming, but with the significant difference that the search for stable models is guaranteed to always terminate. We refrain in this Dissertation from delving into too much semantic details and therefore refer the reader to [22] for a deeper understanding.

II.2 Syntax and Semantics of ASP

ASP uses facts, constraints, rules, and other language elements, mostly derived from Prolog, to specify an instance of a logic problem. For the sake of readability, the language specification will be hereafter given according to the traditional mathematical notation of ASPCore2.0 [32]. A rule r in ASP is an expression of the form:

$$\{a_1; \ldots; a_k\} \leftarrow b_1, \ldots, b_m,\ not\ c_1, \ldots,\ not\ c_n. \tag{II.1}$$

with $k, m, n \geq 0$ and $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ being classical literals. We call *head* of the rule r, denoted by $head(r)$, any subset of $\{a_1, \ldots, a_k\}$. Similarly, the *body* of the rule r is the set $body(r) = \{b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n\}$, where $\{b_1, \ldots, b_m\}$ represents the *positive body literals* denoted by $body^+(r)$, and $\{c_1, \ldots, c_n\}$, the *negative body literals* denoted by $body^-(r)$. A rule (II.1) is a *fact* if $body(r) = \emptyset$ and it is a *constraint* (or *integrity constraint*) when $head(r) = \emptyset$. r is called a *normal* rule if $|head(r)| = 1$ and a *choice* rule if $|head(r)| > 1$ ($|.|$ expresses the cardinality of a set). Intuitively, the rule r is interpreted as: if literals $\{b_1, \ldots, b_m\}$ are true and there is no evidence that any of the atoms c_i $(1 \leq i \leq n)$ holds, then an arbitrary subset of $\{a_1, \ldots, a_k\}$ can be chosen as true. The language also defines *weighted* constraints or aggregate [33] as follows:
$l < \#aggr\{\ v_0; v_1; \ldots; v_n\} < u$ where *aggr* can be either *sum*, *max*, *min*, or *count*. This constraint describes the fact that a subset $A \subseteq \{v_0, \ldots, v_n\}$ of true atoms exists, such that the sum (resp. maximum, minimum, or number) of weights is within the boundaries $[l; u]$. In case *#aggr* would be omitted, the aggregate would default to computing the *sum* of atoms v_i.
For the computation of answer sets, let's consider P to be a logic program, and let's X be a consistent set of literals, i.e for every atom $a \in X$, $\{a, \neg a\} \not\subseteq X$ (where the symbol $\neg$ is the classical negation). We say that X satisfies a rule r of the form (II.1), iff $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. Similarly, X is said to satisfy the program P –or X is *closed* under P– iff it satisfies all the rules in P. We say that X is a stable model or an answer set for the logic program P if it is minimal (relative to set inclusion) among the sets that satisfy P.
For an illustration, let's consider the following example:

$$a. \tag{II.2}$$

$$c, d \leftarrow a, not\ b. \tag{II.3}$$

$$e \leftarrow c. \tag{II.4}$$

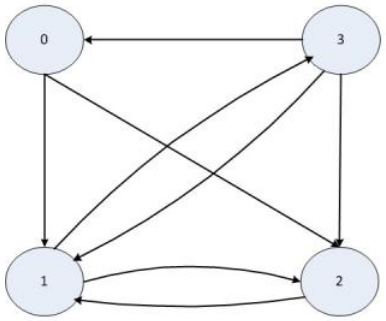
$$\leftarrow a, d. \tag{II.5}$$

This program consists of one *fact* (II.2), two *rules* (II.3) and (II.4), and one *integrity constraint* (II.5). The fact in (II.2) denotes a true information that will always be part of any solution set. Facts are used to specify an instance of the problem that will be applied to a given encoding model. Rule (II.3) is made up of the head ($\{c, d\}$) and the body (e.g.:$\{a, not\ b\}$). This rule states that atoms c and d are exclusively true –that is, they

will be both derived but as part of different solutions sets– if and only if a is true and b is not true (which is actually the case since a is the only fact provided). Note that the connective *not* in this case expresses default negation (contrary to classical negation), which means that the literal *not* b is assumed to hold unless b is derived. Rule (II.3) will therefore produce a new information into the system: either $\{c\}$ or $\{d\}$. Rule (II.4) simply says that if c holds, then e is derived. A derived information will hold as long as it doesn't conflict with any constraint of the system. Equation (II.5) is a *constraint* or *integrity constraint* that is used to reduce the set of solution. This constraint states that atoms a and d could not hold at the same time. Consequently, since a is a given fact to the input problem instance, any stable set that contains a must not contain d anymore. Therefore, the only stable solution for the above program is : $\{a, c, e\}$.

II.3 The Programming Methodology

The most common way to model logic programs for an ASP solver is to divide the program in three parts (*generate*, *define*, and *test*) following the *generate-and-test* paradigm [22]. To quickly explain this type of organization, let's consider the simple program of Figure II.1, representing an ASP-based encoding instance to solve the Hamiltonian path problem; which is to find a closed path that passes exactly once through each vertex of a directed graph. Line 1–3 of the program define the graph




```
1.  node(0).  node(1).  node(2).  node(3).
2.  edge(0,1). edge(0,2). edge(1,2). edge(1,3).
3.  edge(2,1). edge(3,0). edge(3,1). edge(3,2).
4.
5. {path(X,Y)}  ← edge(X,Y).
6.
7. reached(X)  ← path(0,X).
8. reached(X)  ← path(Y,X); reached(Y).
9.
10.            ← node(X); not #count{Y: path(X,Y)} = 1.
11.            ← node(X); not #count{Y: path(Y,X)} = 1.
12.            ← node(X); not reached(X).
```

Figure II.1: A directed graph (left), its corresponding ASP encoding and the modeling of the Hamiltonian path problem (right).

structure using atoms *node()* and *edge()*. For example: *node(0)*, *node(2)*, and *edge(0,2)* define two nodes of the graph and the edge between them. Line 5 is the *generate* part. It

aims at producing the different potential answer sets of the logic program from the initial facts. The rule of Line 5 asserts that any edge of the graph could potentially belong to a Hamiltonian cycle. Line 7–8 represent the *define* part, which basically consists of "Prolog-style" rules defining auxiliary predicates to be used in constraint rules. The rules stipulate that a Hamiltonian cycle starts at node 0 (Line 7) and any node that is (recursively) reachable from node 0 should be included in the path (Line 8). The last lines (10–12) represent the *test* part, where "undesirable" answer sets are eliminated. To avoid passing through a given node more than once, the in- and out-degree of nodes in a Hamiltonian path are constrained to 1 (Line 10–11). In Line 12, another constraint rule is used to eliminate any configuration with isolated nodes. Our proposed model will integrate these three different parts, but not necessarily in the given order.
Through an optimization statement ([34]), a fourth part could be added to our model to generate "optimal" solutions regarding predefined synthesis objectives. Such a statement is used to select among all the stable sets the one that is optimal. It has the form of statement (II.6) where *opt* is either *maximize* or *minimize*, V_i represent literals with their associated weights w_i and priorities p_i. Statement (II.6) means that the optimal answer set is the one with the maximal (or minimal) sum of weights (if all the p_i are equal). If different priorities are specified ($p_i \neq p_j$), then the answer set with the maximal (or minimal) sum of weights relative to the highest priority will be selected (with $p_0 > \ldots > p_n \in \mathbb{N}^+$).

$$\#opt\ \{\ w_0@p_0 : V_0;\ w_1@p_1 : V_1;\ \ \ldots;\ w_n@p_n : V_n\ \} \tag{II.6}$$

As an example, if for every edge *(X,Y)* of the graph in Figure II.1, a cost C is given $[cost(X, Y, C)]$, then statement (II.7) could be used to select the Hamiltonian path with the minimum cost. Note that the priority on weight C has been omitted since there is only one objective.

$$\#minimize\ \{\ C, cost : cost(X, Y, C),\ path(X, Y)\ \} \tag{II.7}$$

III Architecture Synthesis for Reconfigurable Systems-on-Chip

III.1 Introduction

Enabling self-coordination of nodes in a distributed camera network is imperative to guarantee system effectiveness. In fact, streaming all the data to a central location for analysis, as is the case in current CCTV-Based surveillance systems, is no longer an optimal solution given the real-time constraints and the complexity of processing information coming from multiple sources. In network of smart embedded cameras, system effectiveness is guaranteed by equipping each node with sufficient processing and reasoning power to dynamically analyze events in its field of view (FoV) and to simultaneously communicate with other peers in the network in order to realize a comprehensive coverage. This approach improves the system scalability and allows real-time tracking of events of interest among cameras, without relying on a central server. Nevertheless, designing such autonomous and collaborative vision nodes is very challenging due to the many and sometimes conflicting design objectives that must be addressed simultaneously: low latency, low power, low cost, high-performance, high quality, high throughput, etc. Additionally, operating multiple cameras instead of one requires more processing power and communication bandwidth, which are limited resources in practical distributed networks. Therefore, addressing effectiveness in smart camera networks amounts to tackle the issue of optimizing resource utilization when designing the camera architectures; a task that cannot be done manually, but which requires the use of an automatic synthesis methodology.

A distributed camera network is heterogeneous per nature; meaning that it is made up of nodes with various processing (CPUs, memories, buses) and communication (wireless and wired) capabilities. In this chapter, we focus on the synthesis of camera architectures to be used in such dynamic network environments considering the resource limitations on the computing infrastructures. This challenge is described as a combinatorial optimization problem with the objective of finding an optimal architecture for each camera node such as to minimize the resource utilization. Several attempts to provide a solution to such problems have been proposed in the literature. Mapping heuristics such as simulated annealing [35] or dynamic programming [36] have been used for the design of optimal

systems-on-chip (SoC) –a combination of hardware and software systems on a single chip. The use of Integer Linear Programing (ILP) for architecture synthesis of Multi-processor System-on-chip (MPSoC) has also been investigated [37, 38]. However, these works either were limited to produce sub-optimal solutions because of the pre-constrained design space or simply led to intractable problem instances for large systems.

This chapter presents a novel off-line encoding methodology based on an analytical approach for system-on-chip synthesis of smart cameras. Considering the resource limitations on embedded systems, the proposed approach improves upon existing methodologies by leveraging the network structure to optimize the architecture of the computing infrastructures. The synthesis uses Answer Set Programming (see Chapter II) to capture the network environment at start-up in terms of available tasks, resources, and the camera topology. A systematic mapping between the set of tasks and the available resources is then performed with the goal of producing optimal SoCs with regard to minimum system latency, power consumption, and resource utilization. An illustration of such a mapping function is shown in Figure III.1. Here, the SoC for a camera is generated

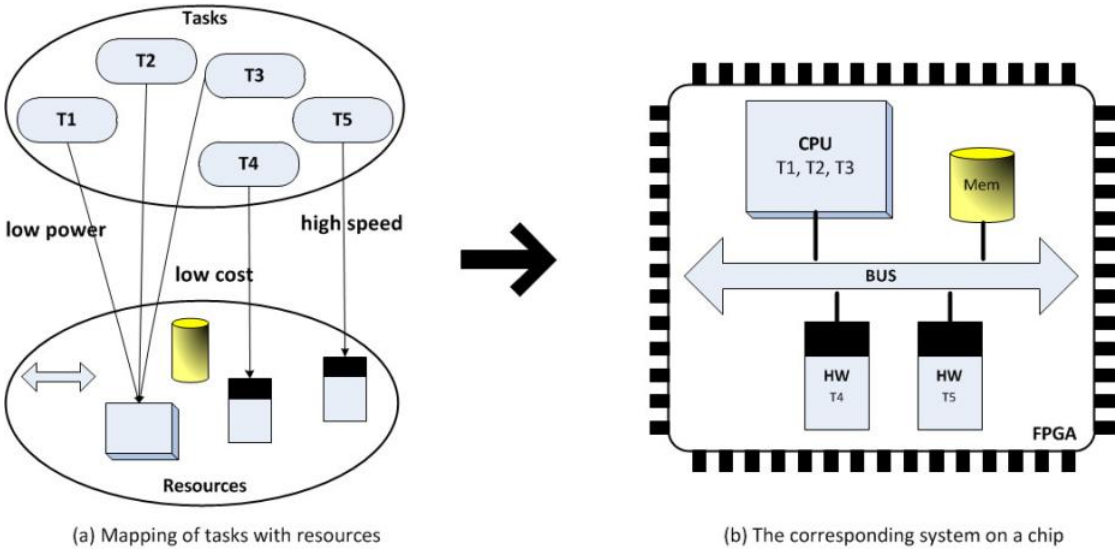


Figure III.1: A system-on-chip generation through mapping of low-level tasks with hardware resources.

through an automatic mapping between elements of the set of tasks (up left) and elements of the set of resources (down left) considering predefined synthesis objectives. A mapping relation defines how a specific task will be implemented, either as a software operation inside the processor (T1, T2, and T3) or as a hardware implementation

synthesized in low-level logic (T4 and T5). Given that the number of mapping configurations –also known as the design space– can be very huge, this synthesis approach necessitates the use of an exact methodology that should be able to cope with large problem instances in an acceptable amount of time.
Results from experiments show that the proposed method has a linear execution time and is able to synthesize all the architectures in a network of 10 cameras –implementing up to 10 tasks each– in less than 40s. Moreover, performing the synthesis simultaneously for all nodes of a network makes it possible to optimize the global utilization of communication resources. This optimization is achieved by first computing the set of *communicating cameras* in the network, called the Camera Communication Graph (CCG). This graph indicates all pairs of cameras that could potentially exchange information at runtime. Then, the CCG is used to devise the minimum amount of communication tasks/modules to be instantiated on each camera node during architecture synthesis. Our experiments show that the use of CCG could help save up to 40% of power and reduce the amount of communication resources in a network by almost 30% without significant performance degradation.
The rest of this chapter is structured as follows: In section III.2, the motivations behind our synthesis approach are presented. Section III.3 clearly formulates the problem that is addressed in this chapter. Related work with regard to synthesis of application-specific architectures are discussed in Section III.4. In Section III.5, the proposed ASP-based encoding model for the automated design of a system-on-chip is presented where emphasis is put on optimizing communication resources. Experimental results are provided in Section III.6, followed by concluding remarks in Section III.7.

III.2 Motivation

The expectations on the target systems-on-chip are: 1) to be fast and flexible enough to support all dynamics in a distributed network environment and 2) to minimize resource utilization in order to fit on the available embedded infrastructures and operate for longer periods of time. This section highlights the challenges to tackle in order to design such computing systems and provides the motivations behind our proposed design methodology.

III.2.1 Obstacles to Communication in a Distributed Network

Obstacles can get in the way of effective communication in an heterogeneous camera network where data exchange usually takes place on a peer-to-peer basis. In such a network, communication among nodes is influenced by two factors: the nature of cameras and the cameras' pose.

The *nature of a camera* is defined here regarding its communication capabilities, which could be *wired*, *wireless*, or even a combination of both (*wireless/wired*). In distributed systems, this heterogeneity increases the complexity of peer-to-peer communications given that two cameras, even geographically close, would not be able to communicate unless they have similar capabilities (here we exclude the possibility of using any off-the-shelf network adapters/bridges devices).

The camera pose –a combination of both cameras' position and orientation– is an important factor in optimizing communication in distributed networks that has already been subject of intensive research [39]. In fact, surrounding physical obstacles, such as walls, buildings, or trees could seriously mitigate communication or tracking handover between cameras in a network and impact the system performance.

As a proposed solution to these issues, we aim to answer the question of *how we can leverage the knowledge gained from the network topology in order to design efficient camera nodes that will improve the overall utilization of communication resources.*

III.2.2 Resource Limitations in Smart Camera Networks

Resource limitation is a typical problem in multi-camera networks [40]. Smart camera networks are basically heterogeneous embedded devices, highly constrained in terms of processing capabilities, available energy, and bandwidth resources. Camera sensors in such systems produce a huge amount of data that has to be manipulated on site. Processing such data in real-time requires complex computer vision operations, which are usually designed for workstations where energy constraints are minimal or non-existent. In contrary, distributed embedded cameras are generally battery-powered with a limited lifetime. Moreover, because of energy efficiency and size considerations, embedded cameras are designed with limited hardware capabilities (CPUs, memories, caches, and buses). Therefore, to operate for longer periods of time, smart power management mechanisms and cost-effective utilization of resources are required.

III.2.3 A Reconfigurable Computing System for Distributed Networks

A smart embedded camera combines video sensing, image processing, and communication within a single chip. The computations on such a complex system can efficiently be handled using a combination of hardware and software. In such a decomposition, while the hardware handles the most computationally demanding tasks, the software takes care of the control parts. By using Field Programmable Gate Arrays (FPGA) as the main processing component, complex computations can be directly implemented in hardware, while the control part are carried out by an embedded processor. The advantages of a FPGA-Based system could be summarized as high performance, high throughput, low cost, and low power. In terms of performance, FPGAs provide the necessary processing power to implement the complex, real-time image processing functions required throughout a distributed network. Moreover, FPGAs offer a low development time and more flexibility (instead of dedicated processing like ASICs) through the combination of hardware and software on a single chip and the possibility of restructuring the hardware at runtime.

III.2.4 An exact Methodology for the System-on-Chip Synthesis

Designing application-specific systems-on-chip for embedded smart cameras is a very complex undertaking that cannot be handle manually because of the various, sometimes conflicting design constraints and objectives that must be simultaneously addressed: system latency, system size, power consumption, system throughput, etc. For example, decreasing the latency of a battery-powered system, will usually increase its power consumption and therefore reduce the operation time. Each of the aforementioned parameters represents a dimension in the design space that has an impact on the overall system performance. The design space of such systems is therefore huge, requiring means to automatically optimize design parameters so as to facilitate wide and disciplined explorations. The complexity resulting from modeling the corresponding system necessitates the use of exact methodologies and optimization heuristics to cope with excessively long runtime, especially for large problem instances.

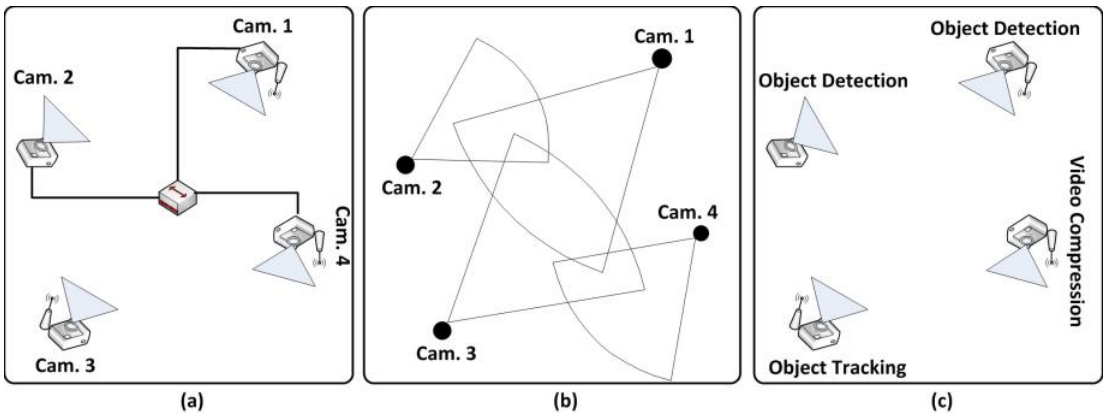


Figure III.2: An Heterogeneous network of four smart cameras. (a) - The physical network inter-connection. (b) - Camera overlapping field-of-views. (c) - Computer vision tasks implemented on each node.

III.3 Problem Formulation

This section presents a concise formulation of the addressed problem. We start by defining some terminologies that will be used throughout the chapter.

III.3.1 Definitions

For a practical formulation, let's consider the network scenario depicted in Figure III.2, in which a set of 4 smart cameras is placed on a 2D space. This network consists of various camera types: wired (node 2), wireless (node 3) and wireless/wired (node 1 and 4). Each of these nodes implements a specific computer vision operation (see Figure III.2(c)), that will be referred in this Dissertation as a *high-level* task, as it could further be decomposed in a series of *low-level* operations (convolution, thresholding, or image filtering). Concepts borrowed from graph theory will be leveraged to capture the *global information* about the network environment, such as the topology and the camera orientation, as well as the *local information* about individual nodes, such as the available resources on a camera and the set of low-level tasks to be implemented.

Definition III.3.1 (Task Graph)**.** A *Task Graph* of a given camera node, denoted $G_T = (V_T, E_T)$, where V_T is the set of tasks and E_T the communication among these tasks, is a graph representing the set of computation and communication tasks to be executed inside the camera and their interdependence.
An edge $e = (i, j) \in E_T$ represents the internal communication between the nodes v_i and

$v_j \in V_T$, which can express a data dependency between two computations. Originally the task graph of each camera will only consist of computation tasks. The communication tasks will be added only after generation of the Communication graph.

Definition III.3.2 (Architecture Graph)**.** The *Architecture Graph* of a camera is a graph that captures the available resources on the camera.
It is denoted $G_A = (V_A, E_A)$, where V_A represents the available resources (CPU for software tasks and hardware slots for hardware tasks) and E_A is the set of communication links among the nodes.

Definition III.3.3 (Camera Interconnection Graph)**.** We define the *Camera Interconnection Graph* (CIG) of a network as the graph that captures the physical interconnections among all the different cameras in the network.
The CIG is denoted $G_{CIG} = (V_{CIG}, E_{CIG})$, where V_{CIG} represents the set of cameras and E_{CIG} the interconnection (wired or wireless) between camera pairs. A link $e_{ij} \in E_{CIG}$ between $v_i, v_j \in V_{CIG}$ indicates that both cameras are within the same transmission range (i.e either they are connected to the same network switch or they are within the same wireless range).

Definition III.3.4 (Camera Orientation Graph)**.** The *Camera Orientation Graph* (COG) of a network is the graph that captures the overlapping field of views among the different camera nodes.
It is denoted $G_{COG} = (V_{COG}, E_{COG})$, where V_{COG} represents the set of nodes in the network. An exiting link $e_{ij} \in E_{COG}$ between two nodes $v_i, v_j \in V_{COG}$ means that both cameras share an overlapping FoV.

Definition III.3.5 (Camera Communication Graph (CCG))**.** The *Camera Communication Graph* (CCG) of a network is defined as the graph capturing all communicating pairs of cameras in the network.
Two cameras are said to be *communicating* if they share an overlapping FoV and are not bounded by any communication constraints as explained in Section III.2.1. The CCG is denoted $G_{CCG} = (V_{CCG}, E_{CCG})$, where V_{CCG} represents the set of cameras. A link $e_{ij} \in E_{CCG}$ between two nodes $v_i, v_j \in V_{CCG}$ symbolizes an inter-camera communication, meaning that all conditions are actually fulfilled for v_i and v_j to communicate. This assumption will be materialized during synthesis by adding new communication tasks to

the task graphs of both nodes. As a consequence to this, each camera node will only instantiate the minimum amount of communication tasks that is necessary to communicate.

Definition III.3.6 (Adjacency Matrix)**.** Given a graph $G = (V, E)$ with n nodes, the *Adjacency Matrix* of G, denoted $M_G \in N^{n \times n}$, is a matrix that indicates which nodes of the graph G are adjacent to which other nodes.
The Adjacency Matrix is an abstract way of representing any graph and is computed using the following formula:

$$M_G[i,j] = \begin{cases} 1 \text{ if } e = (v_i, v_j) \in E, \text{ with } v_i \text{ and } v_j \in V \\ 0 \quad \text{otherwise.} \end{cases} \tag{III.1}$$

III.3.2 Problem Statement

We consider a distributed network $NET = (T, C, R)$ of n camera nodes, where for each node i, the available resources V_{A_i} (architecture graph) and the set V_{T_i} of low-level tasks implemented on i at startup (task graph) are known. We define:

1. $T = V_{T_1} \cup V_{T_2} \cup \ldots \cup V_{T_n}$ as the set of all computation tasks in NET,
2. $C = V_{C_1} \cup V_{C_2} \cup \ldots \cup V_{C_n}$ as the set of communication tasks in NET; with V_{C_i} representing all communication tasks on camera node i that have been generated using the CCG, and
3. $R = V_{A_1} \cup V_{A_2} \cup \ldots \cup V_{A_n}$ as the set of all resources available in NET.

The goal of the architecture synthesis is to define an automatic mapping and scheduling between the set of tasks $(T \cup C)$ and the available resources (R), such as to generate an optimized system-on-chip for each node in the network. Optimization aims at maximizing speed and throughput, while minimizing chip area and power consumption per camera node. This is similar to finding a surjective function $f : T \cup C \to R$ that makes a one-to-one association between elements from sets $T \cup C$ and R, such that design objectives (resources utilization, power consumption, latency, and throughput) are optimized. This is a non-deterministic combinatorial optimization problem that can result in a huge design space (set of solution candidates) even for problems with small size. In other words, even with small amount of tasks and fewer resources available per

camera, the set of possible mapping can become very huge and quickly intractable. The formulated problem can be seen as an extension of the multi-processor system-on-chip (MPSoC) optimization problem to a simultaneous optimization of interconnected MPSoCs in a heterogeneous network topology. Mapping and scheduling problems in MPSoCs have already been proven to be NP-Complete [41], which makes the complexity of the addressed problem to be at least NP-Complete.

III.4 Related Work

To model combinatorial optimization problems and finding optimal solutions, many research have investigated the use of Integer Linear Programming (ILP). In [42] and [43], a node placement problem to achieve effective coverage in distributed sensor networks has been reduced to an ILP problem instance, which is solved using a public-domain solver. In [37], Murali et al. leveraged ILP for designing a power-efficient application-specific crossbar architectures for MPSoCs, while [38] used an ILP model to automate the architecture synthesis for parallel programs on FPGA multi-processor systems. However, the main drawback of ILP is that it is not scalable to large problem instances. While exact, ILP-based high-level synthesis approaches suffer from the limitations imposed by the size of the problem. For large systems, the problem instance becomes easily intractable and leads to indefinite synthesis time.
Most of the existing work on architecture synthesis for multi-processor systems focus more on processor allocation and on-chip communication. To date, very few work has provided a means for the systematic mapping of an application in a hardware/software system [44]. A Branch and Bound approach for solving the task mapping problem in a multi-processor systems has been proposed in [45]. Mapping algorithms such as dynamic programming [36], simulated annealing [35], evolutionary algorithms [39], and application-specific heuristics [46] have been used for general purpose system-on-chip design problems. While these approaches take steps in the right direction by eliminating tedious manual explorations, they are limited for reconfigurable system-on-chip designs because they do not consider cross-effects between the subspaces as they only target specific dimensions, thus producing sub-optimal solutions.
Regarding the optimization of communication resource in distributed networks, a vast majority of research has considered the camera selection approach. In this methodology,

the resource optimization problem is tackled through dynamic selection of camera subsets that will efficiently handle all monitoring tasks at runtime. The idea here is to minimize the amount of active cameras in the network by optimally assigning tasks to a minimum set of nodes according to a certain quality of service without degrading the overall system performance. In [47], three camera selection approaches are presented (centralized, distributed, and proprioceptive) wherein the task of finding the optimal camera subset is either solved on a dedicated node or distributed over a set of collaboratives cameras. [48] presents a resource-aware task assignment approach in a visual sensor network using a marked-based object handover mechanism. In [49], Shen and Hornsey propose a camera selection scheme based on evaluation of local and global quality of view (QoV) metrics. The local metric evaluates the root mean square error of the ellipse fitting the target object and determines how good a camera can track the object; while in the global QoV, the camera subset with the best visual hull volume of the target object is assigned the tracking task. Generally, the major drawback in the camera selection approach is the additional cost (time and hardware resource) spent to dynamically compute the set of active cameras. When the network is overloaded, this cost could negatively impact on the performance of the system. Additionally, these approaches are limited in the fact that they rely on homogeneous networks where all cameras are of same nature.

III.5 Proposed Approach

In this chapter, we propose a vertical design flow for implementing optimal system-on-chips that consists in four phases, sketched in Figure III.3. The presented flow successively covers (i) the *specification of the network environment*, (ii) the *global optimization*, (iii) the *application mapping*, and (iv) the *architecture determination*, the result of which is an abstract description of the architectures for each camera of the network with the optimal amount of inter-camera communication. The input to the design flow is a formal description of the network which captures all computation tasks and infrastructures available in the distributed environment at startup, as well as the network topology. To increase design productivity, computation and communication tasks are available as Intellectual Property (IP) cores, whose information regarding their costs and constraints is provided. The description of low-level infrastructures indicates how much resources (processors, memories, buses, logic elements) are available in the network.

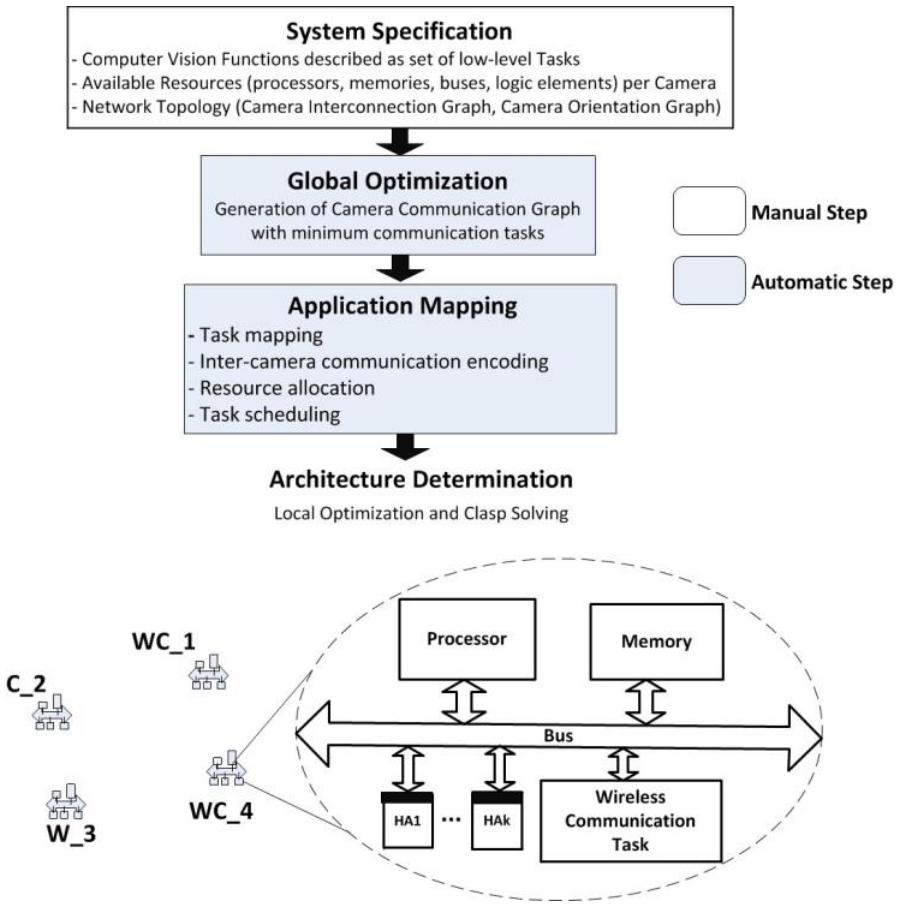


Figure III.3: An analytical architecture synthesis flow applied on the network of Figure III.2.

Starting from the system specification, as can be seen on Figure III.3, an instance of a logic problem is formulated using Answer Set Programming (ASP), then subsequently refined and solved using existing solvers. The generated solution, obtained after the *architecture determination* step, represents an abstract description of systems-on-chip that indicates, for each camera, the number of allocated memories and hardware resources, the task mapping and their schedule on processors. This final architecture could be further passed to existing CAD tools to generate the configuration bitstream, but this last part is not covered in this Dissertation.
Before presenting in detail the proposed ASP encoding model, we next explain its structure.

III.5.1 The Structure of the Proposed ASP Model

In ASP, it is custom to provide a uniform problem definition. Following this methodology, our proposed model will be structured in three different parts, to allow a greater flexibility and reusability: First, the problem description consisting of facts representing the set of beliefs that are held true at system startup, such as the network environment (see Figure III.4), the resources available on the different camera nodes, or the composition of the processing tasks implemented on the nodes (see Figure III.5). Second, a summary of meta-information for all computational functions available in the system, such as the costs (area, power, speed) of Intellectual Properties used to implement the high-level tasks on camera nodes (see Figure III.7). Third, the solver model itself –following the *generate-and-test* paradigm (see Chapter II)– which provides all the necessary rules to devise the answer sets. Each of these parts will be developed in the following sections.

III.5.2 System Specification

The system specification is captured by means of task graphs, architecture graphs, interconnection graph and orientation graph. Each node of the heterogeneous network has one task graph and one architecture graph, but the interconnection graph and the orientation graph are shared by all cameras. The task graph represents computations to be done in a single camera at startup. This set of tasks is defined by the initial setup of the camera. Modification on the camera behavior at runtime is triggered by changes in the environment, which might introduce new tasks in the system and require the task graph to be updated. This dynamic configuration is not addressed in this chapter, but will be covered in detail in Chapter IV.

Global Specification

The global specification captures the topology of the heterogeneous network. ASP facts are used to identify all cameras in the network and describe their interconnection and orientation, yielding an ASP instance of a logic program that will be used during global optimization to generate a minimum set of communication tasks. Figure III.4 shows an ASP-based modeling sample of the network environment presented in Figure III.2. On the logic program, atom $cam(M)$ defines a camera node M; $wired(M, \_)$ and $wrless(M, \_)$ indicates all the wired and wireless connections of M in the network, while $camFov(M, \_)$

```
% Camera Nodes of the Network
 cam(1).      cam(2).      cam(3).      cam(4).

% Physical Inter-connections among Nodes (CIG)
 wired (1,2).    wired(1,4).    wrless(1,3).    wrless(1,4).
 wired(2,4).
 wrless(3,4).

% Overlapping Field of Views (COG)
 camfov(1,2). camfov(1,3). camfov(1,4).
 camfov(2,3).
 camfov(3,4).
```

Figure III.4: Part of the ASP model describing the Network environment in Figure III.2.

lists all nodes sharing an overlapping field-of-view with camera M.

Local Specification

In the local specification, all low-level tasks implemented on a camera node are described as well as the type of available resources. Examples of such tasks are: thresholding, edge detection, background subtraction, color conversion, or image compression. The description of tasks is captured using a task graph, while an architecture graph is used to capture the resources (processors, memories, logic elements, buses). An ASP instance illustrating both graphs is shown in Figure III.5. In this example, atoms $node(M, N_{id}, T)$ and $edge(M, E_{id}, T, T_i, T_{hroughput})$ represent a task T implemented on a camera M and its dependency T_i in the processing chain (with N_id and E_id being a unique identifier for the node and the edge respectively). T is the data throughput on the edge. Atom $waitFor(M, N_1, N_2)$ specifies the synchronization constraints among tasks with node ID N_i of a camera, while $place(M, RES_{type}, RES_{size})$ and $dom(M, IP_{type}, IP_{inst})$ respectively capture for each camera, the total amount of a given resources and the list of IP available. In order to compute an optimal schedule and minimize overall execution time on a camera, two special nodes (V_s and V_e) with no delay are introduced on its task graph: $node(M, V_s, start)$ and $node(M, V_e, end)$. V_s is the task *start* and is incident to all nodes without predecessors in the task graph and all nodes without successors are incident to V_e, the task *end*. Each computation path on a task graph starts with V_s and ends with V_e. At startup, there is no communication task included on the task graphs. These will be generated after the *global optimization* step and added later to task graphs during

```
%%%%% Camera Node #4  %%%%%%
% Local Task definitions
node(4, 0, start).       node(4, 1, im_acq).
node(4, 2, encoder).   node(4, 3, end).

% Communication between tasks
edge(4,4,0,1,1).  edge(4,5,1,2,2).

% Synchronization
wait_for(4,1,2).

% Available Resources
place(4, slices, 64000).   place(4, hw_proc, 2).
place(4, dsp, 64).          place(4, bram, 100).

% Available IP Cores
dom(4,proc,1..1).     dom(4,pu_im_acq,1..1).  dom(4,ddr,1..1).
dom(4,axi_lite,1..1).  dom(4,sdi_comp,1..3).   dom(4,pu_usb,1..1).
dom(4,pu_encoder,1..1).
```

Figure III.5: Part of the ASP model describing the computation tasks and available resources on camera node 4 in Figure III.2. The task graph describes the high-level processing implemented by the camera which consists of an image encoding.

synthesis.

III.5.3 Global Optimization

As shown in Section III.2, an design methodology that does not consider the network environment in a distributed setup is less optimal because of potential obstacles to communication. Using the network description of the previous step, the objective in global optimization is to derive the minimum set of communication tasks for all nodes in the network that will maintain the expected communication traffic. This is done by computing the camera communication graph (CCG), which captures all communicating pairs in the network and indicates the means through which they will be communicating at runtime, whether it be wired or wireless.

To devise the CCG, both the Interconnection Graph (CIG) and the Orientation Graph (COG) are merged, as illustrated in Figure III.6. A node C_i, on the Figure indicates a wired node type, W_i a wireless type, and WC_i a wireless/wired node. The merging process is expressed by equation (III.2) and corresponds to an element-wise matrix

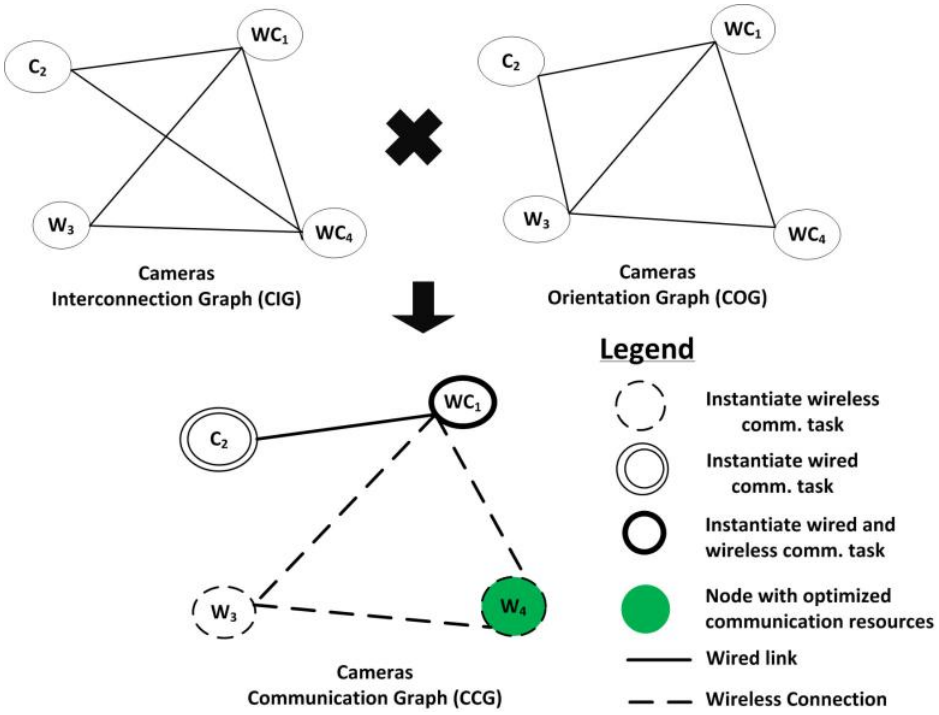


Figure III.6: Generation of the camera communication graph.

multiplication of the adjacency matrices of CIG and COG.

$$M_{G_{CCG}}[i,j] = M_{G_{CIG}}[i,j] \times M_{G_{COG}}[i,j] \tag{III.2}$$

where $M_{G_{CCG}}$ (resp. $M_{G_{CIG}}$ and $M_{G_{COG}}$) represents the adjacency matrix of the graph G_{CCG} (resp. G_{CIG} and G_{COG}).

Put it simply, two nodes are connected in the CCG iff they are connected in both the CIG and the COG. The ASP-rules to encode equation (III.2) are given by:

$$camConn(M,N) \leftarrow wired(M,N). \tag{III.3}$$

$$camConn(M,N) \leftarrow wrless(M,N). \tag{III.4}$$

$$camComm(M,N) \leftarrow camConn(M,N), camFov(M,N). \tag{III.5}$$

Rules (III.3) and (III.4) identify as connected, two cameras M and N linked either by wired or by wireless. This identification helps derive the set of communicating nodes in the network (rule (III.5)), which forms the CCG.

Once the CCG has been generated, we next compute the optimal set of communication tasks to be implemented on each camera node considering its nature. If a node M of the

CCG is a wired type, then a (low-level) task implementing a wire protocol will be generated:

$$camWired(M) \leftarrow wired(M,N),\ \#count\{M : wrless(M,\_)\} = 0,$$
$$\#count\{M : wrless(\_,M)\} = 0. \tag{III.6}$$

$$1\{node(M,ID,com\_wired)\}1 \leftarrow camWired(M),\ ID = M * NT_{max} + 1,$$
$$\#count\{M : camComm(M,\_)\} \geq 1. \tag{III.7}$$

Rule (III.6) uses the aggregate *count* to identify a wired camera as a one having no connection in the network but wired (i.e number of wireless neighbors = 0), while Rule (III.7) generates exactly one new communication task iff the wired camera node is at least connected to one other node in the CCG. The variable ID represents a unique identifier to the new local task obtained using the camera number and the input constant NT_{max}, which is the maximal number of tasks per camera. Likewise, if node M is a wireless camera, then a task implementing a wireless communication protocol will be generated:

$$camWrless(M) \leftarrow wrless(M,N),\ \#count\{M : wired(M,\_)\} = 0,$$
$$\#count\{M : wired(\_,M)\} = 0. \tag{III.8}$$

$$1\{node(M,ID,com\_wireless)\}1 \leftarrow camWrless(M),\ ID = M * NT_{max} + 2,$$
$$\#count\{M : camComm(M,\_)\} \geq 1. \tag{III.9}$$

Regarding a wireless/wired camera type, normally two low-level tasks implementing a wireless and a wire protocol respectively should be created. But to avoid wasting resource, if all adjacent nodes to M in the CCG are the same type, then only one task is necessary:

$$camWrlessWired(M) \leftarrow cam(M),\ \#count\{M : wrless(M,\_)\} \geq 1,$$
$$\#count\{M : wired(M,\_)\} \geq 1. \tag{III.10}$$

$$neighbor(M,N_G) \leftarrow cam(M),\ R = \ \#count\{M : camComm(\_,M)\},$$
$$L = \ \#count\{M : camComm(M,\_)\},\ N_G = L + R. \tag{III.11}$$

$$neighborWrlessWired(M, N_G) \leftarrow camWrlessWired(M),\ neighbor(M, N_G). \quad \text{(III.12)}$$

$$1\ \{node(M, ID, com\_wired)\}\ 1 \leftarrow neighborWrlessWired(M, N_G),\ N_G = \#count\{M : wired(M, \_)\},\ ID = M * NT_{max} + 1. \quad \text{(III.13)}$$

Rule (III.10) identifies camera M as a wireless/wired node if it has at least one wireless and one wired connection, while (III.11) counts the number of adjacent nodes to M in order to devise its neighborhood N_G. If N_G is equal to the number of wired nodes connected to camera M, meaning that every node in the neighborhood is a wired type, then a wired communication task is generated for camera M (Rule (III.13)). Similarly, if all adjacent nodes to M are wireless, then only one task implementing a wireless protocol will be generated:

$$1\{node(M, ID, com\_wireless)\}1 \leftarrow neighborWrlessWired(M, N_G),\ N_G = \#count\{M : wrless(M, \_)\},\ ID = M * NT_{max} + 2. \quad \text{(III.14)}$$

Now, if a wireless/wired node is connected to both a wired and a wireless node in the CCG, then it must implement both types of protocol. This implies that two tasks (one for wired and one for wireless communication) would be generated.

$$1\ \{node(M, ID_1, com\_wired)\}\ 1 \leftarrow neighborWrlessWired(M, N_G),\ N_G \neq \#count\{M : wired(M, \_)\},\ ID_1 = M * NT_{max} + 1. \quad \text{(III.15)}$$

$$1\{node(M, ID_2, com\_wireless)\}1 \leftarrow neighborWrlessWired(M, N_G),\ N_G \neq \#count\{M : wired(M, \_)\},\ ID_2 = M * NT_{max} + 2. \quad \text{(III.16)}$$

Rules (III.15) and (III.16) simply check if the number of wired nodes connected to a wireless/wired camera is different than its neighborhood and then generates two different tasks implementing a wire and a wireless communication protocol.

III.5.4 Application Mapping

Upon generating the communication tasks, the mapping of all low-level tasks in the network with available resources can now be implemented, with the goal of minimizing

the cost (resource usage, power consumption, and latency) of the generated system-on-chips. In [44], an ASP-based encoding model was proposed for rapid prototyping of image processing application. However, this model was implemented for single objective optimization problems and considered only the synthesis of a single architecture at a time. The mapping methodology proposed in this chapter is intended for multi-objective optimization problems and performs the synthesis simultaneously for all nodes of a network considering inter-camera relationships. Our approach yields a better optimization of communication resources since the global view of the network is leveraged. To allow a high flexibility and reusability, the mapping process leverages a *Task description model* and a *Solver model* encoded independently from the model that describes the network environment. The *Task description model*, as illustrated in Figure III.7, provides information about the available implementations of computational/communication modules for the local tasks and edges of the camera nodes.

```
% IP Components with their Name & Type
component(proc, comp).  component(pu_torgb,comp).
component(pu_sobel,comp). component(pu_usb, com_wired).
...
% Tasks and Throughput implemented by Components
implement(proc, torgb, 16). % Task implemented in software
implement(pu_torgb, torgb, 1). % Implemented in Hardware
...
% Units that could be connected to a bus
group(proc, bus_con).      group(ddr, bus_con).
group(pu_usb, bus_con).   group(pu_sobel, sdi).
...
% Available Resources
resource(bram).        % Block RAMs
resource(hw_proc).   % Hardware processor
resource(slices).       % Logic Computing Units
resource(dsp).          % Dedicated Units for Adds and Mults
...
% Explicit utilization
uses(proc, hw_proc, 1). uses(pu_torgb, slices, 119).
uses(pu_torgb,bram,2). uses(pu_torgb, power, 31).
...
```

Figure III.7: Example of a Task description model.

Here we follow a platform-based approach in which computational and communication

modules are made available as Intellectual Property (IP) in a library. For each local task on any task graph of the network, one or more IPs must exist that implement a particular algorithm either as software function or as dedicated hardware module with associated speed, cost, and power consumption provided. In the Description model of Figure III.7, atoms $component(IP_{name}, IP_{type})$ and $implement(IP_{name}, IP_{task}, IP_{throughout})$ define different instances of an IP and specify whether they are implemented as software functions or as hardware macros. The different kind of resources in the system are indicated by $resource(RES_{type})$, while atom $uses(IP_{name}, RES_{type}, Q_{uantity})$ explicits the utilization of a resource instance by an IP in terms of hardware logic elements and power consumption. Finally, to implement the architecture synthesis a *solver model* is used. The model specifies all necessary ASP rules to successively perform allocation of hardware resources, mapping of local tasks with available resources, scheduling of individual operations on each camera of the network, and generation of the final optimal architectures with regard to latency, power, and area minimization.

Mapping Local Tasks to Processing Units

the mapping of low-level tasks assigns each node of a task graph to exactly one processing unit, which can be the processor or a dedicated hardware. This process is simultaneously done for all cameras of the network. For each instantiated component C on a camera M and each local task V, an atom $map()$ is defined:

$$1\ \{map(M, V, C, K) : dom(M, C, K)\}\ 1 \leftarrow node(M, V, O). \tag{III.17}$$

where atom $node(M, V, O)$ captures the task O and the parameter K indicates a particular instance of the IP C that realizes O. Rule (III.17) means that for each task V on the task graph of camera M, the sum of mapped components on M must equal 1, insuring that each local task is mapped to exactly one computational unit.

Capturing Inter-Camera communication

To capture the communication between two cameras, the communication tasks generated during global optimization are added to their respective task graphs. During this adding process, new edges are created to emulate input and output connection flows as shown in Figure III.8. For an input flow, i.e data coming from another camera, the communication

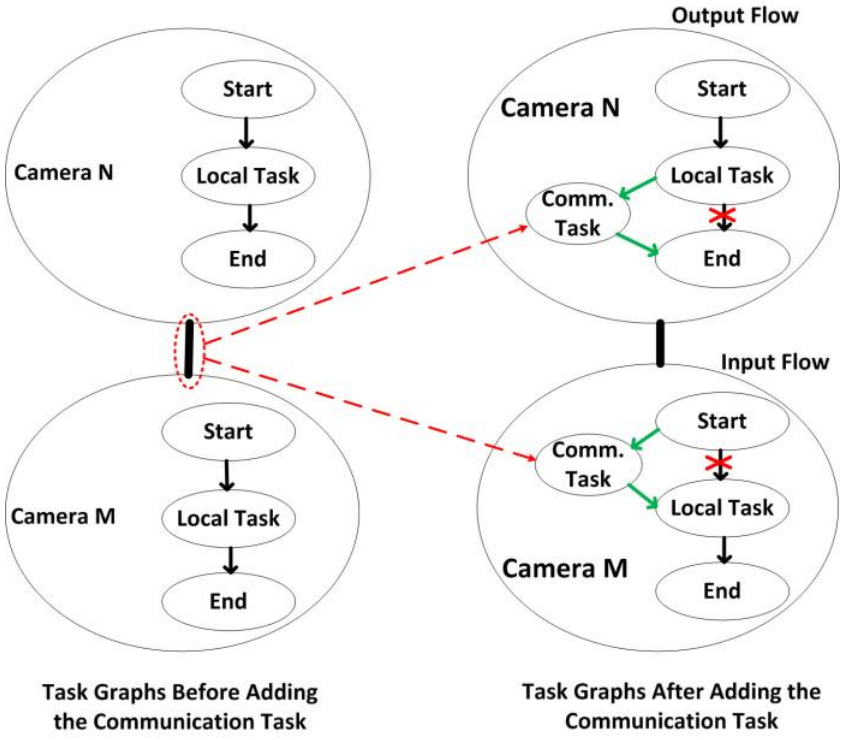


Figure III.8: Modeling inter-camera communication during synthesis.

task is inserted between the task *start* and the first local task –node with the lowest identifier– in the processing chain of the receiving camera:

$$minNode(M,V) \leftarrow cam(M), V = \#min\{O : node(M,O,X),\ X \neq start\}. \quad \text{(III.18)}$$

$$1\ \{edge(M,E,V_s,V_C,1)\}\ 1 \leftarrow camComm(N,M), V_C = M * NT_{max} + 1, E = V_C, \\ node(M,V_s,start), node(M,V_C,com\_wired). \quad \text{(III.19)}$$

$$1\ \{edge(M,E,V_C,V_O,1)\}\ 1 \leftarrow camComm(N,M), V_C = M * NT_{max} + 1, \\ minNode(M,V_O), node(M,V_C,com\_wired). \quad \text{(III.20)}$$

Rule (III.19) captures the connection between V_s, which is the node *start*, and the communication task V_C on camera M. Then Rule (III.20) will connect V_C to the first processing task on M, whose ID is given by atom $min\_node(Cam_{id}, Node_{id})$.
To capture an output flow, the communication task V_C on the sending camera is inserted between the last local task –node with the highest identifier– and the node *end* (Rules (III.21)–(III.23)). This follows the same principle as previously described.

$$maxNode(N,V) \leftarrow cam(N),\ V = \#max\{O : node(N,O,X),\ X \neq end\}. \quad \text{(III.21)}$$

$$1\ \{edge(N,E,V_O,V_C,1)\}\ 1 \leftarrow camComm(N,M), V_C = N * NT_{max} + 1, \\ maxNode(N,V_O), node(N,V_C,com\_wired). \quad \text{(III.22)}$$

$$1\ \{edge(N,E,V_C,V_e,1)\}\ 1 \leftarrow camComm(N,M), V_C = N * NT_{max} + 1,\ E = V_C, \\ node(N,V_s,end), node(N,V_C,com\_wired). \quad \text{(III.23)}$$

Mapping Edges to Local Communication Components

For an edge E of a task graph, which describes data dependency between two local tasks V_i (source) and V_o (sink) in a camera M, the following constraints need to be met:

$$0\ \{send(M,E,C_1,K_1,C_2,K_2)\}\ 1 \leftarrow edge(M,E,I,O,D), dom(M,C_1,K_1), \\ dom(M,C_2,K_2). \quad \text{(III.24)}$$

$$\leftarrow edge(M,E,V_I,V_O,D), dom(M,C,K), \\ send(M,E,C,K,C_1,K_1), \\ not\ map(M,V_I,C,K). \quad \text{(III.25)}$$

$$\leftarrow edge(M,E,V_I,V_O,D), dom(M,C,K), \\ send(M,E,C_2,K_2,C,K), \\ not\ map(M,V_O,C,K). \quad \text{(III.26)}$$

Rule (III.24) captures the data transfer between two tasks (connected by E) mapped on components C_1 and C_2. Constraint (III.25) insures that, if a source task V_I is mapped to a component C there must exist a component C_1 to which C sends data over edge E. Similarly, if a sink task V_O is mapped to a component C, there must exist a component C_2 which sends data to C over edge E (Constraint (III.26)).

Modeling Timing Requirements

Modeling the temporal behavior can be done by defining time values as discrete and finite set of possible time slots. In the proposed ASP model, each local task V is assigned exactly to one time slot T, as described by Rule (III.27):

$$1\ \{map(M,V,T) : time(T)\}\ 1 \leftarrow node(M,V,O). \quad \text{(III.27)}$$

where M is a camera of the network and $time(V_{alue})$ an atom representing the time slots available in the time domain, whose maximum value is given as a constant in our model.

For each task graph in the network environment, the task *start* (V_s) previously defined, is assigned the time slot 0 to insure that it will be the first to be scheduled:

$$\leftarrow node(M, V_s, start),\ \#count\{V_s : map(M, V_s, TX),\ TX > 0\} \geq 1. \tag{III.28}$$

The task *end* (V_e), which is always the last time to be scheduled, is assigned the total runtime/latency of the camera –only known after the mapping of the whole task graph (see Constraint (III.45)).

Modeling Local Scheduling

Finding a schedule for a task graph is subject to defining the right data dependencies among its nodes. Two nodes V_x and V_y are said to be data dependent if they are directly (Rule (III.29)) or indirectly (Rule (III.30)) connected on the task graph.

$$dep(M, V_x, V_y) \leftarrow node(M, V_x, O_1), node(M, V_y, O_2), edge(M, E, V_x, V_y, D). \tag{III.29}$$

$$dep(M, V_x, V_z) \leftarrow node(M, V_x, O_1), node(M, V_y, O_2), node(M, V_z, O_3), \\ edge(M, E, V_x, V_y, D), dep(M, V_y, V_z). \tag{III.30}$$

If a task V_y is data dependent from a task V_x on camera M, then V_y may not start before its predecessor V_x has completed his execution. The following rule captures this constraint:

$$\leftarrow map(M, V_x, T_x), dep(M, V_x, V_y),\ \#count\{V_y : map(M, V_y, T_y),\ T_y < T_x\} \geq 1. \tag{III.31}$$

Capturing System Speed

The speed of a camera system is determined by the speed of its system bus. Each task that is part of a processing chain on the camera will start producing data at a certain time T depending on its throughput T_p.

$$1\ \{workTp(M, V, T_p)\ :\ tp\_dom(T_p)\}\ 1 \leftarrow node(M, V, O),\ O \neq start,\ O \neq end. \tag{III.32}$$

$$taskAtTime(M, V, T) \leftarrow map(M, V, time), workTp(M, V, T_p), \\ edge(M, E, I, V, D),\ time(T),\ T < (T_p * D). \tag{III.33}$$

$$taskAtTime(M, V_s, 0) \leftarrow node(M, V_s, start). \tag{III.34}$$

Similar to time assignment, each local task of a task graph, other than *start* and *end*, is assigned to exactly one throughput slot T_p (Rule (III.32)). Using the throughput, Rule (III.33) estimates for each task, the time at which it would start producing data; with the start node being scheduled at time 0 (Rule (III.34)).
To evaluate the system bus load on camera M, we first compute the workload W that each task V puts on the bus at a time slot T:

$$workDiv(M, V, K, T, W) \leftarrow taskAtTime(M, V, T), map(M, V, proc, K), \\ workTp(M, V, T_p), W = T_p/tp. \quad \text{(III.35)}$$

where tp represents the internal bus capacity given as input parameter. Then, we derive the overall workload W_T on the bus at time T:

$$work(M, div, T, W_T) \leftarrow cam(M),\ time(T), \\ W_T = \#sum\{W_1 : workDiv(M, V, K, T, W_1)\}. \quad \text{(III.36)}$$

Finally, for each time slot T the sum of active bus transfers must always be smaller than the bus capacity tp. This is guaranteed by the following constraint:

$$\leftarrow time(T), tp(bus, T_p), work(M, div, T, W_T), W_T * T_p > tp. \quad \text{(III.37)}$$

Because traffic caused by a component is inversely proportional to its speed, the load of the bus can be used to define its speed.

III.5.5 Architecture Determination

The rules previously defined will be used as constraints while searching for a solution to the initial logic problem. Since many possible solutions may exist, we are interested only in optimal ones, which are those with minimum runtime, resource usage, and power consumption.
To compute the total resource usage on a camera M, we first look for the size S and power consumption P of every Intellectual Property I available on M:

$$size(M, I, R, S) \leftarrow uses(I, R, S),\ dom(M, I, K),\ R \neq power. \quad \text{(III.38)}$$

$$size(M, I, power, P) \leftarrow uses(I, power, P),\ dom(M, I, K). \quad \text{(III.39)}$$

Then, we narrow our computation to IPs implementing tasks that are part of the processing chain on M, which are basically tasks that have been mapped on hardware components (Rule III.40) or used for data transactions (Rule III.41):

$$use(M,C,R,S) \leftarrow size(M,C,R,S), dom(M,C,K), map(M,V,C,K). \tag{III.40}$$

$$use(M,C,R,S) \leftarrow size(M,C,R,S), dom(M,C,K), \\ \#count\{E : send(M,E,C,K,C_1,K_1), dom(M,C_1,K_1)\} \geq 1. \tag{III.41}$$

We finally generate the size of resources used by camera M (Rule III.42), as well as its power consumption (Rule III.43):

$$utilization(M,R,U) \leftarrow resource(R),\ place(M,R,O), \\ U = \ \#sum\{S : use(M,C,R,S)\}. \tag{III.42}$$

$$utilization(M,power,U) \leftarrow cam(M),\ U = \ \#sum\{S : use(M,C,power,S)\}. \tag{III.43}$$

Having estimated the resource utilization and power consumption for all kinds of mapping configuration, we can now perform a local optimization. It consists of selecting, for each camera node, the architecture/mapping configuration that would minimize runtime, area, and power.

$$\begin{aligned} &\#\text{minimize}\{ \\ &\quad T \ @\ 3, map : map(M,V_e,T),\ node(M,V_e,end); \\ &\quad U \ @\ 2, utilization : utilization(M,R,U),\ R \neq power; \\ &\quad P \ @\ 1, utilization : utilization(M,power,P)\}. \end{aligned} \tag{III.44}$$

Subject to constraints:

$$\leftarrow taskAtTime(M,X,T), node(M,V_e,end), \\ \#count\{V_e : map(M,V_e,T_e),\ T_e \leq T\} \geq 1. \tag{III.45}$$

$$\leftarrow place(M,R,U_1),\ \#count\{R : utilization(M,R,U_2),\ U_2 > U_1\} \geq 1. \tag{III.46}$$

The first objective in the first line of Rule (III.44) is the runtime. V_e is the task *end* defined earlier. It should be the last task to be scheduled on camera M (Constraint (III.45)) and therefore defines the overall system runtime of M. Each atom

$map(M, V_e, T)$ is weighted by its time slot number T. Because only one atom is true, the sum results in the time slot number of the end task and hence the total runtime of camera M. The second objective, minimizing the resource utilization on each camera node, is defined with the second equation. U is the total utilization of resource type R (different than power) by all instantiated components on camera M. This amount is kept minimal. The third equation minimizes the power consumption. The total amount of power used by all instantiated components on camera M is given by variable P and should be kept minimal. The priority order specified in Statement (III.44) assumes that the system runtime is the most important objective during the synthesis, followed by the resource utilization, and finally the power consumption. This order, however, can be changed according to the designer. Finally, all optimizations must be done subject to resource constraint, which is defined by Rule (III.46). The constraint states that the total amount U_2 of resource type R used by instantiated components on camera M should not exceed the total available resource U_1 on the FPGA of M.

III.6 Experimental Results

In this section, we presents the result of experiments conducted to demonstrate the feasibility of the proposed ASP-based encoding. Experiments are conducted in three phases. After presenting the evaluation environment, we will first demonstrate the feasibility of the proposed synthesis approach using the network depicted in Figure III.2 (Section III.6.2). Then, we will prove the effectiveness of our optimization methodology by showing –with a set of synthetic network scenarios– how it can effectively reduce the communication resources and power consumption in a distributed network (Section III.6.3). Finally, we will evaluate the robustness of the proposed search approach by synthesizing architectures for various network configurations (Section III.6.4).

III.6.1 Evaluation Platform

The computing infrastructure that has been used to design the smart cameras is our RazorCam platform [50]. The RazorCam is a smart embedded camera offering a flexible and extensible hardware/software environment to prototype and to verify video applications. It is capable of processing and analyzing image data through a Xilinx FPGA-board featuring an embedded processor (Microblaze or ARM). Additionally, the

RazorCam offers a host of real world interfaces including a Gigabit Ethernet transceiver on the main board and a Hi-Speed USB port, that could be used for wired and wireless transmission. Originally, the RazorCam is a wired/wireless camera, but it could be configured to be used either as a pure wired system –only accepting wired connections– or as a pure wireless camera. For our experiments, two different communication protocols have been considered: Serial RapidIO Gen 2 for wired communication and USB 2.0 for wireless communication. These choices however did not exclude other communication alternatives.

Table III.1 gives a brief summary on some implementations of available Intellectual Properties (IPs) and their costs, which are also described in the ASP model of Figure III.7. These IPs are available in the library in several implementations. All cost

	Throughput		**Resources**			**Power**
Task - Abbreviation	**MBlaze**	**HPU**	**Slice**	**BRAM**	**DSP**	**(mW)**
Gauss - GF	16	1	280	1	0	37
Sobel- SF	16	1	296	1	0	37
Gradient - GR	8	2	100	0	0	30
Harris Corner - HC	16	1	740	0	0	187
Bayer2RGB - BRG	16	1	119	2	0	31
Bayer2Gray - BG	16	1	1250	3	0	183
Extract Histo - EH	16	1	153	0	0	25
Mean Shift - MS	16	1	1050	0	0	150
JPEG Encoder - JE	16	5	11460	24	29	1758
Threshold - IT	16					
JPEG Head - AJH	16					
Draw Box - DB	16					
USB2.0 - USB		1	2424	0	0	98
Serial RapidIO - SR		1	6244	11	0	637
Axi_SDI - AXS			1865	7	0	190

Table III.1: Meta-information on the available intellectual properties

parameters regarding the timing (throughput of MicroBlaze and Hardware Processing

Unit) and the density (Slices LUTs, BRAMs, and DSPs) of the IPs modules were either retrieved from vendor data sheets or obtained after synthesis and implementation on the RazorCam using Xilinx synthesis tools (ISE and PlanAhead). It is worth noting that any embedded camera system, other than the RazorCam, could be used for these experiments, providing that the IP modules are synthesized for the target platform. To estimate static and dynamic power consumption, the Xilinx Power Analyzer (XPA) has been used. The power consumption provided in Table III.1 are estimated values –not measurements taken during real-time execution– and could therefore be slightly different at runtime. Nevertheless, these estimations provided a good reference point for the architecture synthesis and we intend in the next steps of the research to update our model using more realistic measurements. The workstation used for the ASP-based synthesis is a desktop computer featuring an Intel Core i5 2.67GHz processor with 3.8GB of memory.

III.6.2 Phase I - Method Feasibility

To evaluate the feasibility of the proposed synthesis methodology, we applied it on the network depicted in Figure III.2. We performed the architecture synthesis for the four cameras of the network, whose processing chains are indicated in Table III.2. The

Nodes	**Operations**	**Steps**	**Goals**
#1	Obj. Detection	$BG \rightarrow GF \rightarrow SF \rightarrow GR \rightarrow IT$	min. latency
#2	Obj. Detection	$BG \rightarrow GF \rightarrow SF \rightarrow GR \rightarrow IT$	min. area
#3	Obj. Tracking	$BRG \rightarrow EH \rightarrow MS \rightarrow DB$	min. latency
#4	Video Compress.	$BRG \rightarrow JE \rightarrow AJH$	min. latency

Table III.2: Processing chain on camera nodes for the distributed network of Figure III.2

computations implemented by each camera represent high-level computer vision operations that have been randomly assigned. The first and second camera nodes implement a canny edge detection to extract object edges in images for recognition. The third camera implements an object tracking algorithm based on the r-bin Hue histogram matching, while the fourth camera performs a video compression after the Motion JPEG standard. In all the defined processing chains, no communication tasks have been specified. These latter will be identified and added to the each camera after the *global optimization* step.

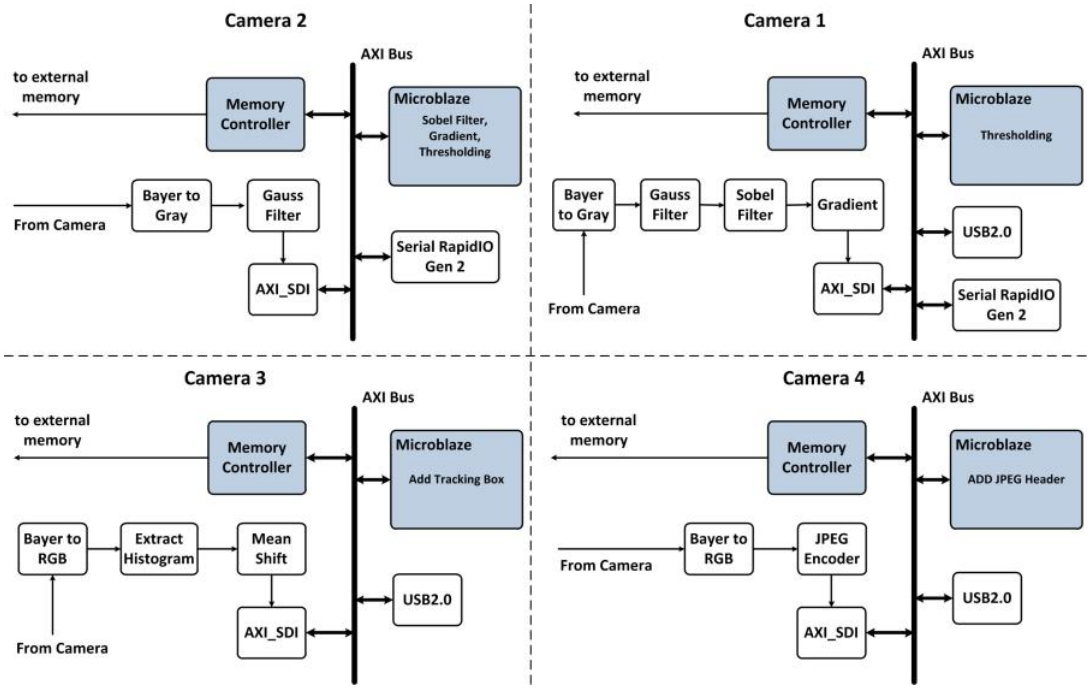


Figure III.9: Generated systems-on-chip for the 4-Node network of Figure III.2.

On the testing workstation, the ASP solver (Clingo [51]) takes approximately 6 seconds to synthesize the architectures of the whole network. Figure III.9 shows the generated systems-on-chip upon completion of the synthesis. The generated architectures indicate the mapping of low-level tasks onto the computing infrastructures following the optimization goals. We observe that, even though camera 1 and 2 implement the same operations, their architectures are different. An explanation to that is the difference in the synthesis objectives. While the goal for camera 1 was to maximize the processing performance, the objective in camera 2 was to minimize the density of the final architecture. On camera 1, computations are implemented in hardware whenever possible, but on camera 2, some tasks are scheduled on the Microblaze processor to minimize the area utilization.

Another observation is the optimization of communication resource on camera 4. Even though the node is originally a wireless/wired type, only a wireless task (USB2.0) –as communication protocol– has been added on the generated architecture. This is due to the global optimization step. Table III.3 summarizes the results of the synthesis and provides information about the utilization of resources and the estimated runtime (or latency) per camera node. The estimated runtime of a camera is understood as the time

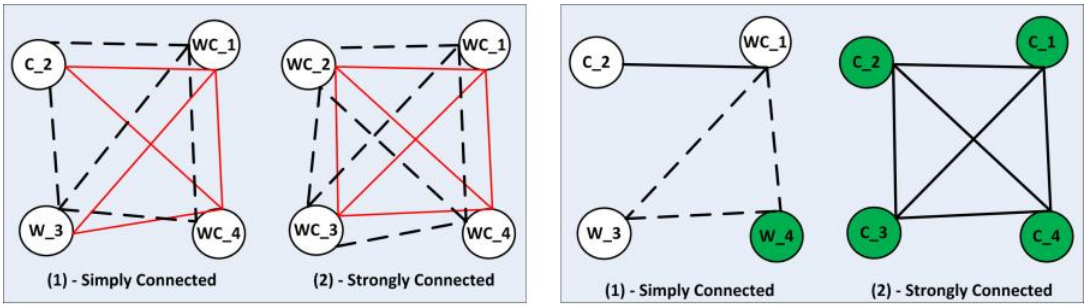
Nodes	Slice	BRAM	DSP	Power(mW)	Time slots
#1	12459	23	0	1212	21
#2	9639	22	0	1047	42
#3	5611	9	0	494	19
#4	15868	33	29	2077	22

Table III.3: Resources utilization and estimated runtime

it takes to the first pixel –produced by the camera sensor– to move through all the stages of the processing chain and be available at the output unit (the main processor or the communication component in the case of data transmission). Table III.3 shows for example that it takes 21 time units to camera 1 to output its first results –assuming there is no delay in bus transactions. We can also observe that this is faster than camera 2, which occupies however less area and consumes less power. This latency difference follows the design objectives as clearly indicated on Table III.2. The power consumption of the generated SoCs is also provided in Table III.3. Here we have only considered the processing tasks that have been mapped on a hardware logic and did not include the consumption of shared units, such as the memory controllers or the microblaze processor.

III.6.3 Phase II - Method Effectiveness

In this experimental phase, we focused on evaluating the performance of the global optimization approach to minimize communication resources in a distributed network. Global optimization has been applied on a set of synthetic networks and the generated results were compared with the performance obtained for the same network configurations without optimization. We evaluated four groups of network with different size: 4,5,6, and 8 nodes. Each group consisted of two case scenarios representing a simply connected network –as in public areas like airports– and a strongly connected network –as in highly secure places such as nuclear plants. A strongly connected network is a mesh topology in which all cameras are the same type (wireless/wired or WC). The choice for these two types of topology was motivated by the desire to emulate network systems encountered in almost every realistic environment. Because of the lack of space, a network (before optimization) was represented by its camera interconnection graph (in straight red links) and its orientation graph (in dashed black links), both combined on a single graph.

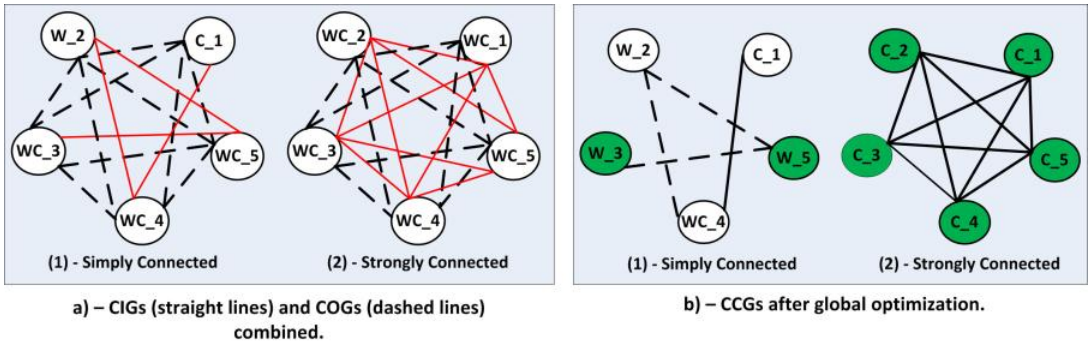


a) – Two Networks represented by their CIG (straight lines) and COG (dashed lines) combined.

b) – CCGs after global optimization. Strong lines and dashed lines represent wired and wireless connections respectively.

Figure III.10: 4-Node distributed network scenarios. (a) Original network topologies captured using the camera interconnection graph (red lines) and the orientation graphs (dashed lines). (b) The networks after global optimization, with green nodes representing cameras on which have been successfully saved.

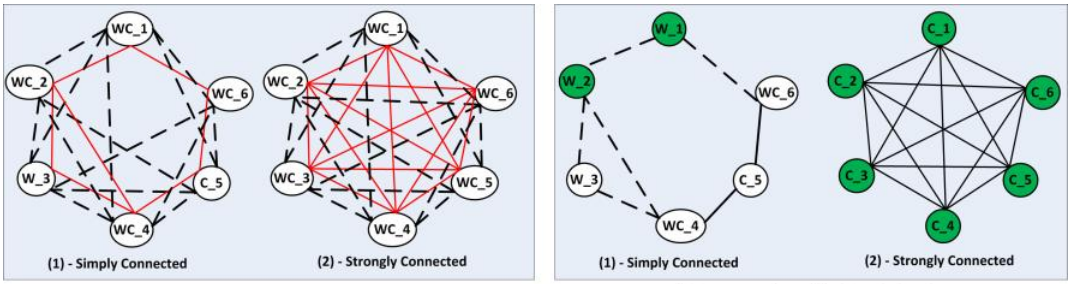
Figure III.10 shows the results of the optimization applied on the 4-Node network scenarios. Green nodes on the CCG of Figure III.10b represent cameras where communication resources have been successfully minimized. A straight edge between two nodes of the CCG represents a wired connection, while a dashed edge indicates a wireless connection. In the network configuration of Figure III.10a-(1) – which is our initial case example–, it could be observed that node WC_4 communicates only with wireless cameras. This node could therefore be synthesized as a pure wireless camera even though it is originally a wireless/wired type. This design choice will not reduce the communication performance of the camera since its wired port will not be used at runtime. Such design information are captured through the CCG, as shown on Figure III.10b-(1) concerning camera node 4. In the case of a strongly connected network, such as in Figure III.10a-(2)), all nodes can inter-communicate either by wired or by wireless. In such configurations, the priority is always given to the wired connection, as it is more reliable, can achieve higher throughput, and provides better bandwidth. As a consequence, each camera of the network will be synthesized as a pure wired node (type C) after global optimization (see Figure III.10b-(2)).
While Figure III.11, III.12, and III.13 shows the results of our optimization applied respectively on 5-Node, 6-Node, and 8-Node network scenarios, Table III.4 summarizes the output of the architecture synthesis relative to the slices utilization and the synthesis time. Also the gain in terms of resources and power saving resulting from using our global optimization approach is provided.



a) – CIGs (straight lines) and COGs (dashed lines) combined.

b) – CCGs after global optimization.

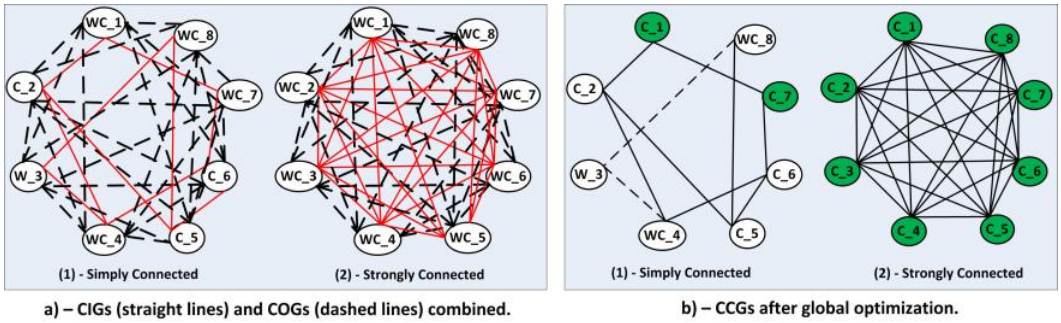
Figure III.11: 5-Node distributed network scenarios.



a) – CIGs (straight lines) and COGs (dashed lines) combined.

b) – CCGs after global optimization.

Figure III.12: 6-Node distributed network scenarios.



a) – CIGs (straight lines) and COGs (dashed lines) combined.

b) – CCGs after global optimization.

Figure III.13: 8-Node distributed network scenarios.

		Without CCG		With CCG		Gain (%)	
Netw	**Appl.**	**Time(s)**	**Slice**	**Time(s)**	**Slice**	**Slice**	**Power**
4-Node	(1)	05.72	26004	05.75	19760	24.01	28.88
	(2)	12.04	34672	12.25	24976	27.96	13.33
5-Node	(1)	08.51	34672	08.81	22184	36.01	43.33
	(2)	13.870	43340	13.940	31220	27.96	13.33
6-Node	(1)	11.320	43340	11.440	30852	28.81	34.66
	(2)	16.640	52008	16.760	37464	27.96	13.33
8-Node	(1)	11.480	55828	11.600	50980	08.68	03.96
	(2)	22.230	69344	22.380	49952	27.96	13.33

Table III.4: Results of architecture synthesis using global optimization vs. synthesis without optimization.

The slice utilization parameter shows the amount of slices used by all instantiated communication resources in a particular network. In the FPGA fabric, slices are the basic building block components containing Look Up Tables (LUTs), Registers, and other logic elements. In order to be accurate in the interpretation of the results, we decided to focus only on the cost of communication resources. The size and costs of individual IPs have been listed in Table III.1. The synthesis time is the time it takes to the ASP solver to generate instances of SoCs for all cameras in a network scenario. The gain indicates the amount of slices and power saved for communication components using our optimization approach, compared to the synthesis without global optimization (also called the blind approach).

From Table III.4, several observations can be drawn. First, with approximately the same synthesis time as the blind design approach, the proposed global optimization could help save up to 30% of the overall communication resources and up to 40% of the power consumption in a distributed network. Additionally, we realized that for strongly connected networks, the gain always remains constant independently of the size of the network and the initial costs of communication elements (27.96% and 13.33% in our case, respectively for slices and power). This is because in mesh networks, the optimization takes place on every node. Therefore the gain for such networks could be easily projected. Finally, given that wired communications are more expensive than wireless communications in terms of power and hardware resources, the more wired resources are

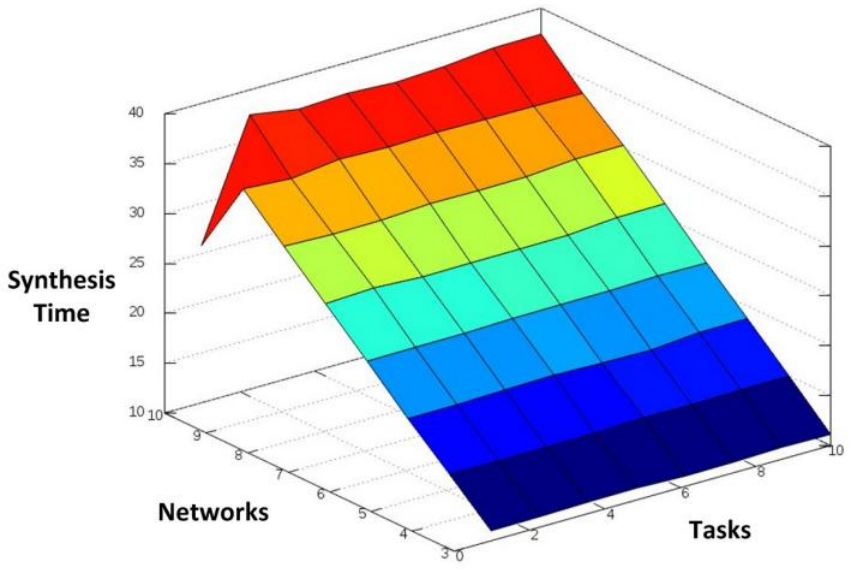


Figure III.14: Synthesis time for the ASP-based encoding.

saved, the higher the gain will be. The 5-Node network scenario of Figure III.11b gives us an illustration, where 2 wired components are saved in the simply connected case against 5 wireless components in the mesh network. But the resulting gain, shown in Table III.4, does not reflect this proportion.

III.6.4 Phase III - Synthesis Robustness

In this experimental phase, the main goal was to evaluate the robustness of the proposed ASP-based encoding when the size of the input logic problem increases. We analyzed the synthesis time of our encoding model when the size of the network and the number of local tasks per camera node vary simultaneously from 0 to 10. Due to their complexity, only meshed topologies (strongly connected networks) have been considered and the results of this experiment are presented in Figure III.14. The horizontal axis ($Networks$) of Figure III.14 represents the number of nodes per network; the depth axis ($Tasks$) represents the number of local tasks per node and the vertical axis ($Synthesis\ Time$) gives the resulting ASP synthesis time in second. The shape of Figure III.14 confirms our expectation that the synthesis time is linear when the size of an ASP model increases. As explained in Section III.5.1, an ASP model captures the description of the network, the set of computation tasks, the available computing infrastructures, and the solver rules to perform the architecture synthesis. The amount of variables, constants, and propositional

objects used to encode a model gives its size. Therefore, by increasing the number of local tasks per node and the number of nodes per network, the size of the resulting ASP problem is automatically increased. As the graph of Figure III.14 shows, it takes less than 40s (37.43s exactly) to the ASP solver to synthesize SoCs for a network of 10 nodes, with 10 tasks per node.

Additionally, we observe on Figure III.14 that the synthesis time remains constant when the number of tasks per node in a network increases. This observation could be very useful during runtime restructuring, especially when new tasks are dynamically introduced or removed in/from the network. When a new object enters a distributed network, it needs to be assigned to a camera. Therefore, new local tasks would be created on the receiving node. This latter should be able to predict the time it will take to generate its new architecture, in order to anticipate any performance degradation due to runtime reconfiguration. With a constant or linear synthesis time, the duration for runtime hardware restructuring could be easily predicted.

A side-by-side comparison with existing synthesis methodologies, such as Integer Linear Programming (ILP) or Constraints Programming (CP), would have provided more insight on the quality of the proposed ASP-based encoding. However, the presented work is new and can be seen as an extension of the MPSoC synthesis problem to simultaneous resource and runtime optimization of interconnected MPSoCs in a heterogeneous network. It has been shown in [52] that ILP, which is the main competitor to our ASP-based approach, is not even tractable for small size MPSoCs, thus making it less likely to tackle the complex problem presented in this chapter.

III.7 Conclusion

In this chapter, we provided a compact encoding model based on ASP for a multi-objective systems-on-chip synthesis for smart cameras in a distributed network. Using the proposed ASP-based method, we overcame the issue of size explosion and exponential synthesis time encountered with other synthesis approaches such as Integer Linear programming or Constraint programming. The new compact encoding generates in linear time optimal system-on-chip architectures representing the mapping of low-level tasks onto available camera infrastructures. The feasibility of our synthesis approach was demonstrated on several network scenarios. Experimental results showed that it is

possible to synthesize architectures for a network of 10 cameras –with 10 tasks per node– in less than 40s. We have also demonstrated that by leveraging network information, such as the network topology and the camera overlapping FoVs, we were able to reduce the costs of communication resource in a distributed network; thus reducing the power consumption.

Given that hardware flexibility and dynamic reconfiguration of the computing architecture are the main reasons of using of FPGAs as target platforms, the next chapter will present the generated SoC in detail and show how the HW/SW decomposition properties of FPGA-based systems are leveraged to provide the required flexibility to an embedded smart camera.

IV A Reconfigurable System-on-Chip for Embedded Computer Vision

Designing smart cameras for distributed embedded processing in a dynamic network is very challenging. In fact, embedded computer vision places enormous computational demands on smart cameras. It leverages time-consuming and computationally intensive models constructed with the aid of geometry or physics in order to produce symbolic information from raw image data. However, besides having to meet very tight execution deadlines, smart cameras operate in general with restricted capabilities and must therefore be designed to use few hardware resources and to consume very low power and energy. Additionally, the dynamic nature of the network, considering camera failure probability, requires the computational structure on the camera nodes to be readjustable at runtime to keep with system performance.
In chapter III, we presented a synthesis methodology for generating generic systems-on-chip (SoC) for computer vision operations on platforms with restricted resources. This chapter will present the generated SoC in detail and explain how the internal structure of the camera system is composed to maximize processing performance and support runtime orchestration and auto-collaboration with other smart cameras in a network.

IV.1 System Overview

Figure IV.1 presents a block diagram of the proposed system-on-chip (SoC) architecture implemented inside a FPGA, which is the main computing element within the smart cameras. From a practical point of view, the proposed architecture aims at optimizing the performance of video processing and communication in embedded camera. The resulting hardware/software partitioning implements low-level computationally demanding tasks such as segmentation and corner detection as dedicated hardware component while control-dominated tasks such as self-coordination and handover decision are executed in software. As explained in chapter III, the mapping decision between hardware and software is guided by design objectives and as consequence, this architecture could be slightly different from one camera to another.
From the diagram of Figure IV.1, raw image data received from a digital camera sensor

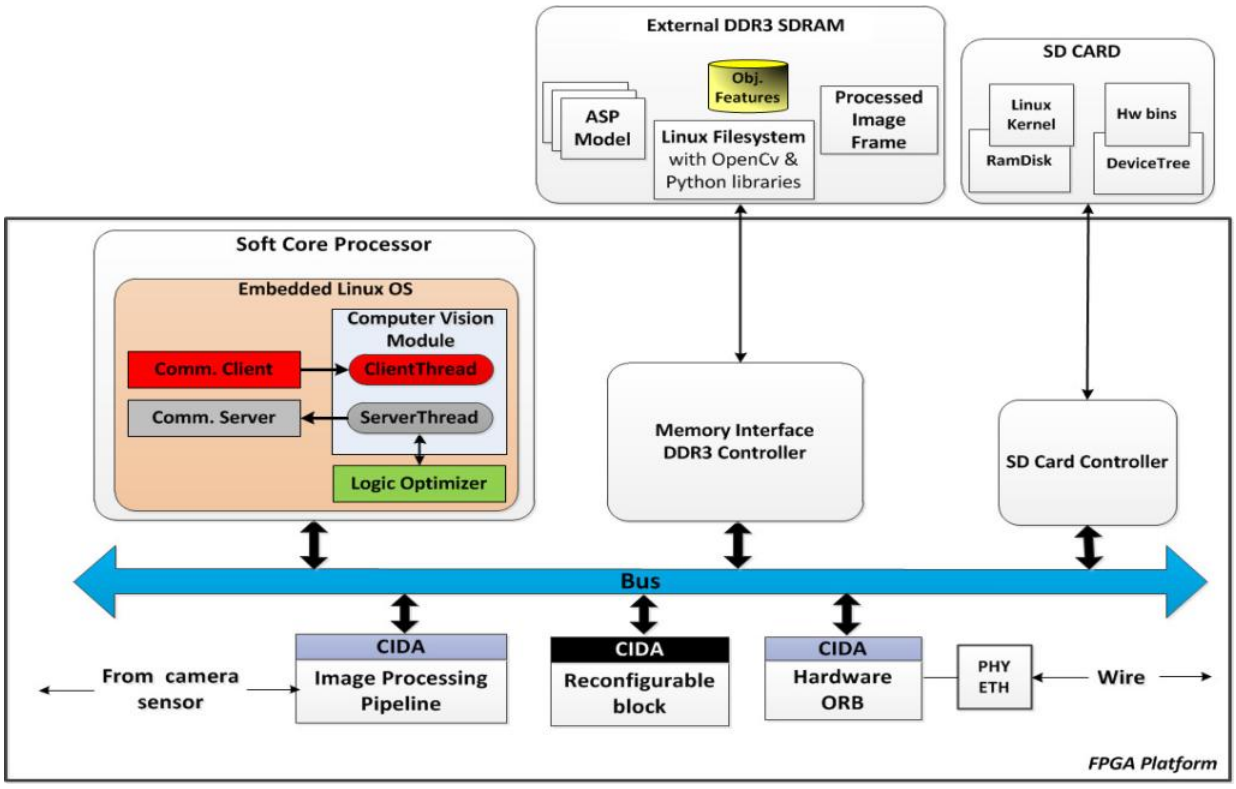


Figure IV.1: The FPGA-based system-on-chip with the embedded on-line optimization engine.

are first processed in hardware through a pipelined chain of IP cores and stored into an off-chip DDR3 Memory for analysis. Elements of this chain consist of low-level image processing functions, such as color conversion (Bayer to RGB) or convolution (sobel edge detection), that are good candidate for hardware acceleration because of their computational structure. Next, the processed frames are analyzed by the computer vision module (CVM) inside the soft core processor to extract new knowledge that will be passed to the embedded logic optimization engine for evaluation. Extracted information represent either object features (appearance, motion, etc.) or events (object at entrance/exit zone). Upon evaluation of new information, outputs of the optimization module (e.g. target transfer, new object assignment) are transformed into commands and executed locally on the camera. In case of inter-camera data exchange, a hardware object request broker (ORB) component [53] is used as middleware for seamless and real-time intercommunication among camera nodes in the network. The processor runs an embedded Linux operating system that is booted from SD card at system start-up. Besides holding processed image frames, the external memory also contains the Linux

root file-system, the ASP rules for modeling self-coordination, and a list of detected target features.

IV.2 The Hardware Platform

Selecting the appropriate processing platform for a smart camera is an important issue. In chapter III, we provided the motivations for the use of FPGAs as the target platform of our embedded smart cameras. These can be summarized as high-performance, high throughput, low cost, and low power. In terms of performance, FPGAs are becoming increasingly attractive for image processing tasks because they can perfectly exploit the inherently parallel nature of many vision problems [54]. Moreover, through the partial reconfiguration property, FPGA-based systems offer the possibility of restructuring (part of) the hardware at runtime. Among available FPGA families, we use Spartan6 and Zynq-70x FPGA platforms from Xilinx for our experiments due to their low price and high volume of logic gates, but other technologies could be used as well without significant modification.

IV.3 Image Processing Pipeline

This unit is composed of several low-level image processing elements which can execute in parallel with each other and perform specific task on incoming video data. The generic flow usually consists of a Bayer-to-RGB color conversion followed by an image segmentation where background and foreground of frames are separated. Before being saved into the external memory and used for software analysis, several filters (sobel, threshold) are applied on the foreground image to enhance specific regions such as edges or corners. A reconfigurable logic area is also provided for runtime modification of the image processing operations. When reserved, this region is simply used as an additional step to the pipelined chain.

IV.4 CIDA Interface and Hardware ORB Middleware

The Component Interconnect and Data Access (CIDA [55]) is a portable interface module designed for data exchange among software and hardware components in a SoC. It is made of a streaming interface and a memory-mapped interface, which can be used for

data transfer between memory/peripheral and hardware module without processor intervention. DMA capabilities in the CIDA allows for fast data transfer between memory, processor, peripheral, and hardware accelerators.
The middleware technology has been used in the past to address the issues of collaboration and remote data access. Usually, a client node requests an object via an interface and in response, a server replies and services the requested object. In this work, we rely on the well known Common Object Request Broker (CORBA) middleware and use the hardware ORB component proposed in [53]. As a CORBA-based middleware IP core, the hardware ORB allows clients in a real-time manner to invoke operations on distributed objects without concern about communication protocols. In fact, for high-bandwidth and high data-rate communication, a significant amount of protocol processing is implemented in hardware to exchange contents of node memory through the GIOP protocol (TCP/IP), directly in native gate level without usage of embedded processor. A predefined block of memory is used as a CORBA Servant Object such as processed images or network configuration data. A request from a remote node acting as CORBA client will trigger the hardware ORB to fetch this requested data from memory and transfer it through the network. We modified the hardware ORB and added the CIDA interface to handle all transactions local to a camera node, while the CORBA mechanism takes care of object sharing in the network.

IV.5 Software Architecture

This section gives an overview of the overall software architecture. The software structure inside the processor is adapted from [53]. In addition to the already existing processes (the computer vision module (CVM), the communication client (CC), and the communication server (CS)), the embedded logic optimizer module (ELO) is added for runtime evaluation of extracted knowledge.

IV.5.1 Computer Vision Module

The computer vision module (CVM) is the component in charge of launching the upper intelligence layer, analyzing processed frames stored in the external memory, and executing the commands derived after logic optimization. It leverages the OpenCV library from Intel, cross-compiled for the embedded processor, that encapsulates several

computer vision algorithms for fast and complex image processing such as the optical flow for motion tracking or the cascade classification for object identification and classification. The CVM consists of two threads as shown in Figure IV.1: the ServerPeer_Thread (SPT) and ClientPeer_Thread (CPT).
Periodically, the CPT receives network information regarding the tracking activities in the direct neighborhood (such as the list of neighbors and their respective targets) from the communication client process and updates locally the list of active neighbors to match the dynamism in the network.
The SPT is responsible of updating the list of targets locally assigned to the camera, analyzing processed frames inside main memory, and extracting interesting features and events. These features are then encoded as ASP data (facts, constraints, etc.) and evaluated by the logic optimizer. If any action –such as a target transfer– is to be taken as the result of optimization, the SPT will notify the communication server process by sending a user signal.

IV.5.2 Communication Server and Client Processes

The communication server (CS) and communication client (CC) processes are in charge of sharing information between a camera and its direct neighborhood. They have an interconnection with the CVM through shared memory. The CC module is used to request data to a remote camera, while the CS is used to transmit information to a particular node. Upon the request of a client process in another smart camera node, the hardware ORB fetches the requested data (CORBA Servant Object) from memory and directly serializes and transmits it through the protocol stack towards the client node. Periodically, each camera node will distribute its tracking status to all nodes in its direct neighborhood. This task is insured by the CS process.

IV.5.3 Embedded Logic Optimization Engine

The embedded logic optimization module (ELO) is the heart of the entire camera system as it is the one that decides appropriate actions to be taken based on information extracted from processed frames. Figure IV.2 depicts the 4-step cycle describing the intercommunication between the SPT and the ELO from analyzing an image frame to performing the appropriate action based on activities in the frame.

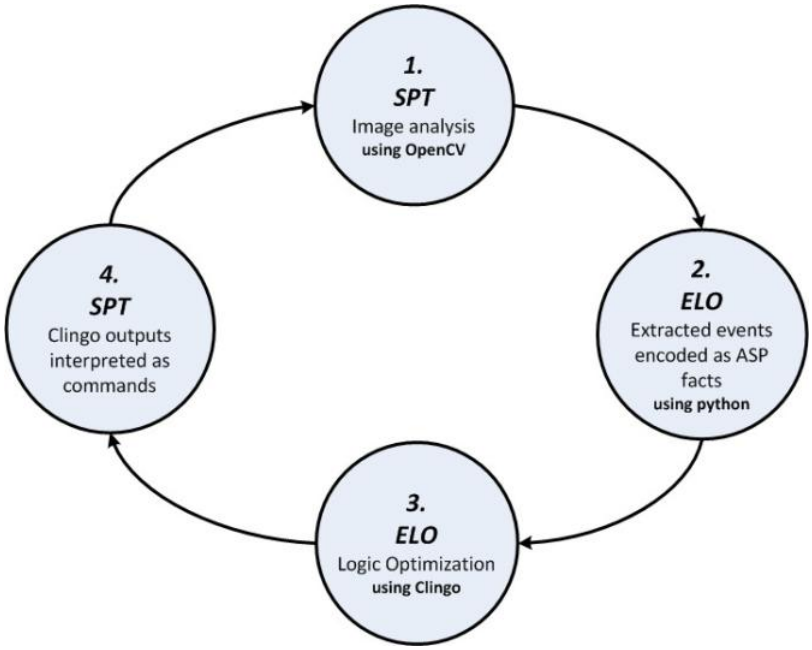


Figure IV.2: Processing cycle from analyzing an image frame to performing the appropriate action based on activities in the frame.

After frame analysis, events extracted –such as a target at exit zone– or received from the neighborhood –such as a node failure– are encoded as ASP facts/constraints and passed to the logic solver as a self-coordination problem instance. For this research, the set of *interesting* events is limited to those that trigger either a target handover (object at entrance/exit zone of a camera FoV) or a target redistribution (node failure). Next, the encoded ASP facts are applied on the rules modeling the self-coordination (see Chapter V) for determining appropriate behavior of the camera, such as updating the list of assigned objects in case of node failure. Results from logic optimization are then sent back to the SPT, interpreted and transformed into commands that will be executed on the camera. For runtime evaluation of ASP-based self-coordination problem instances, the ELO leverages Clingo from the POTASSCO toolset [56], an award winner and incremental answer set solver for extended normal and disjunctive logic programs. Clingo combines the high-level modeling capacities of ASP with state-of-the-art techniques from the area of Boolean constraint solving and mostly relies on conflict-driven nogood learning, a technique that proved very successful for satisfiability checking (SAT). Clingo has the capacity of either evaluating a rule only once (at the beginning of an incremental computation) or accumulating the results of evaluations over a whole incremental

processing. These properties help to adequately emulate stochastic operations that occur randomly over the time, as it is the case in distributed tracking.

IV.5.4 Embedded Linux Operating System

The embedded Linux operating system is exploited in the proposed architecture to run on the embedded processor. It aims at increasing system programmability by allowing developers to easily migrate their existing applications into the proposed camera platform. For instance, Intel's computer vision library OpenCV [57] and the Python library are ported to provide the ability of high level programming in the image processing domain.

IV.6 Conclusion

This chapter presented a reconfigurable architecture for embedded vision processing. Besides the performance and flexibility abilities of FPGAs, programmability remains one of the key factors to make the platform usable. Hence, the goal of this chapter was the internal software programmability aspect of the proposed architecture and the ability to make the platform usable by a large research community. To tackle the performance issue of embedded computer vision, we proposed a hardware/software decomposition approach that allows computational intensive blocks of the system to be accelerated in hardware, while the control parts remain in software. Integrating the embedded Linux, the open source framework OpenCV, and the Python Library to the proposed platform was ideal to increase the acceptance of the platform in the image processing research community. Chapter V will evaluate the performance of the proposed SoC by investigating the implementation of a distributed object tracking system.

V Runtime Self-Coordination of Embedded Smart Cameras for Distributed Object Tracking

V.1 Introduction

Smart camera networks have emerged as an alternative to traditional CCTV for efficiently implementing multiple object tracking systems in large areas [9]. In such networks, the distributed nodes collaborate to approach a comprehensive coverage of the area under monitoring; which not only improves the responsiveness of the system but also increases its flexibility. The prerequisite for a better coverage in a smart camera network is a viable approach for tracking across cameras. The goal in this case is not necessary to follow as many targets as possible, but the most relevant ones. "Keeping and eye" on important targets, such as a man abandoning a bag in an airport, could be very challenging in scene with increasing number of moving objects. To address this problem efficiently, we need: 1) a rigorous formalism that captures all facets of the tracking problem, 2) a computational tractable and accurate strategy to devise a solution from the previous formulation and finally, 3) a flexible camera infrastructure that could be modified on the fly to adapt to runtime changes in the monitoring scenes.
In Chapter III and IV, we synthesized and implemented a reconfigurable system-on-chip that meets the computational requirements stated in 3). In this chapter, the focus is on investigating a compact and robust formalism that will leverage the designed architecture to enable runtime orchestration of smart embedded cameras in a distributed network setup. The auto-coordination of cameras will aim at tracking moving objects in a distributed network environment so as to optimize the camera-to-target assignment and the camera handover.
In the literature, proposed solutions for object tracking in a smart camera network range from the implementation of complex computer vision operations for PC-based infrastructures [58], the design of dedicated hardware systems [16], the use of Pan-Tilt-Zoom(PTZ) cameras [59] and software agents for self-coordination in smart network systems [18, 19, 60, 61, 15]. However, because of the complexity of the implemented operations, the random nature of the network environment, and the severe limitations of the computing infrastructures –in terms of processing power, energy

lifetime, and memories– many of the proposed solutions are not efficient for real-time implementation on distributed embedded systems. Consequently, when the size of the network and the amount of moving targets increase, the system performance (processing speed, target distribution time, amount of target lost, camera handover time, etc.) decreases exponentially.

In this work, we present a new approach for enabling self-coordination of smart cameras during object-tracking in a distributed network. Our methodology differs from existing work in that we introduce for the first time the use of an analytically exact approach, derived from boolean satisfiability (SAT), for camera-target assignment and camera handover decision. We encode the camera-target assignment, the camera handover process, and the target redistribution after node failure as a logic problem that is dynamically solved through a systematic search over a space of potential solutions. We leverage Answer Set Programming (see Chapter II) to formally capture the target assignment problem in a distributed network as a mapping problem of a set of tasks (targets) onto a set of resources (cameras). ASP is guaranteed to always produce a solution for a given optimization problem; that is, to always terminate. While system performance in current camera assignment methodologies suffers from an increasing number of cameras and objects at runtime, ASP-based methods have proven to achieve satisfactory performances for the synthesis of problems with hundreds of nodes [24] and represent therefore a viable alternative for addressing the complexity of auto-organizing efficient target handover among camera nodes of a dynamic network. Experiments on a set of network scenarios demonstrates the feasibility and robustness of the proposed method.

The rest of the chapter is organized as follows: Section V.2 introduces in detail the addressed problem in this work. Related research with regard to distributed object tracking in camera network are discussed in Section V.3. In Section V.4, we present our approach for object tracking and camera handover using Answer Set programming where emphasis is put on minimizing the target-to-camera distance and the runtime load distribution in the network. Experimental results are provided in Section V.5, followed by concluding remarks in Section V.6.

V.2 Problem Definition

This chapter addresses the problem of assigning moving targets to static cameras in a distributed network with the goal of maximizing the network coverage, while minimizing the overall tracking cost and handover time. Our formulation is borrowed from high-level synthesis where a *task graph* is used to represent all objects/tasks, and an *architecture graph* to capture all camera nodes in the network along with their capabilities. Because of changes in the set of targets to monitor due to motion, the functions, set and variables used to capture our problem must explicitly be time dependent.

Formally, we capture an application using an *object graph* $g_o^t = (V_o^t, E_o^t)$ where V_o^t represents all the objects present in the environment at time t and E_o^t represents the dependencies among them. An edge $e = (i, j) \in E_o^t$ represents the distance between two objects $i, j \in V_o^t$ at time t. The camera topology is captured using a *camera graph* ${g_c}^t = ({V_c}^t, {E_c}^t)$, where ${V_c}^t$ is the set of camera nodes in the network and E_c^t is the set of communication links among the nodes. We consider a node with all its resources as a whole, which means that the granularity of the communication will not be broken down to resource communication across the nodes. *The goal in high-level synthesis is to find an assignment (mapping) of the nodes in the task graph to those in the architecture graph that best match with the lowest cost factor.* Formally a mapping $\beta_t : V_o^t \rightarrow {V_c}^t$ is a function that makes a many-to-one association between elements of the *domain* (set V_o^t) and element in the *range* (set ${V_c}^t$) so as to optimize predefined objectives. In our case, examples of such objectives will be to minimize the camera-target distance at system start-up and maximize the overall load balancing at runtime, that is, the object distribution across cameras at any particular time. This is similar to pruning a large data set of candidate solutions and searching for a particular (pareto) optimal. Finding such a mapping function is a non-deterministic combinatorial optimization problem that could result in an exponential searching time, especially in case of large networks; which is very critical for real-time embedded applications where time constraints are of utmost importance.

V.3 Related Work

In the literature, numerous works have focused on implementing practical solutions for object tracking in a multi-camera network. In [62], a distributed system has been

presented for tracking people moving in an indoor environment. The system leverages a blob segmentation approach and the Kalman Filter to respectively extract object features from images and track those features as they move over the different camera field of views (FoV). Isler et al. [21] proposed a tracking solution to address the Focus of Attention problem (FOA) in a distributed network. The solution consisted of assigning $2n$ camera sensors for tracking n objects, with the goal of minimizing the expected error associated with a position estimate obtained by fusing the information coming from any pair (i,j) of sensors following a target k. Hartmann et al. [63] presented a camera system implementing a Gaussian Mixture Model for estimating and tracking human positions in an indoor environment. In [64], a tracking system for distributed camera networks was presented that leveraged the Monte-Carlo approach. VeliPasalar et al. [65] designed a distributed camera system for multi-object tracking using a peer-to-peer PC-based infrastructure. This implementation addressed the fault tolerance issue by using inter-camera communication to allow each camera to detect and keep track of its targets as they move across other camera FoVs, instead of transferring over the control. All these approaches, however, were neither scalable because of the pre-constraints on the amount of cameras and targets in the network, nor suitable for embedded applications given the resource requirements of implemented operations. Indeed, many of these aforementioned solutions leverage complex, power- and resource-hungry computer vision processing. Another common approach of implementing decentralized object tracking in multi-camera networks that has received a lot of attention over the past decades is the use of agents [18, 19, 66, 60, 15]. Agents are code segments that can migrate in the network and trigger a computation on a given node using local resources. They are mostly used for the fusion of data collected from different cameras and for coordination of computation. On a distributed platform, cameras, stationary and moving objects can indeed be considered as autonomous agents that co-operate to infer the most plausible scene understanding. Starzyk et al. [18] proposed a negotiation-based handoff approach for camera coordination in a distributed network. In their solution, a camera leverages the knowledge of its surrounding cameras –in term of tracking activity– to negotiate and generate conditional offers during handoff sessions. A similar auction-based handover approach is proposed in [19] wherein self-interested autonomous agents exchange responsibility for tracking targets in a market mechanism. While feasible in software, agents are not the optimal approach to implement real-time distributed tracking systems on limited computing

infrastructures since they generally need additional resources (memory) and their code is usually interpreted, and therefore slower than pre-compiled and optimized software or hardware.
There also exist several work that have focused on implementing image processing solutions for tracking on embedded devices while considering the limited nature of the computing infrastructures. Rowe et al. [67] designed a low-cost embedded vision system that, due to the very limited memory and computing resources, only implemented low-level image processing functions such as threshold and filtering. In [16], Quaritsch proposed a smart embedded camera platform for autonomous tracking and object handover in a distributed network. Their architecture features several DSPs as computing element and an ARM-based network processor running a standard Linux operating system for the network communication.

V.4 The Tracking Approach

The main objective of the distributed tracking system presented in this chapter is to always insure the best (if not a complete) coverage of a monitoring area over time as targets move in and out of the FoVs of visual sensors nodes. The tracking is subject to constraints that each camera is static and has a limited *tracking capacity* (C_{max}), which is the maximum amount of targets that it can simultaneously track. The proposed approach consists in three steps, sketched in Figure V.1: 1) camera-target assignment at system initialization, 2) runtime features update, and 3) self-coordinated target handover.
The input to the tracking system is a description of the network environment at start-up: cameras and objects, camera-target distance, and network coverage. Shared by all cameras, this description represents the network state when the system kicks off. First, objects are assigned to cameras in a purely distributed manner so as to minimize the camera-target distance. Results coherency is guaranteed by the fact that all cameras share the same vision of the network at initialization. We leverage Answer Set Programming (ASP) to encode the assignment problem as a search problem and to compute the optimal configuration by pruning the set of all possible assignment scenarios. Once objects are assigned to a given camera, their features are locally extracted –using well-known computer vision operations– and will be used during camera handoff. Upon leaving the camera FoV, features of moving targets are transmitted to the node where the

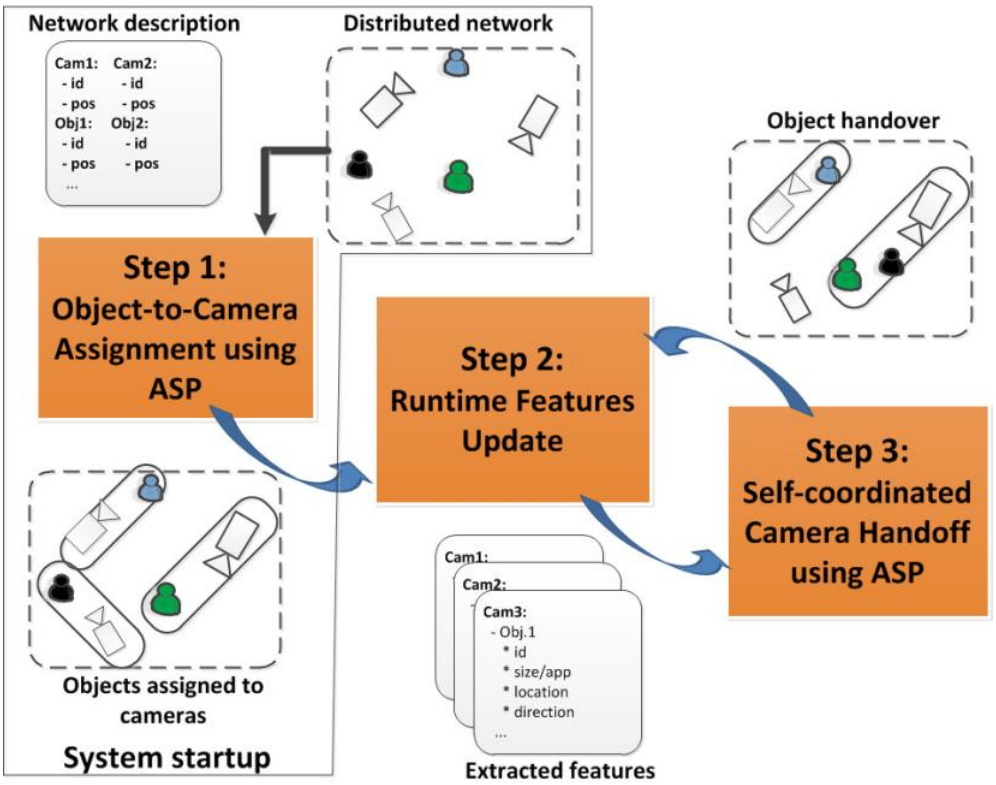


Figure V.1: Processing flow of the distributed tracking system.

objects are moving. This handover step is encoded as an ASP-based search process whose goal is to find the handoff configuration that realizes both the minimum camera-target distance and distribution.

The ASP model proposed in this work consists of three parts: the network specification (V.4.1), the description of a problem instance or tracking scenario (V.4.2), and the solver model (V.4.3). This separation offers a high flexibility and reusability of the proposed model as it allows, for example, different tracking scenarios to be tested on the same solver or a given problem instance to be modified without having to update the network description. Although terms *object* and *target* could be used interchangeably, the proposed model description will refer to *target* as an object already assigned to a given camera.

V.4.1 The Network Specification

The network specification describes: i) the distributed environment with all cameras and objects available at system initialization, ii) the camera pose, and iii) the network

coverage.

A camera in the network is captured with an atom $cam(C)$, where C refers to a unique *camera identification number* (CIN). An object is modeled as $obj^t(O, A_{app}, M_{dir})$, where $A_{app} \in \mathbb{N}^+$ is the object appearance inside the tracking camera, t the time-stamp, and $M_{dir} \in \{none, north, south, east, west\}$ the direction of motion relative to the tracking camera. $O \in \mathbb{N}$ is a randomly assigned *object identification number* (OIN) that uniquely identifies the object in the system. Values M_{dir} and A_{app} default to *none* and 0 respectively at $t = 0$, describing a motionless and unknown-appearance object. Note that atom *obj()* does not explicitly (need to) contain any reference to the tracking camera since the model is distributively executed on all the nodes.

The camera pose –that is the position and orientation of cameras in the network– is captured using atoms $cPos^t(C_i, C_j, L_{locate}, D_{dist})$ and $cFov(C_i, C_j)$, with $L_{locate} \in \{north, south, east, west\}$ being the position of camera j relative to i and D_{dist} the inter-camera distance at time t. Even though camera position is static, these atoms are time-constrained given that the network topology could change at runtime due to node failure. Distance to a failed node will be marked as equal to 0. For example, atoms $cPos^0(1, 2, south, 10)$ and $cFov(1, 3)$ refer to the fact that at initialization, camera 2 is located at 10 units south of camera 1, which shares an overlapping field-of-view with node 3.

Finally, for each object O in the network that can be viewed by a camera C at start-up, atom $cov^t(C, O, D_{dist})$ is defined to capture the coverage of C at time t. D_{dist} is the camera-target distance, which is estimated at system start-up and no longer required at runtime. For instance, atom $cov^0(1, 2, 3)$ will hold iff camera 1 is covering object 2 at initialization –with camera-target distance equal to 3 units.

The network specification represents the global view of the distributed environment, shared by all camera nodes. Decentralizing the processing requires all nodes to start with the same network configuration in order to avoid inconsistencies in the initial camera-target assignment.

V.4.2 Specification of a Problem Instance

The second part of our tracking model consists in the description of a handover problem instance on every camera of the network. Problem specification on a given node is

achieved by providing up-to-date information regarding the targets to follow. Knowing objects motion and direction within a camera region allows to determine the appropriate time (*when*) and place (*where*) to transmit their features in case of handoff. The work described in this chapter focuses mainly on runtime object handover and target redistribution after node failure, and makes a certain number of assumptions about nodes in the network:

- a camera can leverage existing computer vision operations to track targets assigned to it within its FoV;
- the object features computed by a camera are robust enough to uniquely identify a moving target;
- a camera is able to detect when an object is entering/leaving its FoV and determine its direction.

These are reasonable assumptions motivated by recent advances in human behavior analysis [68, 69] and pedestrian detection, recognition and tracking in video surveillance [70]. Moreover, previous researches have made similar assumptions [18, 19, 17].

Intra-camera tracking is insured by having each camera node constantly updating the list and parameters of assigned objects. Although more features could be used at runtime to increase the tracking robustness, only appearance and direction of objects will be considered in this research. An object O leaving the FoV of a camera C at time t is modeled with the atom $lFov^t(C, O)$, while $eFov^t(C, A_{app}, S_{side})$ captures an unknown object with appearance A_{app} first entering the FoV of C from side S_{side} at t. Here, the reference to the tracking camera is explicitly indicated insides both atoms, but it could be omitted as well without significant impact on the modeling. Node failure is modeled with atom $fail^t(C)$ where C indicates the CIN of the failed node. To guarantee a synchronous target reassignment process, this work assumes that the failure is simultaneously detected by all cameras in the direct neighborhood of C.

Figure V.2 gives an example of a tracking problem instance. Lines 1, 4 & 8 of the Figure represent commented lines. At Line 2, the tracking capacity of camera 1 is set to 10. Line 5 and 6 capture the fact that, at time $t = 1$, object 4 with appearance 5 is moving to the south (within camera 1 region), while an unknown object with appearance 7 is entering

```
1.  %% Tracking Capacity %%
2.  mObj(1, 10).
3.
4.  %% Time step 1 %%
5.  obj(4,5,south,1).
6.  eFov(1,7,north,1).
7.
8.  %% Time step 2 %%
9.  lFov(1,4,1).
10.
11. %% Time step 5 %%
12. fail(3,5).
```

Figure V.2: Example of an ASP-based tracking problem instance captured on camera node 1 at time $t = 1$, $t = 2$, and $t = 5$.

the camera FoV from the north (object at entrance zone). At time 2, object 4 is detected at the exit zone of camera 1 (Line 9), with the exit direction implicitly devised from the object direction. This event will trigger a handover process with the goal of transferring object properties to the appropriate camera in the moving direction. At time 5, node 3 is detected as broken (Line 12) and at this point camera 1 will start a target recovering process. In general, an event detected/recorded at time t will be processed/evaluated at the next time-stamp $t + 1$.

V.4.3 The Solver Model

The solver model defines all necessary rules to find on a camera the solution to a given tracking problem instance considering a particular network scenario. This model is shared by all nodes in the network in order to guarantee a uniform self-coordination. Encoding of the proposed model first deals with camera-to-target assignment at system start-up wherein all identified objects in the network at initialization are decentrally-distributed to cameras. Next, rules for self-coordinated object handover are defined for allowing moving targets to be effectively exchanged among camera nodes without any human intervention. Finally, the target redistribution process after node failure is presented where objective is to minimize the amount of lost targets in the network. Each of these steps is implemented according to the *generate-define-and-test* pattern (as defined in Chapter II) where all possible camera-target assignments or handoff scenarios are first generated, followed by a systematic elimination of *non-satisfactory* configurations regarding predefined user

constraints, until the optimal configuration is found.

a) Camera-to-Target Assignment at Initialization

The assignment process at initialization (t = 0) maps each object to exactly one camera node of the network. Figure V.3 illustrates this mapping operation. Basically, from a

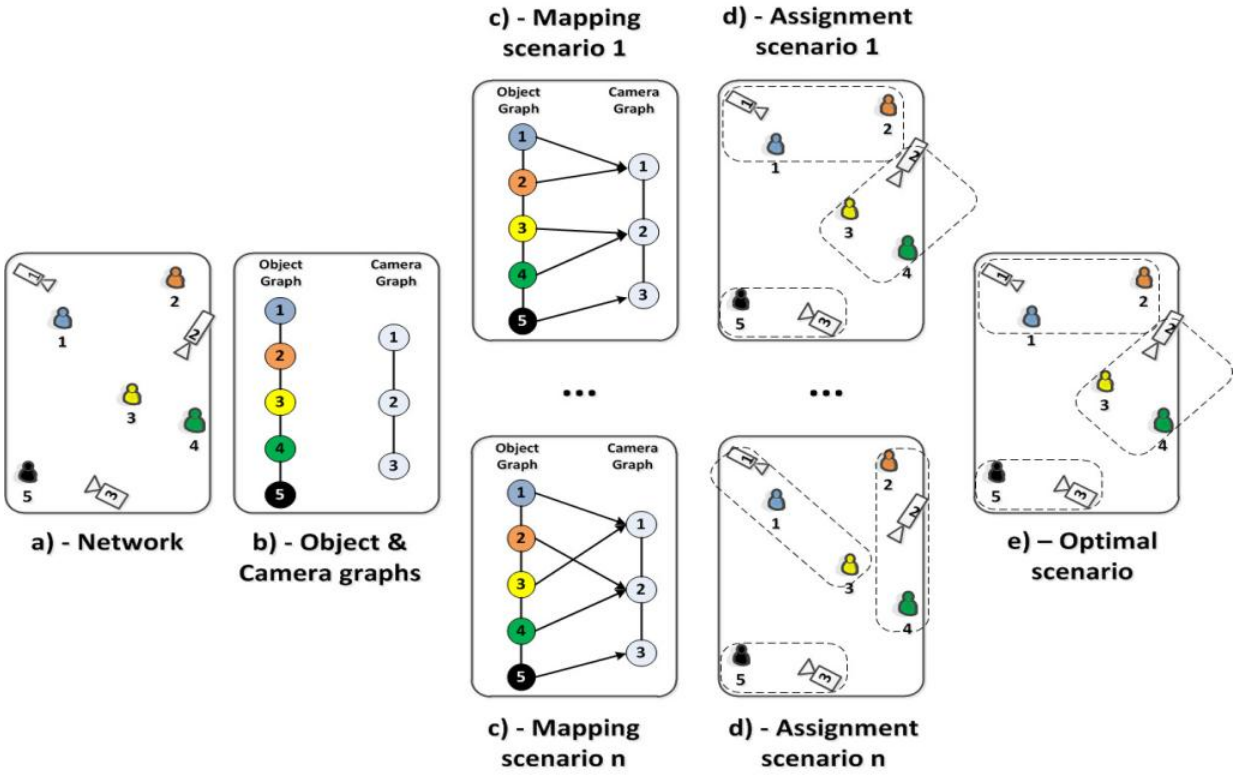


Figure V.3: The camera-target mapping at system start-up. (a) The distributed network. (b) The corresponding object and camera graphs. (c) n Mapping diagrams representing possible many-to-one association scenarios. In the first scenario (c-1), camera 1 is assigned objects 1 and 2, while objects 3 and 4 are assigned to camera 2, and object 5 to camera 3. (d) The corresponding network organization with objects assigned to cameras. (e). The optimal solution is selected regarding user objectives.

network of distributed cameras (step a), two graphs are generated (step b): the camera interconnection graph (as defined in Chapter III) and the object graph, capturing the set of objects in the network at start-up. A systematic association (step c) between these graphs is then performed producing several possible mapping configurations (step d) that would be successively refined until optimal solution (e).

For each camera C and each object O in the initial network configuration, an atom $ass^0(C, O)$ is defined representing the mapping of O with C at time $t = 0$. This

association is subject to the coverage constraint, that is, a camera can only be assigned an object within its FoV:

$$\{ass^0(C,O) : cam(C), cov^0(C,O,D)\}\ 1 \leftarrow obj^0(O,0,none). \tag{V.1}$$

Rule V.1 derives (at most) a unique assignment from each camera-object coverage defined in the initial network specification. This is the *generate* step where all mapping scenarios between cameras and objects in the network are produced, with the only condition that a camera is associated to an object if this latter is within the coverage area. A target that is not covered at start-up will not be assigned to any node at this step and will be handled as new object when entering a camera coverage area.
To insure that a target is mapped to exactly one camera node of the network at a time, the number of assignments for any object must always be equal to 1. In other words, two different cameras (C_i and C_j) must not be assigned the same object (O) at the same time t:

$$\leftarrow cam(C_i), cam(C_j), C_i \neq C_j, ass^t(C_i,O), ass^t(C_j,O). \tag{V.2}$$

To guarantee a balanced distribution at start-up, it should not be possible such a scenario where a camera (C_i) remains idle while a neighbor node (C_j) is tracking more than one object (Constraint V.3). This condition is only valid at the initialization phase and will not be anymore a requirement at runtime.

$$\begin{aligned} \leftarrow cam(C_i),\ N_i\ =\ \#count\{O_i : ass^0(C_i,O_i)\}, \\ N_j\ =\ \#count\{O_j : ass^0(C_j,O_j),\ C_j \neq C_i\},\ N_i = 0,\ N_j > 1. \end{aligned} \tag{V.3}$$

Constraint V.3 leverages aggregate *count* to prohibit that the amount N_i of objects assigned to a given camera C_i is equal to 0 while another node C_j is assigned more than one object.
Next, it must always be insured that the amount of targets (N) assigned to a node (C) at any time t – also refer as the *runtime tracking activity* (RTA)– does not exceed its tracking capacity, since nodes are embedded platforms with limited resources. This is encoded in the following rules:

$$tracN^t(C,N) \leftarrow cam(C), N\ =\ \#count\{O : ass^t(C,O)\}. \tag{V.4}$$

$$\leftarrow mObj(C,C_{max}), tracN^t(C,N), N > C_{max}. \tag{V.5}$$

where RTA N and tracking capacity C_{max} of a camera node C at time t are respectively captured with atoms $tracN^t(C, N)$ (Rule V.4) and $mObj(C, C_{max})$ (Constraint V.5). Finally, the optimal solution at this level would be the configuration that minimizes the overall camera-to-target distance:

$$\#\text{minimize}\{\ D,\ cov : cov^0(C, O, D), ass^0(C, O)\}. \tag{V.6}$$

In rule V.6, the total camera-target distance, for each possible assignment configuration, is computed and the scenario realizing the lowest value is chosen as the initial mapping configuration.

b) Self-coordinated Object Handover

To insure an efficient tracking over a distributed network after camera-target initialization, any object exiting or entering a camera region must be successfully handed over. In this work, a consistent labeling is implemented by preserving the identification of targets while they move across the different camera regions.

Target leaving the FoV of a Camera Upon exiting the monitoring area of a camera C_s, a moving target should be handed over to the closest node into whose field of view the object is moving. Consequently, an appropriate receiver candidate C_r should be any active neighbor with an overlapping view:

$$\begin{aligned} cPfov^t(C_s, C_r, P, D) \leftarrow\ & cam(C_s), cam(C_r), cPos^t(C_s, C_r, P, D), D \neq 0. \\ & cFov(C_s, C_r). \end{aligned} \tag{V.7}$$

The handover is then made possible when a receiver is identified:

$$\begin{aligned} 0\{send^t(C_s, C_r, O)\}1 \leftarrow\ & obj^{t-1}(O, A, M), ass^{t-1}(C_s, O), lFov^{t-1}(C_s, O), cam(C_r), \\ & \#count\{C_r : cPfov^{t-1}(C_s, C_r, P, D_{sr}), P = M\} \geq 1. \end{aligned} \tag{V.8}$$

Rule V.8 simply states that: an object O (with appearance A) assigned to a camera C_s (the sender) and which is leaving the coverage area at time $t-1$ will be handed over to a neighbor C_r (the receiver) located in the direction where O is moving; assuming C_r has an overlapping view with C_s. The aggregate *count* in this rule is used to count the number of nodes located at a position $P = M$ relative to C_s.

For an effective handoff, it is ideally expected to find a camera with an overlapping view in the direction where a target is moving. In reality, however, this will not always happen. In such a case, the exiting object (O) will be handed over to any non-overlapping neighbor (C_r) located in the direction (M) of motion:

$$0\{send^t(C_s, C_r, O)\}1 \leftarrow obj^{t-1}(O, A, M), ass^{t-1}(C_s, O), lFov^{t-1}(C_s, O), cam(C_r), \\ \#count\{C_r : cPos^{t-1}(C_s, C_r, P_r, D_{sr}), P_r = M\} \geq 1, \\ \#count\{C_i : cPfov^{t-1}(C_s, C_i, P_i, D_{si}), P_i = M\} = 0. \quad \text{(V.9)}$$

with atom $cPos()$, instead of $cPfov$, now used as the selection criteria for the set of receiver candidates.
Now, if there is no camera (C_i) in the direction (M) where the target (O) is moving, this latter will be broadcast to any node (C_r) in the neighborhood, even if located in an opposite direction:

$$0\ \{send^t(C_s, C_r, O)\}\ 1 \leftarrow obj^{t-1}(O, A, M),\ ass^{t-1}(C_s, O),\ lFov^{t-1}(C_s, O), \\ cam(C_r),\ \#count\{C_r : cPos^{t-1}(C_s, C_r, P_r, D_{sr})\} \geq 1, \\ \#count\{C_i : cPos^{t-1}(C_s, C_i, P_i, D_{si}),\ P_i = M\} = 0. \quad \text{(V.10)}$$

In this case, the handoff direction is not important anymore. The objective now is to minimize the risk of losing a target even at the cost of increasing the amount of network communication messages.
With Rules V.8-V.10 (and the lower bound of header aggregates being equal to 0), it is possible to generate configurations where a leaving target is not handed over. Such scenarios must be avoided by guaranteeing that the total amount of transfer (when a target is leaving) is different than zero (Constraint V.12).

$$tSendN^t(C_s, C_r, O, N) \leftarrow lFov^{t-1}(C_s, O),\ cam(C_r), \\ N = \#count\{O : send^t(C_s, C_r, O)\}. \quad \text{(V.11)}$$

$$\leftarrow lFov^{t-1}(C_s, O),\ \#sum\{N : tSendN^{t-1}(C_s, \_, O, N)\} = 0. \quad \text{(V.12)}$$

Whenever atom $send()$ is derived on a node, appropriate actions are taken on the embedded processor to actually transfer object properties to the target camera.

Target Entering the FoV of a Camera At runtime, every camera manages two lists of objects: the list of local targets under tracking (or active targets) and the list of received objects. Whenever a new object enters a camera FoV, its appearance (A_n) is search within the list of active targets. If there is a match with an object (O) already being tracked, then no further action is required –meaning that the assignment will still hold at the next time-stamp:

$$ass^t(C,O) \leftarrow eFov^{t-1}(C,A_n,S), obj^{t-1}(O,A,M), A = A_n, ass^{t-1}(C,O). \tag{V.13}$$

This scenario might be the result of targets exiting and immediately re-entering a camera region before completion of the handover process.
Otherwise, if the object is not under tracking but is found in the list of received features, then it is a target coming from a neighbor node. The object features –only the OIN (O_r) in our case– are therefore updated to match received properties for consistent labeling (Rule V.14) and a new tracking assignment is created on the current camera (Rule V.15):

$$1\ \{obj^t(O_r,A_r,none)\}\ 1 \leftarrow eFov^{t-1}(C,A_n,S), saved^{t-1}(O_r,A_r),\ A_r = A_n,\ not\ obj^{t-1}(O_r,A_r,\_). \tag{V.14}$$

$$1\ \{ass^t(C,O_r)\}\ 1 \leftarrow eFov^{t-1}(C,A_n,S),\ saved^{t-1}(O_r,A_r),\ A_r = A_n,\ not\ ass^{t-1}(C,O_r). \tag{V.15}$$

with atom *saved*() being used to capture object features (OIN and appearance) received from neighbors that are locally saved into memory.
Finally, if no received features match the new object, it will be given a new OIN (Rule V.16) and assigned to the monitoring camera (Rule V.17):

$$1\ \{obj^t(O_n,A_n,none)\}\ 1 \leftarrow eFov^{t-1}(C,A_n,S),\ O_n = C \times 100 + A_n,\ not\ saved^{t-1}(\_,A_n),\ not\ obj^{t-1}(\_,A_n,\_). \tag{V.16}$$

$$1\ \{ass^t(C,O_n)\}\ 1 \leftarrow eFov^{t-1}(C,A_n,S),\ O_n = C \times 100 + A_n,\ not\ saved^{t-1}(\_,A_n),\ \#count\{O : ass^{t-1}(C,O),\ obj^{t-1}(O,A,\_),\ A = A_n\} = 0. \tag{V.17}$$

From these rules, note that the unique OIN (O_n) is generated by using each current CIN (C) as the base address added to the object appearance.

c) - Target Redistribution After Node Failure

Recovering from runtime camera failure in a distributed network is usually a very complex undertaking. In the proposed work, the main objective is not to fully repair the network after a node failure –since node healing is not addressed– but to elaborate a cost-effective, distributed and self-organized target recovering from the damaged node to its neighbors with the goal of minimizing the amount of lost targets. This problem is identical to the initial camera-target assignment problem with the only difference that objects to consider are now the targets to be recovered and the participating cameras are the nodes belonging to the direct neighborhood of the broken entity. Figure V.4 depicts this reassignment process applied on a 4-node network scenario.

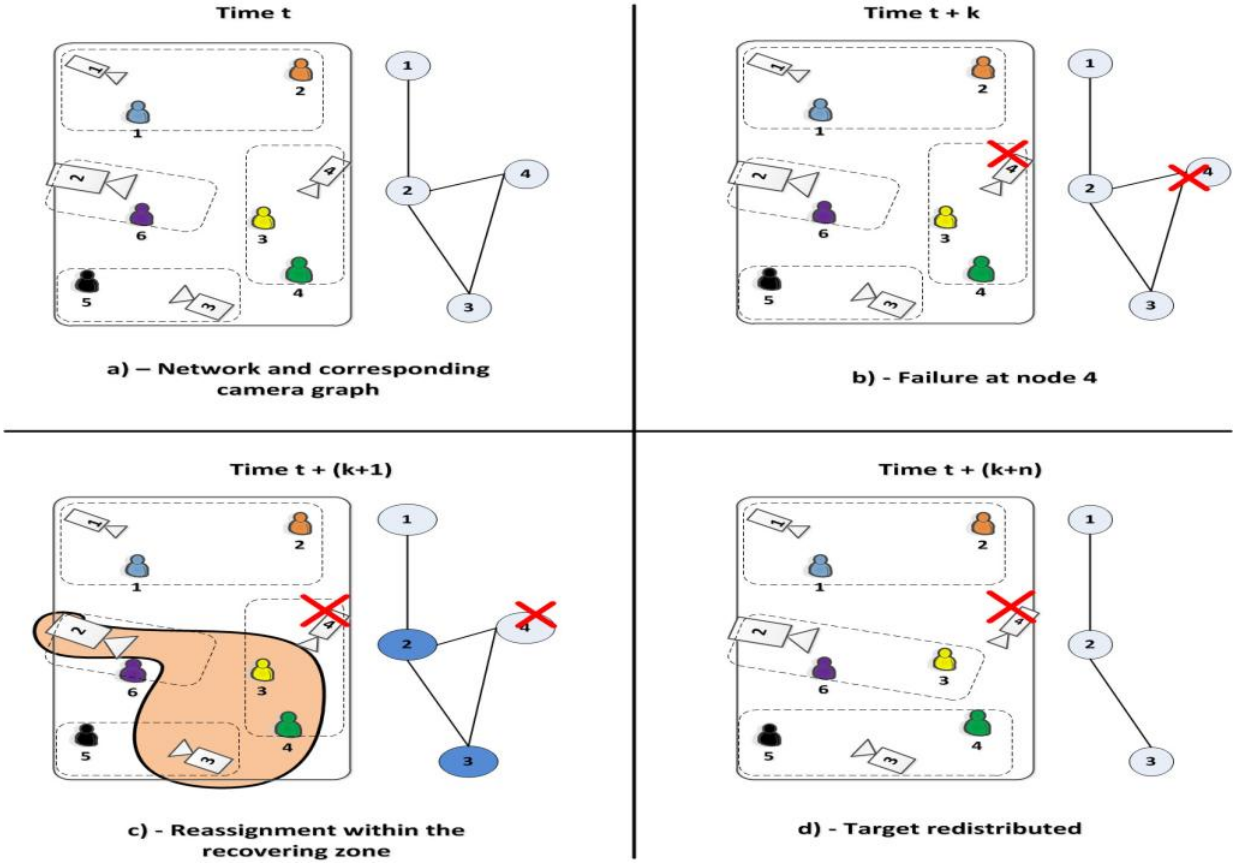


Figure V.4: The target recovering process after node failure. (a) The distributed Network with the corresponding camera interconnection graph. (b) Failure on node 4 detected at time $t + k$. (c) Beginning of target reassignment process within the recovering zone. (d) Network state after reassignment with updated camera graph.

On the proposed scenario of Figure V.4, after detecting the failed node 4 at time $t + k$,

the reassignment process starts immediately within the *recovering zone* (RZ). This region is made up of targets (3 and 4) previously assigned to node 4 plus camera 2 and camera 3 that were part of the failed node direct neighborhood. However, it might happen that some nodes within RZ are already overwhelmed at the time of failure and could not take part in the redistribution. Such nodes, once identified (using atom $full()$), will be exempted from participating in the redistribution.

$$full^t(C) \leftarrow fail^t(C_f),\ cam(C),\ C \neq C_f,\ tracN^t(C,N), \\ mObj(C, C_{max}), N = C_{max}. \tag{V.18}$$

In Rule V.18, a node C is considered overwhelmed when its runtime tracking activity has reached its maximum tracking capacity.
Now, objects and cameras inside RZ will be mapped following a process similar to our initial camera-target assignment but now with the unique goal of balancing the distribution. Consequently, for each camera C (not overwhelmed) and each target O in RZ, at most one assignment is generated:

$$0\ \{send^t(C_f, C, O)\}\ 1 \leftarrow not\ full^{t-1}(C), fail^{t-1}(C_f), ass^{t-1}(C_f, O), \\ cPos^{t-1}(C_f, C, P, D), D \neq 0. \tag{V.19}$$

Rule V.19 states that if a node C (not overwhelmed) detects a failed neighbor C_f at time $t-1$ to which a target O was assigned, then this latter is reassigned to C after failure. By using atom $send()$ instead of $ass()$, target O is not directly mapped with camera C at time t, but the reassignment operation is encoded as the object leaving camera C_f to node C. This is similar to node C_f releasing/distributing all its targets right before the failure. Since, there is no guarantee at the time of failure that the recovered object O is already within camera C monitoring region, it cannot be immediately considered as a tracking target. Upon entering or identification within node C FoV, the tracking assignment of O will become effective (see Rules V.13-V.17).
With Rule V.19 (and the lower bound of the header aggregate being equal to 0), it is possible to produce configurations where no target are redistributed. Such scenarios will be avoided by guaranteeing that, after a node failure, the total amount of redistribution is

different than zero (Constraint V.21).

$$tSendfN^t(C_f, C, N) \leftarrow fail^{t-1}(C_f),\ cPos^{t-1}(C_f, C, P, D),\ D \neq 0,\ not\ full^{t-1}(C),$$
$$N = \#count\{O : send^t(C_f, C, O)\}. \tag{V.20}$$
$$\leftarrow fail^{t-1}(C_f),\ \#count\{O : ass^{t-1}(C_f, O)\} \geq 1,$$
$$\#sum\{N : tSendfN^{t-1}(C_f, \_, N)\} = 0. \tag{V.21}$$

To avoid a duplicate tracking, it should be insured that the same target O is not redistributed to more than one camera:

$$\leftarrow fail^{t-1}(C_f),\ ass^{t-1}(C_f, O),\ send^t(C_f, C_i, O),\ send^t(C_f, C_j, O),\ C_f \neq C_i \neq C_j. \tag{V.22}$$

Finally, the distance to the failed node is set to zero to symbolize the failure:

$$cPos^t(C_f, C_i, P, 0) \leftarrow fail^{t-1}(C_f), cPos^{t-1}(C_f, C_i, P, D), D \neq 0. \tag{V.23}$$
$$cPos^t(C_i, C_f, P, 0) \leftarrow fail^{t-1}(C_f), cPos^{t-1}(C_i, C_f, P, D), D \neq 0. \tag{V.24}$$

The final step in the solver modeling is to find the best configuration, which is devised by narrowing the set of candidate solutions according to predefined search objectives.

Minimizing the Load Balance among Cameras The objective in runtime distribution is to insure a good network coverage and also a fair load balancing among nodes of the network. The optimal configuration is seen as the scenario that would minimize not only the load balance among cameras at runtime but also the inter-camera distance during target transfer. Whenever targets are distributed, it should be guaranteed that a node is not overloaded while its neighbors are idle.
Our modeling will start by computing the *future tracking activity* (FTA) of each camera C at time t, which is the current tracking activity (N_c) of the node, to which has been removed the amount of released targets (N_r) at time t:

$$fTrac^t(C, N) \leftarrow cam(C), tracN^{t-1}(C, N_c), N_r\ =\ \#count\{O : send^t(C, C_r, O)\},$$
$$N = N_c - N_r. \tag{V.25}$$

Now, the tracking activity among cameras in the same neighborhood should be balanced (during transfer) so as to minimize load disparity. This amounts to compute, at time t,

the absolute difference between the tracking activity of two neighbor nodes (Rule V.26). This computation is made possible by the fact that each node is aware of its neighborhood tracking activity.

$$balance^t(C, C_i, N_b) \leftarrow cam(C),\ cam(C_i),\ fTrac^t(C, N),\ tracN^{t-1}(C_i, N_i), \\ N_f = \#count\{O : send^t(C, C_i, O)\}, N_b = |N - (N_i + N_f)|. \quad \text{(V.26)}$$

In Rule V.26, observe that to compute the balance with a node C_i, all future transfers (N_f) to this node are also considered.
Then, the best configuration at time t would be the scenario that minimizes not only the sum of load difference among cameras but also the handover distance among cameras exchanging targets:

$$\begin{aligned} &\#\text{minimize}\{ \\ &\quad N@2, balance : balance^t(C_i, C_j, N); \\ &\quad D@1, cPos : cPos^t(C_i, C_j, L, D), send^t(C_i, C_j, O), D \neq 0\}. \end{aligned} \quad \text{(V.27)}$$

Having load balancing as the highest priority ($p = 2$) will force the target distribution to be even-handedly applied to all nodes during handover.

V.4.4 Runtime Feature Update

In this step, outputs of the logic optimization are interpreted. In the proposed model, two atoms are of particular interest: $ass()$ and $send()$. Whenever atom $ass^t(C, O)$ is produced, it refers to a new assignment instruction. Consequently, object O will be added to the list of tracking targets on the current camera C (on which rules are executed). For atom $send^t(C_s, C_r, O)$, if the receiver C_r is the current camera, then object O is a recovered target (see Rule V.19). Its properties will be locally saved as having been received from a neighbor until the object is detected within the coverage area of node C_r and effectively assigned. If C_s is the current camera, then this is a normal target exiting scenario. Object properties in this case will be fetched from the memory and transmitted to the indicated node C_r. Once objects have been assigned to cameras in the network, each node is responsible for tracking and updating its target features as they locally move within the coverage area.

V.5 Experimental Results

In this section, results of running experimentations are presented. The camera prototype used for our embedded systems is the RazorCam [50] (Figure V.5 left), a smart embedded camera for video processing. The RazorCam is a flexible and extensible

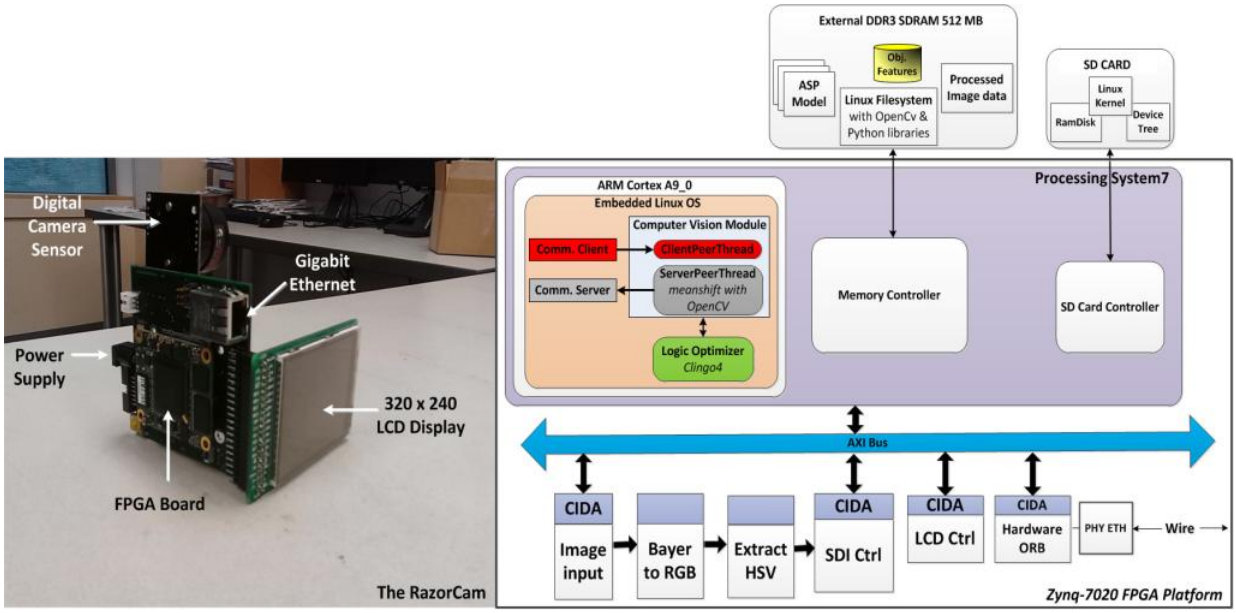


Figure V.5: The experimental platform. The RazorCam system is presented on the left (photo taken by author). The hardware/software internal structure of the camera is presented on the right.

Hardware/software environment for the prototyping of video applications. It is capable of processing image data through a Xilinx Zynq 7020 FPGA-board featuring an ARM dual processor, which runs an embedded Linux operating system. The processor frequency is 667 MHz and the FPGA board features an off-chip 512 MB DDR3 SDRAM as storage unit, with a Gigabit Ethernet Transceiver for communication. The color conversion, the segmentation, and some filtering operations are implemented as hardware units for performance improvement and communicate with the general processor and the main memory through a high-speed Xilinx AXI bus.

Upon request of a new frame from the processor (SPT component in Figure V.5(right)), Bayer input pixels from the digital camera are converted (in hardware) into an RGB representation that is passed to a Histogram extractor module. This module will create a confidence map based on color histogram of objects in the RGB image and saved the map in the external memory using the SDI controller component. The confidence map is then

read in software by the SPT module which will use (a modified) mean shift algorithm implemented with OpenCV to estimate object motion inside the frame. Some characteristics of the proposed SoC are summarized in Table V.1.

Software	
ARM CPU frequency	667 MHz
ASP Solver	39 lines
Size of cross-compile Clingo4	3.0 Mb
Size of the embedded logic optimizer	3.1 Kb
Executable size (SPT)	43.1 Kb
Frame memory usage	225 Kb
Size of the extracted features	1.6 Kb/Target

Table V.1: Characteristics of the feature extraction algorithm.

The image resolution of our embedded camera is 320×240; resulting in a memory usage of 225 Kb. The size of the cross-compile executable SPT is about 43.1 Kb. Extracted features are: a 3-channel color Histogram, objects direction, and bounding boxes around targets, with a total size of 1.6 Kb per object. In average, the SPT can process about 3 frames per second. This slow performance is mostly due to the adapted mean shift step being executed in software on the embedded Linux. The cross-compiled Clingo tool for evaluating events at runtime has a size of only 3.0Mb and the ASP solver model presented in Section V.4.3 is only about 39 lines/rules, which shows the compactness of the proposed model.

Table V.2 gives the overall resource usage by the proposed architecture. Not all resources have been listed but only the most relevant ones. The maximum frequency of the system obtained after logic synthesis is 106.32 MHz.

Several aspects of the proposed architecture could be evaluated, from the runtime computational performance, the power consumption, the design efficiency (in term of resource), to the hardware/software task swapping or runtime reconfiguration properties. However, we limited our evaluation to experiments that will demonstrate the effectiveness of the proposed declarative modeling approach (using the optimization engine) to allow runtime self-coordination of embedded smart cameras in a distributed network setup. Therefore, experiments have been conducted in two phases aiming at testing the novel

Components	Slices	LUTs	RAMs
Overall system	9490 (9%)	20058 (37%)	31 (22%)
Image input	109 (0%)	289 (0%)	0 (0%)
Bayer2RGB	541 (0%)	93 (0%)	0 (0%)
SDI Ctrl	1646 (1%)	1789 (3%)	1 (0%)
LCD Ctrl	156 (0%)	361 (0%)	24 (17%)
ExtractHSV	67 (0%)	152 (0%)	0 (0%)
HardwORB	4829 (4%)	14047 (26%)	1 (0%)

Table V.2: FPGA (Zynq7020clg484) resource usage by the proposed system-on-chip.

aspects of the proposed coordination approach:

- **Phase I - System Reactivity**: in this phase we aim at demonstrating the feasibility of the proposed modeling approach on a network where events occur randomly over time.
- **Phase II - System Robustness**: decentralized object distribution is a crucial step in distributed tracking system. This phase will evaluate the robustness of the proposed coordination approach on a network setup as the amount of target/camera increases.

V.5.1 Phase I - System Reactivity

To evaluate the feasibility of the proposed modeling approach, we tested a distributed tracking scenario on the 4-node network setup of Figure V.6, with the results provided in Table V.3. The objective in this experimental phase was mainly to observe the system reactivity (camera self-coordination) when random events occurred in the network environment. The executed scenario will include events like node failure or target entering/exiting a camera region. To eliminate the effect of image processing (e.g. object detection and tracking), we directly provided the different cameras with appropriate events, listed in column 3 of Table V.3; thus bypassing the image processing step. The evaluation time can be observed in column 5 of Table V.3. This is the processing time of the optimization engine to evaluate accumulated events on the participating cameras (Column 2 of Table V.3). Since events are decentrally evaluated, the time provided in

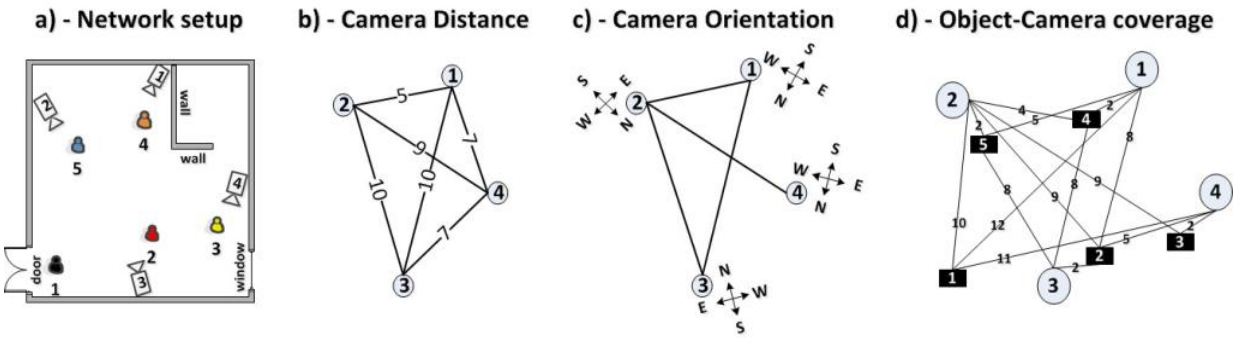


Figure V.6: Experimental network setup. a) The network layout. b) The inter-camera distance graph. c) The camera orientation graph (from the camera perspective). d) The camera-target coverage graph at start-up.

Table V.3 is the reaction time of the slowest node. This time does not include the communication delays between cameras. In this phase, the tracking capacity for all nodes in the network has been set to 4. For more consistency, the provided evaluation time has been averaged over 10 successive runs –number of executions per event.
Figure V.7 shows the corresponding network setup after event evaluation:

- At initialization ($T = 0$), targets are decentrally assigned to the four cameras in less than 3 seconds (b).
- At time $T = 2$, object 4 exits camera 1 in the direction East and is automatically transfered to node 4 at time $T = 3$ (c). This evaluation takes approximately 690 ms.
- At $T = 4$, an entering object (with feature 5) is detected by camera 4 from West, while target 5 is leaving camera 2 FoV. After evaluation of event, node 4 realizes that the entering object is target 4 previously sent by camera 1 and the tracking assignment is done at $T = 5$ (d). Meanwhile, leaving object 5 is transfered by camera 2 to node 3.
- At $T = 6$, an unknown object (with appearance 11) enters the room and is detected by camera 2 and assigned a new ID 211 at $T = 7$ (e).
- At $T = 8$, an entering object is identified at node 3 as target 5 previously sent by camera 2 (f).
- At $T = 10$, camera 2's failure is notified to all its neighbors $(1, 3, 4)$ in the network. This will immediately trigger the recovering process that will take about 1 second to

T	Cam	Events	Clingo outputs	Eval. time(s)
0	1,2 3,4	initialization	$ass^0(1,4)$,$ass^0(2,1)$, $ass^0(2,5)$, $ass^0(3,2)$, $ass^0(4,3)$	2.91
2	1	$lFov^2(1,4)$	$send^3(1,4,4)$	0.69
4	2 4	$lFov^4(2,5)$, $eFov^4(4,5,west)$	$send^5(2,3,5)$, $ass^5(4,4)$	1.14
6	2	$eFov^6(2,11,north)$	$ass^7(2,211)$	0.56
8	3	$eFov^8(3,8,north)$	$ass^9(3,5)$	1.16
10	1,3 4	$fail^{10}(2)$	$send^{11}(2,1,211)$, $send^{11}(2,1,1)$	1.13
12	1	$eFov^{12}(1,11,north)$	$ass^{13}(1,211)$	1.14
14	1	$eFov^{14}(1,2,north)$	$ass^{15}(1,1)$	1.30
15	1,3,4		$ass^{15}(1,211)$,$ass^{15}(1,1)$, $ass^{15}(3,2)$, $ass^{15}(3,5)$, $ass^{15}(4,3)$, $ass^{15}(4,4)$	

Table V.3: A distributed tracking scenario with node failure.

complete (g). After target redistribution, targets 1 and 211 are both reassigned to camera 1, event though node 3 and 4 participated in the recovering process. The reason being that at the time of failure, nodes 3 and 4 are already tracking two objects each while camera 1 is idle. The load balance objective will force the redistributed targets to be assigned to the camera with the less overload. On camera 1, features of the indicated targets will be saved until they are locally detected.

- At $T = 12$ and At $T = 14$, targets 211 and 1 respectively enter camera 1 monitoring area and are immediately tracked (h-i).

V.5.2 Phase II - System Robustness

Decentralized object distribution is a crucial step in distributed tracking system. In this phase of our experiments, we evaluate the robustness of the proposed approach regarding the object distribution and runtime handover. The goal here is to measure how quick all

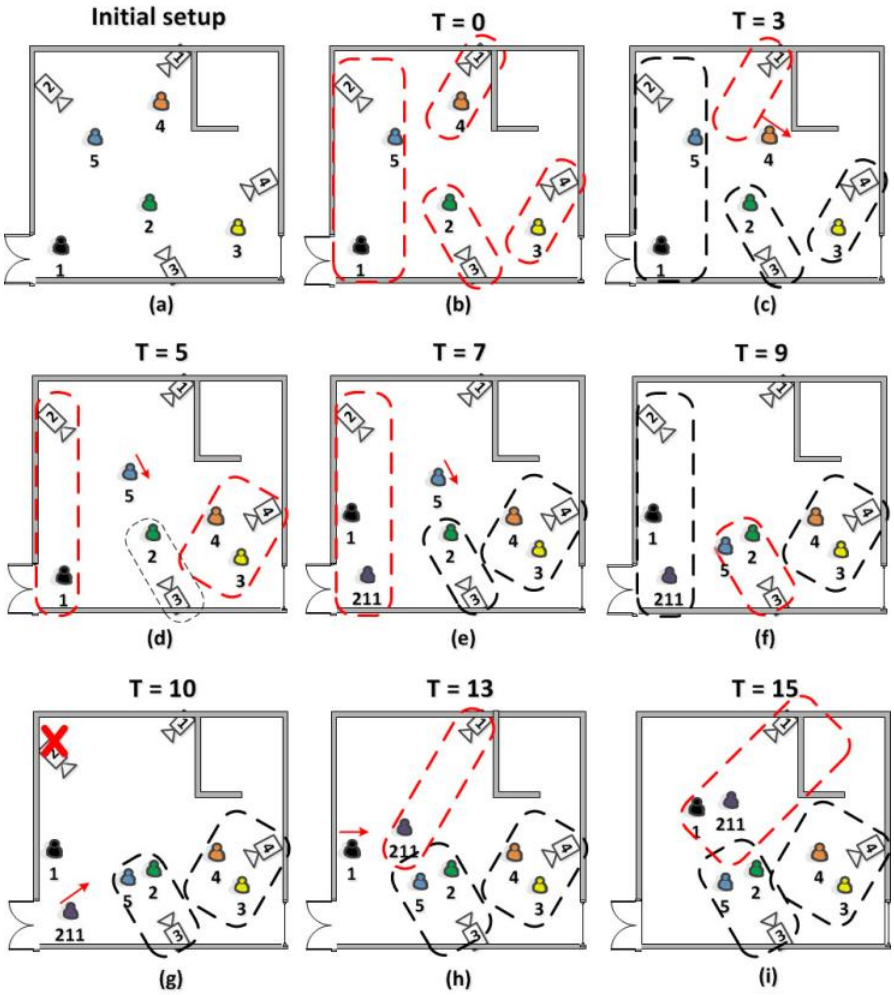


Figure V.7: Network setup after events have been evaluated.

camera nodes will decentrally agree on object assignments and distribution as these move randomly across the network. During experiments, we will observe the camera-target assignment and the handoff time when the size of a network (amount of cameras and objects) increases.

Camera-Target Assignment. Figure V.8 shows the results of our experiments regarding the camera-target assignment process on networks with 2 to 8 cameras and targets, emulated using QEMU. The network topology is a mesh (nodes strongly connected) and target positions have been randomly assigned at initialization. The provided evaluation time has been averaged over 2 runs for result consistency. In all these scenarios, we always assume the *worst coverage case* where each node can cover all

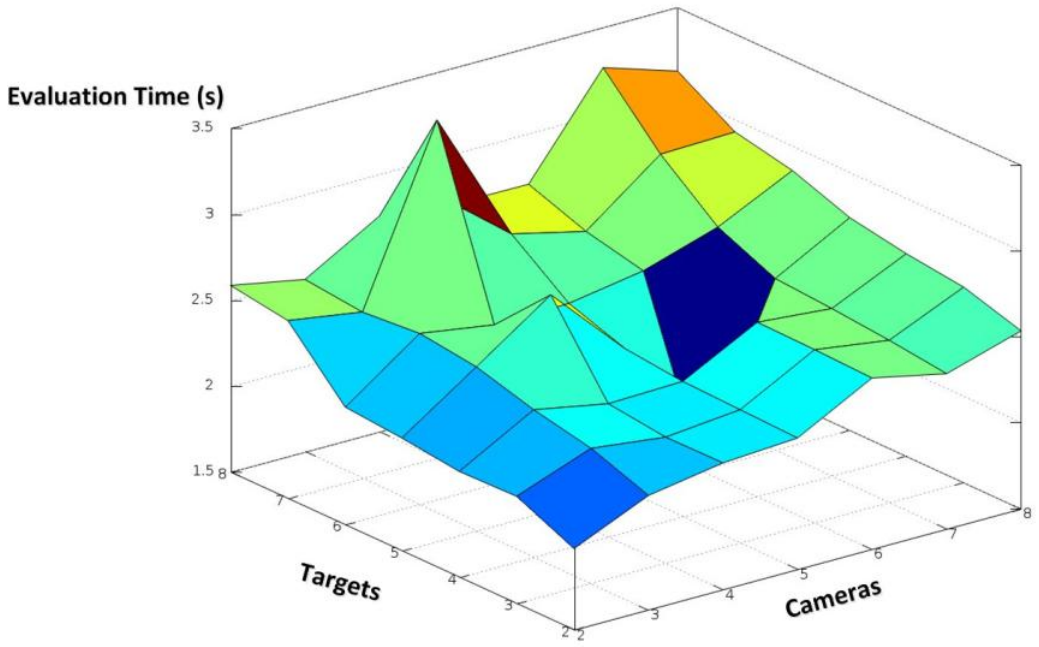


Figure V.8: Evaluation of the camera-target distribution process at initialization.

objects in the environment, with the tracking capacity on each camera being unlimited. These assumptions aim at maximizing the number of possible configurations (exploration space) and thus, the search time.

As could be observed on Figure V.8, it takes approximately 3.13 seconds for a set of 8 cameras to decentrally agree on a distribution of 8 targets. In other words, the proposed methodology is able to search over a design space of more than 40000 possible configurations in less than 3.5 seconds. In comparison, the same setup takes 40 ms with smart cameras running on a Desktop computer featuring an Intel Core i5 2.67GHz processor with 3.8Gb of memory.

However, it is difficult to estimate (and project) the behavior of the evaluation time function from the image of Figure V.8, since it does not depict a clear shape. To address this, we reevaluated the same network scenarios and increase the number of runs per scenario from 2 to 5. Figure V.9 shows the results of evaluations.

It could be observed from Figure V.9 that when the number of execution per scenario increases, the shape of the image slowly becomes uniform and exhibits a linear shape,

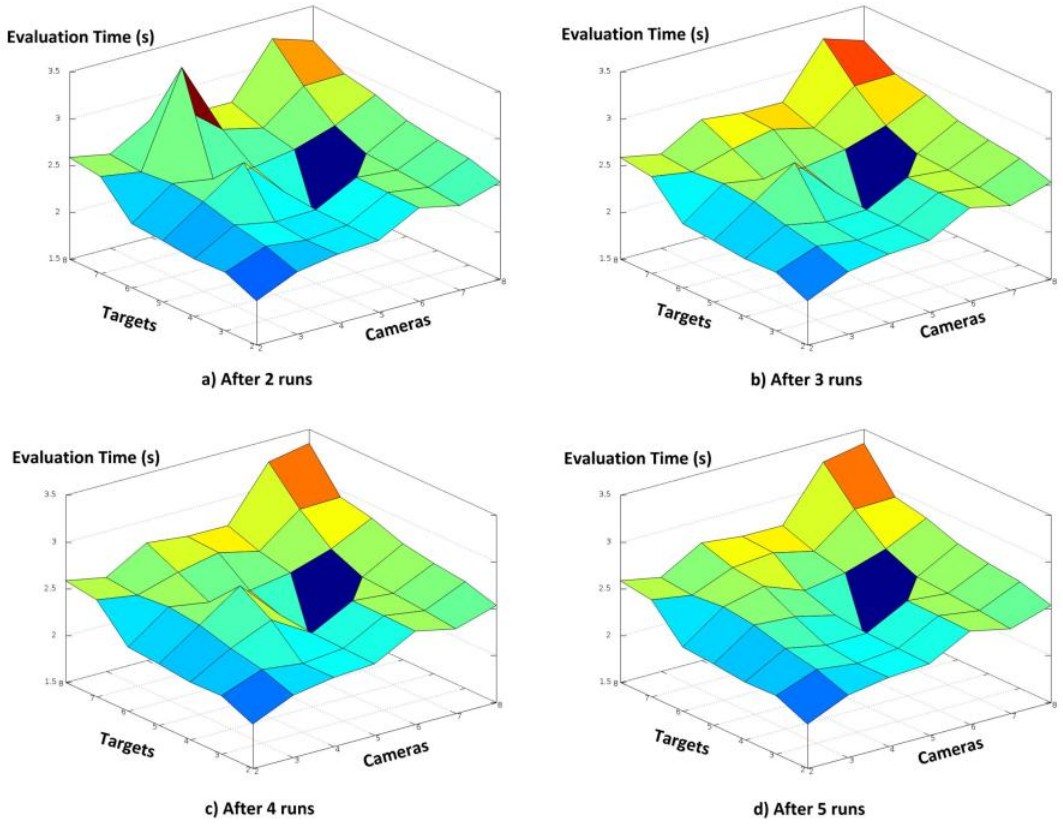


Figure V.9: Study of the evaluation time during camera-target distribution.

that is, the evaluation time increases linearly with the network size and not exponentially as it would have been the case with another search heuristic methodology such as ILP.

Camera Handoff. We evaluated the effectiveness of the handoff time on network scenarios with 2, 4, and 6 cameras respectively, as shown in Figure V.10. The camera orientation and mesh topology of the proposed networks have been chosen in order to maximize the search time during handoff. In all scenarios, targets (with their motion) have been randomly assigned to nodes.

Figure V.11 shows the execution time of the handoff process, average over 5 runs. The horizontal axis of the figure indicates the amount of targets that are simultaneously handed over at a particular time. It takes less than 5 seconds to a set of 6 distributed camera nodes to successively complete the handoff process of 30 moving targets. On the Desktop station, the same scenarios has been solved after 60ms. With unlimited tracking capacity on each camera node, the search space is maximized and thus, the generated

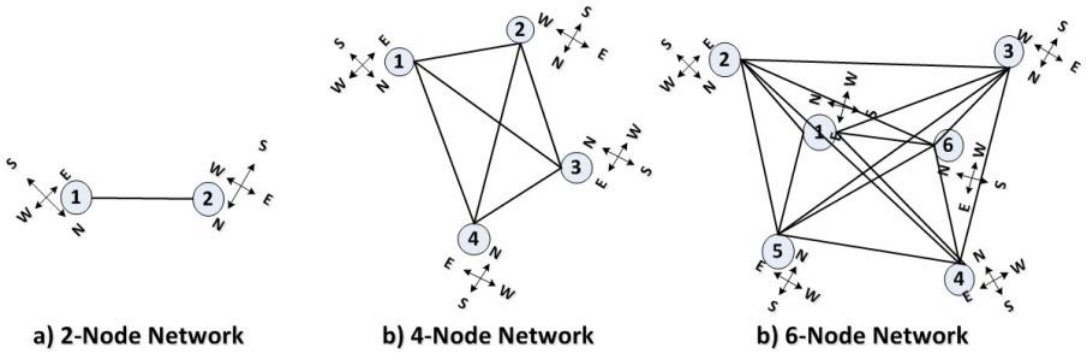


Figure V.10: Network topologies for evaluating camera handoff.

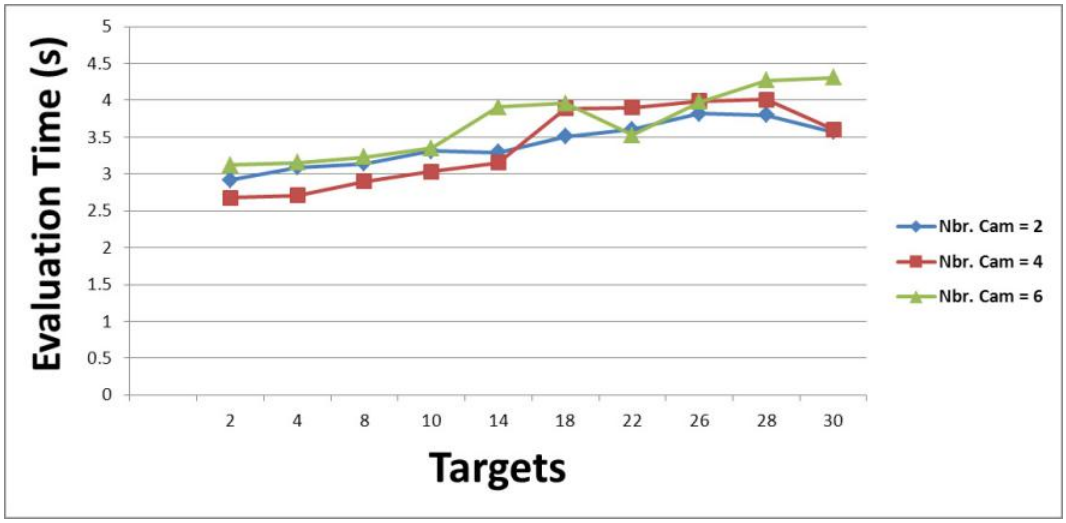


Figure V.11: Evaluation time during camera handoff.

handoff time. In reality, this space will be smaller given the resource constraints on embedded cameras. Additionally, it is worth noting that the handoff process is linear, just like with the camera-target assignment. When using a search methodology, it is always important to guarantee that the exploration time is not exponential. With the linearity of the proposed handoff process –if confirmed on a bigger network scale, improving the tracking efficiency of distributed smart camera nodes will amount to improve the image processing aspect on each node by pushing more computation in hardware for acceleration (such as the mean shift computation in the proposed SoC architecture).

V.6 Conclusion

In this chapter, we presented a compact and robust declarative model for self-coordination of object handover in a distributed network of embedded smart cameras. The model leverages Answer Set Programming, a logic programming paradigm, to encode the decentralized tracking of moving targets in a network as a search problem that is solved dynamically by an embedded answer set solver engine. In the proposed approach, a camera in the network will rely on the knowledge of tracking activities in its direct surrounding to decentrally hand over targets as they move across its FoV. In case of node failure, cameras in the immediate neighborhood of the failed node will automatically start a target recovering process and manage object transfer without any human intervention. Experiments have proven the feasibility and the robustness of our solution on different network scenarios with up to 6 cameras nodes and 30 targets. The tracking system has been design for embedded platforms and leverages an FPGA fabric to accelerate the image processing performance. Unfortunately, the inaccuracy of video processing, leading to many false positives in extracting events, and the slow image processing performance due to software implementation of the mean-shift algorithm, made it almost impossible to evaluate the effectiveness of the proposed approach on a real-life network setup.

VI Summary and Future Work

This dissertation provided significant contributions to the self-coordination of distributed smart embedded cameras in a network setup. We proposed a clear and compact formalism using answer set programming, a declarative programming paradigm, for dynamically and decentrally capturing and solving auto-coordination problem on each camera node of the network. Given resource and power limitations on embedded camera platforms and the processing requirements of embedded computer vision, we addressed the hardware and software aspect of the camera architecture in tandem by using a multi-objective system-on-chip synthesis methodology. Besides being resource and power efficient, the resulting camera system is optimized to implement computational intensive processing in hardware, while high level reasoning tasks are kept in software.

After briefly introducing the answer set programming concept in Chapter II, the proposed declarative-based modeling approach for the synthesis of system-on-reconfigurable-chip is presented in Chapter III. Implementing architecture for reconfigurable embedded platforms is very challenging given all design parameters and constraints that must be simultaneously addressed (system performance, resource constraints, power limitation). The synthesis has been encoded as a mapping problem between a set of tasks and a set of resources, with the goal of generating the optimal mapping configuration regarding user objectives. Using answer set programming, the proposed methodology overcame the issue of size explosion and exponential synthesis time encountered with conventional synthesis approaches such as Integer Linear programming or Constraint programming. Moreover, the proposed approach is network centric, that is, smart camera architectures are generated so as to minimize the cost of communication resource in the network, without performance degradation. The feasibility of the proposed synthesis method has been demonstrated on several network scenarios, in which it has been showed that it is possible to synthesize architectures for a 10-node network –with 10 tasks per node– in less than 40s.

In chapter IV, we presented the generated system-on-chip in detail. The proposed architecture has been designed to tackle the performance issue of embedded computer vision through a hardware/software decomposition approach. Such a decomposition

allows computational intensive blocks of a processing system to be accelerated in hardware, while the control parts remain in software. Considering the resource and power constraints on embedded systems, we implemented the proposed system on an FPGA-based platform to leverage runtime hardware restructuring and task swapping properties of such technologies. Additionally, the Linux operating system with OpenCV and Python libraries have been integrated into the camera system in order to increase system programmability and to allow developers to easily migrate their existing applications into the proposed camera platform.

Finally, chapter V presented the proposed declarative-based model for capturing runtime self-coordination in a distributed network setup. In contrary to existing approaches, our methodology, borrowed from high-level synthesis, encoded a tracking problem as a design space exploration that systematically evaluates all possible coordination scenarios and select the optimal choice regarding load balancing among nodes of the network. The approach is distributed, meaning that, each camera is equipped with sufficient reasoning capabilities to autonomously extract and evaluate meaningful information in its monitoring area and devise the appropriate reaction in case of target leaving or entering the field-of-view. The feasibility and robustness of the decentralized coordination method have been demonstrated on various network scenarios, where it has been shown that it was possible to self-coordinate up to 30 moving targets in less than 5 seconds in a 6-node network. Also, the linearity of the proposed approach as a search methodology has been demonstrated.

Several promising research avenues are suggested by the work presented in this dissertation, and a few of them are detailed here. First, to match with reality regarding existing surveillance systems, such as in banks, nuclear plants, or airports, the performance of the proposed coordination approach would be studied on larger networks of up to fifty or hundred nodes. Then, addressing video processing issues encountered during experiments would be considered to boost the camera performance. Finally, we will also consider improving the coordination model to allow a priority-based target tracking. In such a case, targets will be tracked among cameras based on priority level assigned to tracking events, with the possibility given to each node to release formerly assigned targets (with a less priority) if capacity concern.

Bibliography

[1] C. Savage, "US Doles out Millions for Street Cameras, Local Efforts raise Privacy Alarms," *The Boston Globe*, August 2007.

[2] A. Greggo and M. Kresevich, *Retail Security and Loss Prevention Solutions*. London: CRC Press Taylor and Francis Group, 2011.

[3] T. A. Press, "Wal-mart Losing $3 Billion a year from Thefts," http://archive.azcentral.com/business/consumer/articles/0613biz-walmarttheft13-ON.html, June
2007.

[4] I. Jack L. Hayes International, "Annual retail theft survey 2013," http://hayesinternational.com/news/annual-retail-theft-survey/, 2013.

[5] J. Chen, L. Yip, J. Elson, H. Wang, D. Maniezzo, S. Member, R. E. Hudson, K. Yao, and D. Estrin, "Coherent Acoustic Array Processing and Localization on Wireless Sensor Networks," in *Proc. the IEEE*, 2003, pp. 1154–1162.

[6] P. J. F. Carle and T. D. Barfoot, "Global Rover Localization by Matching LIDAR and Orbital 3D Maps," in *ICRA*. IEEE International Conference, 2010, pp. 881–886.

[7] T. Ainsworth, "Buyer Beware," *Security Oz.*, pp. 18–26, 2002.

[8] Y. M. Mustafah, B. A. W. Azman, A. Bigdeli, and B. C. Lovell, "An Automated Faced Recognition System for Intelligence Surveillance: Smart Camera Recognizing Faces in Crowd," 2007.

[9] A. Karimaa, *Efficient Video Surveillance: Performance Evaluation in Distributed Video Surveillance Systems*. INTECH Open Access Publisher, 2011.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[12] R. E. Perez and K. Behdinan, "Particle Swarm Approach for Structural Design Optimization," *Comput. Struct.*, vol. 85, no. 19-20, pp. 1579–1588, Oct. 2007.

[13] M. Dorigo, M. Birattari, and T. Stützle, "Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique," *IEEE COMPUT. INTELL. MAG*, vol. 1, pp. 28–39, 2006.

[14] J. S. Rosenschein, "Rational Interaction: Cooperation among Intelligent Agents," Ph.D. dissertation, Stanford, CA, USA, 1986, uMI order no. GAX86-08219.

[15] F. Castanedo, J. Garcia, M. A. Patricio, and J. M. Molina, "Data Fusion to Improve Trajectory Tracking in a Cooperative Surveillance Multi-Agent Architecture," *Information Fusion*, vol. 11, no. 3, pp. 243 – 255, 2010.

[16] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof, and B. Strobl, "Autonomous Multicamera Tracking on Embedded Smart Cameras," *EURASIP J. Emb. Sys.*, 2007.

[17] F. Z. Qureshi, "Collaborative Sensing via Local Negotiations in Ad Hoc Networks of Smart Cameras," in *Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras*, ser. ICDSC '10, 2010, pp. 190–197.

[18] W. Starzyk and F. Z. Qureshi, "A Negotiation Protocol with Conditional Offers for Camera Handoffs," in *Proceedings of the International Conference on Distributed Smart Cameras*, ser. ICDSC '14, 2014, pp. 17:1–17:7.

[19] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner, "Socio-economic Vision Graph Generation and Handover in Distributed Smart Camera Networks," *ACM Trans. Sen. Netw.*, vol. 10, no. 2, pp. 20:1–20:24, Jan. 2014.

[20] J. A. Kalomiros and J. Lygouras, "Design and Evaluation of a Hardware/Software FPGA-based System for Fast Image Processing," *Microprocess. Microsyst.*, vol. 32, no. 2, pp. 95–106, Mar. 2008.

[21] V. Isler, S. Khanna, J. Spletzer, and C. Taylor, "Target Tracking with Distributed Sensors: The Focus of Attention Problem," *Computer Vision and Image Understanding Journal*, no. 1-2, pp. 225–247, November 2005.

[22] V. Lifschitz, "Answer Set programming and Plan Generation," *ARTIFICIAL INTELLIGENCE*, vol. 138, p. 2002, 2002.

[23] M. Gelfond and V. Lifschitz, "Classical Negation in Logic Programs and Disjunctive Databases," *New Generation Computing*, vol. 9, pp. 365–386, 1991.

[24] Ishebabi, H., Mahr, P. and Bobda, C., "Automatic Synthesis of Multiprocessor Systems From Parallel Programs under Preemptive Scheduling," in *International Conference on ReConFigurable Computing and FPGAs*, Mexico, December 2008.

[25] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, "An A-Prolog Decision Support System for the Space Shuttle," in *In PADL 2001*. Springer, 2000, pp. 169–183.

[26] E. Irurozki, B. Calvo, and J. A. Lozano, "A Preprocessing Procedure for Haplotype Inference by Pure Parsimony," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, pp. 1183–1195, 2011.

[27] M. Gebser, T. Schaub, S. Thiele, and P. Veber, "Detecting Inconsistencies in Large Biological Networks with Answer Set Programming," *Theory Pract. Log. Program.*, vol. 11, no. 2-3, pp. 323–360, March 2011.

[28] M. Gebser, R. Kaminski, and T. Schaub, "aspcud: A Linux Package Configuration Tool Based on Answer Set Programming," pp. 12–25.

[29] E. Coban, F. Ture, and E. Erdem, "Comparing ASP, CP, ILP on two Challenging Applications: Wire Routing and Haplotype Inference," *In Proc. of LaSh*, 2008.

[30] Wikipedia, "Boolean Satisfiability Problem," http://en.wikipedia.org/wiki/-Boolean_satisfiability_problem/, 2014.

[31] J. W. Lloyd, *Foundations of Logic Programming, 2nd Edition.* Springer, 1987.

[32] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub, "Asp-core-2 input Language Format," 2013. [Online]. Available: http://arxiv.org/pdf/1403.0541v1.pdf

[33] I. Niemel, P. Simons, and T. Soininen, "Stable Model Semantics of Weight Constraint Rules," in *Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR99)*, vol. 1730. Springer-Verlag. LNAI, 1999, pp. 317–331.

[34] T. Syrjnen, "Lparse 1.0 User's Manual," 2002. [Online]. Available: http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz

[35] B. Meyer and D. Thomas, "Rethinking Automated Synthesis of MPSoC Architectures," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–6, March 2007.

[36] B. K. Dwivedi, A. Kumar, and M. Balakrishnan, "Automatic Synthesis of System on chip Multiprocessor Architectures for Process Networks," pp. 60–65, 2004.

[37] S. Murali, L. Benini, and G. D. Micheli, "An Application-Specific Design Methodology for On-Chip Crossbar Generation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 26, no. 7, pp. 1283–1296, 2007.

[38] H. Ishebabi and C. Bobda, "Automated Architecture Synthesis for Adaptive Multiprocessors on Chip Systems," in *Journal of Microprocessors and Microsystems.* Elsevier Science, February 2009.

[39] B. Dieber, C. Micheloni, and B. Rinner, "Resource-Aware Coverage and Task Assignment in Visual Sensor Networks," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 21, no. 10, pp. 1424–1437, 2011.

[40] H. Aghajan and A. Cavallaro, *Multi-Camera Networks.* Elsevier, 2009.

[41] S. Prakash and A. C. Parker, "Synthesis of Application-specific Heterogeneous Multiprocessor Systems," vol. 16, 1992, pp. 338–351.

[42] C. Krishnendu, I. Sitharama, Q. Hairong, and C. Eungchun, "Grid Coverage for Surveillance and Target Location in Distributed Sensor Networks," *IEEE Transaction on Computers*, vol. 51, no. 12, pp. 1448–1453, 2002.

[43] Y. Osais, M. St-Hilaire, and R. Y. Fei, "The Minimum Cost Sensor Placement Problem for Directional Wireless Sensor Networks." IEEE, 2008, pp. 1–5.

[44] F. Mühlbauer, M. Großhans, and C. Bobda, "Rapid Prototyping of OpenCV Image Processing Applications using ASP," 2011, pp. 16–22.

[45] K. Ivekanandarajah and S. K. Pilakkat, "Task Mapping in Heterogeneous MPSoCs for System Level Design," 2008, pp. 56–65.

[46] D. Bertozzi and A. Jalabert, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, 2005.

[47] B. Rinner, B. Dieber, L. Esterle, P. Lewis, and X. Yao, "Resource-aware Configuration in Smart Camera Networks," pp. 58–65, 2012.

[48] B. Dieber, L. Esterle, and B. Rinner, "Distributed Resource-aware Task Assignment for Complex Monitoring Scenarios in Visual Sensor Networks," *International Conference on Distributed Smart Cameras*, pp. 1–6, 2012.

[49] E. Shen and R. Hornsey, "Local Image Quality Metric for a Distributed Smart Camera Network with Overlapping FOVs," in *Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on*, 2011, pp. 1–6.

[50] M. Mefenza, F. Yonga, and C. Bobda, "RazorCam: A Prototyping Environment for Video Communication," *International Workshop on Mobile Computing Systems and Applications*, February 2013.

[51] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Clingo = ASP + Control: Preliminary Report," *CoRR*, vol. abs/1405.3694, 2014.

[52] H. Ishebabi, P. Mahr, C. Bobda, M. Gebser, and T. Schaub, "Answer Set versus Integer Linear Programming for Automatic Synthesis of Multiprocessor Systems from Real-time Parallel Programs," *International Journal of Reconfigurable Computing*, vol. 2009, pp. 1–6, January 2009.

[53] A. A. Zarezadeh and C. Bobda, "Hardware Middleware for Person Tracking on Embedded Distributed Smart Cameras," *Int. J. Reconfig. Comput.*, vol. 2012, pp. 11:11–11:11, Jan. 2012.

[54] P. Chalimbaud and F. Berry, "Embedded Active Vision System Based on an FPGA Architecture," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 26–26, Jan. 2007.

[55] M. Mefenza, F. Yonga, and C. Bobda, "Design and Verification Environment for High-Performance Video-Based Embedded Systems," in *Distributed Embedded Smart Cameras*, C. Bobda and S. Velipasalar, Eds. Springer, 2014, pp. 69–90.

[56] University of Potsdam, "Potassco - Tools for Answer Set Programming," http://potassco.sourceforge.net/, 2010.

[57] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV*. Cambridge, MA: O'Reilly.

[58] M. Kushwaha and X. Koutsoukos, "3D Target Tracking in Distributed Smart Camera Networks with In-network Aggregation," in *Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras*, ser. ICDSC '10, NY, USA, 2010, pp. 25–32.

[59] S. Kang, J.-K. Paik, A. Koschan, B. R. Abidi, and M. A. Abidi, "Real-time Video Tracking using PTZ Cameras," *Sixth International Conference on Quality Control by Artificial Vision*, vol. 5132, May.

[60] L. Marchesotti, S. Piva, and C. Regazzoni, "An Agent-Based Approach for Tracking People in Indoor Complex Environments," *International Conference on Image Analysis and Processing*, vol. 0, p. 99, 2003.

[61] B. Rinner and M. Quaritsch, "Embedded Middleware for Smart Camera Networks and Sensor Fusion," *Elsevir*, 2008.

[62] N. T. Nguyen, S. Venkatesh, G. West, and H. H. Bui, "Multiple Camera Coordination in a Surveillance System," *ACTA Automatica Sinica*, vol. 29, pp. 408–422, 2003.

[63] H. Robert, F. A. Machot, P. Mahr, and C. Bobda, "Camera-based System for Tracking and Position Estimation of Humans." in *DASIP*. IEEE, 2010, pp. 62–67.

[64] M. Borkar, V. Cevher, and J. H. McClellan, "Estimating Target State Distributions in a Distributed Sensor Network using a Monte-Carlo Approach," in *IEEE MLSP 2005*, Connecticut, 28–30 September 2005.

[65] S. Velipasalar, J. Schlessman, C.-Y. Chen, W. Wolf, and J. Singh, "SCCS: A Scalable Clustered Camera System for Multiple Object Tracking Communicating via Message Passing Interface." IEEE, 2006, pp. 277–280.

[66] J. Orwell, S. Massey, P. Remagnino, D. Greenhill, and G. Jones, "A Multi-Agent Framework for Visual Surveillance," *International Conference on Image Analysis and Processing*, vol. 0, p. 1104, 1999.

[67] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A Second Generation Low Cost Embedded Color Vision System," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 136, 2005.

[68] M. Cristani, R. Raghavendra, A. Del Bue, and V. Murino, "Human Behavior Analysis in Video Surveillance: A Social Signal Processing Perspective," *Neurocomput.*, vol. 100, pp. 86–97, Jan. 2013.

[69] T. Ko, "A Survey on Behavior Analysis in Video Surveillance for Homeland Security Applications." in *AIPR*. IEEE Computer Society, 2008, pp. 1–8.

[70] M. Enzweiler and D. M. Gavrila, "Monocular Pedestrian Detection: Survey and Experiments," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 99, no. 1, 2009.